



Windows SDK Guide for Visual Studio .Net

#80152520-001

Rev. C

Revision History

Revision	Description and Reason for Change	Date
A	Initial Release - Manual;User;DotNet;SDK;Windows	2/08/2020

Contents

1	IDTech Windows SDK Reference Guide for Visual Studio Development	1
2	Using ZMQ Server/Client	2
2.1	Setting Up ZMQ Server App	2
2.2	Setting Up ZMQ Client App	3
2.3	ZMQ Server / Client Example	4
3	Using SDK on iOS/Android with MAUI	12
3.1	Setting Up iOS Platform Bindings	12
3.2	Setting Up Android Platform Bindings	13
3.3	Setting Up Main (Shared) Platform Bindings	13
3.4	Communicating with Devices	14
4	Using SDK on Android with Xamarin	16
4.1	Communicating with Android Devices	17
5	Using SDK on iOS with Xamarin	19
5.1	Communicating with iOS Devices	20
6	SDK Class Organization	22
7	Important Security Notice	25
7.1	Applicability	25
7.2	What Does PA-DSS Mean to You?	25
7.3	Third Party Applications	26
7.4	PA-DSS Guidelines	26
7.5	More Information	31
8	Main Transaction Commands	33
8.1	EMV Methods	33
8.2	MSR/CTLS	34
8.3	All Interfaces	35
9	Connecting to IDTech Devices	36

9.1	Connect with USB:	37
9.2	Connect with Serial Interface (COM)	37
9.3	Connect with TCP/IP	37
9.4	Automatically Accept Incoming TLS Secure Connections	38
9.5	Using Alternate TLS Certificates	38
10	Core Implementation: WinForms	39
10.1	Integrating with IDTechSDK.dll	39
10.2	Import the necessary libraries	39
10.2.1	Obtaining Latest SDK	39
10.3	Add using statements to utilize library	40
10.4	Implement the callback functions	40
10.5	Initialize Device:	42
10.6	Sample Source Code Info	43
11	L100 Pass-Through Mode	44
12	Virtual Device Operation	45
13	LCD Foreign Language Mapping Table	48
14	Error Code Reference	50
15	Enumeration Reference	62
16	EMV Callback	64
17	EMV Tag Reference	65
18	Namespace Index	78
18.1	Packages	78
19	Class Index	79
19.1	Class List	79
20	Namespace Documentation	80
20.1	IDTechSDK Namespace Reference	80
21	Class Documentation	81
21.1	IDTechSDK.IDT_Augusta Class Reference	81
21.1.1	Detailed Description	84
21.1.2	Member Function Documentation	84
21.1.2.1	config_getBeeperController(ref bool firmwareControlBeeper, string ident="")	84
21.1.2.2	config_getEncryptionControl(ref bool msr, ref bool icc, string ident="")	84

21.1.2.3	<code>config_getLEDController(ref bool firmwareControlMSRLED, ref bool firmwareControlICCLEd, string ident="")</code>	85
21.1.2.4	<code>config_getModelNumber(ref string response, string ident="")</code>	85
21.1.2.5	<code>config_getSerialNumber(ref string response, string ident="")</code>	86
21.1.2.6	<code>config_setBeeperController(bool firmwareControlBeeper, string ident="")</code>	86
21.1.2.7	<code>config_setCmdTimeOutDuration(int newTimeOut, string ident="")</code>	86
21.1.2.8	<code>config_setEncryptionControl(bool msr, bool icc, string ident="")</code>	86
21.1.2.9	<code>config_setLEDController(bool firmwareControlMSRLED, bool firmwareControlICCLEd=false, string ident="")</code>	87
21.1.2.10	<code>createFastEMVData(ref IDTTTransactionData cData, string ident="")</code>	87
21.1.2.11	<code>device_activateTransaction(int timeout, byte[] tags, bool forceOnline, bool isFastEMV=false, string ident="")</code>	88
21.1.2.12	<code>device_controlBeep(int index, int frequency, int duration, string ident="")</code>	88
21.1.2.13	<code>device_controlLED(byte indexLED, byte control, int intervalOn=500, int intervalOff=500, string ident="")</code>	89
21.1.2.14	<code>device_controlLED_ICC(int controlMode, int interval, string ident="")</code>	89
21.1.2.15	<code>device_controlLED_MSR(byte control, int intervalOn=500, int intervalOff=500, string ident="")</code>	89
21.1.2.16	<code>device_disableAugustaLED(bool disable, string ident="")</code>	90
21.1.2.17	<code>device_getAnyRKIStatus(bool isMultiKey, ref string status, ref Dictionary< string, RKI_KEY_TYPE > keys, string ident="")</code>	90
21.1.2.18	<code>device_getConfigurationFromMemory(ref string json, string ident="")</code>	91
21.1.2.19	<code>device_getDateTime(ref byte[] dateTime, string ident="")</code>	91
21.1.2.20	<code>device_getDRS(ref byte[] codeDRS, string ident="")</code>	91
21.1.2.21	<code>device_getFirmwareVersion(ref string response, string ident="")</code>	92
21.1.2.22	<code>device_getKeyFormatForICCDUKPT(ref byte format, string ident="")</code>	92
21.1.2.23	<code>device_getKeyStatus(ref Boolean newFormat, ref byte[] status, string ident="")</code>	92
21.1.2.24	<code>device_getKeyTypeForICCDUKPT(ref byte type, string ident="")</code>	93
21.1.2.25	<code>device_getQuickChipMode(ref bool isOn, string ident="")</code>	93
21.1.2.26	<code>device_getResponseCodeString(RETURN_CODE eCode)</code>	93
21.1.2.27	<code>device_getRKIStatus(bool isProd, bool isMultiKey, ref string status, ref Dictionary< string, RKI_KEY_TYPE > keys, string ident="")</code>	103
21.1.2.28	<code>device_getTransArmorID(ref string TID, string ident="")</code>	104
21.1.2.29	<code>device_isSRED(string ident="")</code>	104
21.1.2.30	<code>device_readConfigurationToMemory(string memo, Configs.InstallRules rules, List< Configs.DeviceCommand > cmds, string ident="", bool isForeground=false)</code>	104
21.1.2.31	<code>device_rebootDevice(string ident="")</code>	105
21.1.2.32	<code>device_RemoteKeyInjection(RKI_KEY_TYPE type, string keyName, string ident="", bool performOnForeground=false)</code>	105
21.1.2.33	<code>device_selfCheck(string ident="")</code>	105
21.1.2.34	<code>device_sendConfiguration(string filename, VC_OPERATION_TYPE type, bool matchFW, string ident="", bool isForeground=false)</code>	106

21.1.2.35 device_sendConfigurationFromZip(byte[] zip, string filename, VC_OPERATION↵ _TYPE type, bool matchFW, string ident=""), bool isForeground=false)	107
21.1.2.36 device_sendDataCommand(string cmd, bool calcLRC, ref byte[] response, string ident="")	108
21.1.2.37 device_sendDataCommand_ext(string cmd, bool calcLRC, ref byte[] response, int timeout, bool noResponse, string ident="")	108
21.1.2.38 device_sendPAE(string command, ref string response, int timeout, string ident="")	108
21.1.2.39 device_setDateTime(string ident="")	109
21.1.2.40 device_setQuickChipMode(bool turnON, string ident="")	109
21.1.2.41 device_setTransArmorEncryption(byte[] cert, string ident="")	109
21.1.2.42 device_setTransArmorID(string TID, string ident="")	109
21.1.2.43 device_StartRKI(int type, string ident="")	110
21.1.2.44 device_startRKI(string ident="")	110
21.1.2.45 device_startTransaction(double amount, double amtOther, int exponent, int type, int timeout, byte[] tags, bool forceOnline, bool isFastEMV=false, string ident="")	110
21.1.2.46 device_updateDeviceFirmware(byte[] firmwareData, string ident="")	111
21.1.2.47 device_updateFirmwareFromZip(byte[] zipfile, string ident="", bool is↵ Foreground=false)	112
21.1.2.48 device_verifyBackdoorKey(string ident="")	113
21.1.2.49 emv_activateTransaction(int timeout, byte[] tags, bool forceOnline, bool isFastE↵ MV=false, string ident="")	113
21.1.2.50 emv_addTerminalData(byte[] tlv, string ident="")	113
21.1.2.51 emv_allowFallback(bool allow, string ident="")	114
21.1.2.52 emv_authenticateTransaction(byte[] updatedTLV, string ident="")	114
21.1.2.53 emv_autoAuthenticate(bool authenticate, string ident="")	114
21.1.2.54 emv_autoAuthenticateTags(bool authenticate, byte[] tags, string ident="")	115
21.1.2.55 emv_callbackResponseLCD(EMV_LCD_DISPLAY_MODE type, byte selection, string ident="")	115
21.1.2.56 emv_callbackResponseMSR(byte[] MSR, string ident="")	115
21.1.2.57 emv_callbackResponsePIN(EMV_PIN_MODE type, byte[] KSN, byte[] PIN, string ident="")	116
21.1.2.58 emv_cancelTransaction(string ident="")	116
21.1.2.59 emv_completeTransaction(bool commError, byte[] authCode, byte[] iad, byte[] tlvScripts, byte[] tlv, string ident="")	116
21.1.2.60 emv_getEMVConfigurationCheckValue(ref string response, string ident="")	117
21.1.2.61 emv_getEMVKernelCheckValue(ref string response, string ident="")	117
21.1.2.62 emv_getEMVKernelVersion(ref string response, string ident="")	117
21.1.2.63 emv_removeAllApplicationData(string ident="")	118
21.1.2.64 emv_removeAllCAPK(string ident="")	118
21.1.2.65 emv_removeAllCRL(string ident="")	118
21.1.2.66 emv_removeApplicationData(byte[] AID, string ident="")	118
21.1.2.67 emv_removeCAPK(byte[] capk, string ident="")	119
21.1.2.68 emv_removeCRL(byte[] crlList, string ident="")	119

21.1.2.69	<code>emv_removeTerminalData(string ident="")</code>	119
21.1.2.70	<code>emv_retrieveAIDList(ref byte[] response, string ident="")</code>	120
21.1.2.71	<code>emv_retrieveApplicationData(byte[] AID, ref byte[] tlv, string ident="")</code>	120
21.1.2.72	<code>emv_retrieveCAPK(byte[] capk, ref byte[] key, string ident="")</code>	120
21.1.2.73	<code>emv_retrieveCAPKList(ref byte[] keys, string ident="")</code>	121
21.1.2.74	<code>emv_retrieveCRLList(ref byte[] list, string ident="")</code>	121
21.1.2.75	<code>emv_retrieveTerminalData(ref byte[] tlv, string ident="")</code>	122
21.1.2.76	<code>emv_retrieveTransactionResult(byte[] tags, ref IDTTransactionData tlv, string ident="")</code>	122
21.1.2.77	<code>emv_setApplicationData(byte[] name, byte[] tlv, string ident="")</code>	122
21.1.2.78	<code>emv_setCAPK(byte[] key, string ident="")</code>	122
21.1.2.79	<code>emv_setCRL(byte[] list, string ident="")</code>	123
21.1.2.80	<code>emv_setTerminalData(byte[] tlv, string ident="")</code>	123
21.1.2.81	<code>emv_setTerminalMajorConfiguration(int configuration, string ident="")</code>	124
21.1.2.82	<code>emv_startTransaction(double amount, double amtOther, int exponent, int type, int timeout, byte[] tags, bool forceOnline, string ident="")</code>	124
21.1.2.83	<code>emv_trySetTerminalData(byte[] tlv, ref byte[] rejectedTLV, ref byte[] convertedTLV, bool overwrite=false, string ident="")</code>	124
21.1.2.84	<code>getCommandTimeout(string ident="")</code>	125
21.1.2.85	<code>icc_disable(string ident="")</code>	125
21.1.2.86	<code>icc_enable(bool withNotification, string ident="")</code>	125
21.1.2.87	<code>icc_exchangeAPDU(string c_APDU, ref byte[] response, string ident="")</code>	126
21.1.2.88	<code>icc_exchangeEncryptedAPDU(string c_APDU, ref byte[] response, string ident="")</code>	126
21.1.2.89	<code>icc_getAPDU_KSN(ref byte[] ksn, string ident="")</code>	127
21.1.2.90	<code>icc_getFunctionStatus(ref bool enabled, ref bool withNotification, string ident="")</code>	127
21.1.2.91	<code>icc_getICCRReaderStatus(ref byte status, string ident="")</code>	128
21.1.2.92	<code>icc_getKeyFormatForICCDUKPT(ref byte format, string ident="")</code>	128
21.1.2.93	<code>icc_getKeyTypeForICCDUKPT(ref byte type, string ident="")</code>	128
21.1.2.94	<code>icc_getSetting(byte setting, ref byte value, string ident="")</code>	129
21.1.2.95	<code>icc_getSettings(byte setting, ref byte[] value, string ident="")</code>	129
21.1.2.96	<code>icc_powerOffICC(string ident="")</code>	129
21.1.2.97	<code>icc_powerOnICC(ref byte[] ATR, string ident="")</code>	130
21.1.2.98	<code>icc_setKeyFormatForICCDUKPT(byte encryption, string ident="")</code>	130
21.1.2.99	<code>icc_setKeyTypeForICCDUKPT(byte encryption, string ident="")</code>	130
21.1.2.100	<code>icc_setSetting(byte setting, byte value, string ident="")</code>	131
21.1.2.101	<code>icc_setSetting(byte setting, byte[] value, string ident="")</code>	131
21.1.2.102	<code>cd_retrieveMessage(DisplayMessages.DISPLAY_MESSAGE_LANGUAGE ← E lang, DisplayMessages.DISPLAY_MESSAGE_IDENTIFIER id, ref string line1, ref string line2)</code>	131
21.1.2.103	<code>msr_cancelMSRSwipe(string ident="")</code>	131
21.1.2.104	<code>msr_captureMode(bool isBufferMode, bool withNotification=false, string ident="")</code>	132

21.1.2.105	<code>msr_disable(string ident="")</code>	132
21.1.2.106	<code>msr_getClearPANID(ref byte value, string ident="")</code>	132
21.1.2.107	<code>msr_getExpirationMask(ref byte value, string ident="")</code>	133
21.1.2.108	<code>msr_getFunctionStatus(ref bool enabled, ref bool isBufferMode, ref bool withNotification, string ident="")</code>	133
21.1.2.109	<code>msr_getSetting(byte setting, ref byte value, string ident="")</code>	133
21.1.2.110	<code>msr_getSettings(byte setting, ref byte[] value, string ident="")</code>	134
21.1.2.111	<code>msr_getSwipeEncryption(ref byte encryption, string ident="")</code>	134
21.1.2.112	<code>msr_getSwipeForcedEncryptionOption(ref byte option, string ident="")</code>	134
21.1.2.113	<code>msr_getSwipeMaskOption(ref byte option, string ident="")</code>	135
21.1.2.114	<code>msr_RetrieveWhiteList(ref byte[] value, string ident="")</code>	135
21.1.2.115	<code>msr_setClearPANID(byte val, string ident="")</code>	136
21.1.2.116	<code>msr_setExpirationMask(bool mask, string ident="")</code>	136
21.1.2.117	<code>msr_setSetting(byte setting, byte value, string ident="")</code>	136
21.1.2.118	<code>msr_setSettings(byte setting, byte[] value, string ident="")</code>	136
21.1.2.119	<code>msr_setSwipeEncryption(byte encryption, string ident="")</code>	137
21.1.2.120	<code>msr_setSwipeForcedEncryptionOption(bool track1, bool track2, bool track3, bool track3card0, string ident="")</code>	137
21.1.2.121	<code>msr_setSwipeMaskOption(bool track1, bool track2, bool track3, string ident="")</code>	137
21.1.2.122	<code>msr_setWhiteList(byte[] value, byte[] MAC, string ident="")</code>	138
21.1.2.123	<code>msr_setWhiteListFromBDK(byte[] val, byte[] BDK, string ident="")</code>	138
21.1.2.124	<code>msr_startMSRSwipe(int timeout, string ident="")</code>	138
21.1.2.125	<code>msr_switchUSBInterfaceMode(bool blsUSBKeyboardMode)</code>	139
21.1.2.126	<code>msr_switchUSBInterfaceMode(bool blsUSBKeyboardMode, ref string ident)</code>	139
21.1.2.127	<code>SDK_Version()</code>	139
21.1.2.128	<code>setCallback(CallBack my_Callback)</code>	140
21.1.2.129	<code>setCallback(IntPtr my_Callback, SynchronizationContext context)</code>	140
21.1.2.130	<code>setCommandTimeout(int milliseconds, string ident="")</code>	140
21.1.3	Property Documentation	140
21.1.3.1	SharedController	140
21.2	IDTechSDK.IDT_BTMag Class Reference	141
21.2.1	Member Function Documentation	142
21.2.1.1	<code>config_getEncryptionControl(ref bool msr, ref bool icc, string ident="")</code>	142
21.2.1.2	<code>config_getLEDController(ref bool firmwareControlMSRLED, ref bool firmwareControlICCLEd, string ident="")</code>	143
21.2.1.3	<code>config_getSerialNumber(ref string response, string ident="")</code>	143
21.2.1.4	<code>config_setCmdTimeOutDuration(int newTimeOut, string ident="")</code>	144
21.2.1.5	<code>config_setEncryptionControl(bool msr, bool icc, string ident="")</code>	144
21.2.1.6	<code>config_setLEDController(bool firmwareControlMSRLED, bool firmwareControlICCLEd=false, string ident="")</code>	144
21.2.1.7	<code>device_controlBeep(int index, int frequency, int duration, string ident="")</code>	145

21.2.1.8	<code>device_controlLED(byte indexLED, byte control, int intervalOn=500, int intervalOff=500, string ident="")</code>	145
21.2.1.9	<code>device_controlLED_ICC(int controlMode, int interval, string ident="")</code>	146
21.2.1.10	<code>device_controlLED_MSR(byte control, int intervalOn=500, int intervalOff=500, string ident="")</code>	146
21.2.1.11	<code>device_getAnyRKIStatus(bool isMultiKey, ref string status, ref Dictionary< string, RKL_KEY_TYPE > keys, string ident="")</code>	147
21.2.1.12	<code>device_getConfigurationFromMemory(ref string json, string ident="")</code>	147
21.2.1.13	<code>device_getFirmwareVersion(ref string response, string ident="")</code>	148
21.2.1.14	<code>device_getKeyStatus(ref byte[] status, string ident="")</code>	148
21.2.1.15	<code>device_getResponseCodeString(RETURN_CODE eCode)</code>	148
21.2.1.16	<code>device_getRKIStatus(bool isProd, bool isMultiKey, ref string status, ref Dictionary< string, RKL_KEY_TYPE > keys, string ident="")</code>	158
21.2.1.17	<code>device_readConfigurationToMemory(string memo, Configs.InstallRules rules, List< Configs.DeviceCommand > cmds, string ident="", bool isForeground=false)</code>	159
21.2.1.18	<code>device_rebootDevice(string ident="")</code>	159
21.2.1.19	<code>device_RemoteKeyInjection(RKL_KEY_TYPE type, string keyName, string ident="", bool performOnForeground=false)</code>	159
21.2.1.20	<code>device_sendConfiguration(string filename, VC_OPERATION_TYPE type, bool matchFW, string ident="", bool isForeground=false)</code>	160
21.2.1.21	<code>device_sendConfigurationFromZip(byte[] zip, string filename, VC_OPERATION_TYPE type, bool matchFW, string ident="", bool isForeground=false)</code>	161
21.2.1.22	<code>device_sendDataCommand(string cmd, bool calcLRC, ref byte[] response, string ident="")</code>	162
21.2.1.23	<code>device_sendPAE(string command, ref string response, int timeout, string ident="")</code>	162
21.2.1.24	<code>device_setDateTime(string ident="")</code>	162
21.2.1.25	<code>device_updateFirmwareFromZip(byte[] zipfile, string ident="", bool isForeground=false)</code>	163
21.2.1.26	<code>getCommandTimeout(string ident="")</code>	163
21.2.1.27	<code>msr_cancelMSRSwipe(string ident="")</code>	163
21.2.1.28	<code>msr_captureMode(bool isBufferMode, bool withNotification=false, string ident="")</code>	163
21.2.1.29	<code>msr_disable(string ident="")</code>	165
21.2.1.30	<code>msr_getClearPANID(ref byte value, string ident="")</code>	165
21.2.1.31	<code>msr_getExpirationMask(ref byte value, string ident="")</code>	165
21.2.1.32	<code>msr_getFunctionStatus(ref bool enabled, ref bool isBufferMode, ref bool withNotification, string ident="")</code>	166
21.2.1.33	<code>msr_getSetting(byte setting, ref byte value, string ident="")</code>	166
21.2.1.34	<code>msr_getSettings(byte setting, ref byte[] value, string ident="")</code>	167
21.2.1.35	<code>msr_getSwipeEncryption(ref byte encryption, string ident="")</code>	167
21.2.1.36	<code>msr_getSwipeForcedEncryptionOption(ref byte option, string ident="")</code>	167
21.2.1.37	<code>msr_getSwipeMaskOption(ref byte option, string ident="")</code>	168
21.2.1.38	<code>msr_RetrieveWhiteList(ref byte[] value, string ident="")</code>	168
21.2.1.39	<code>msr_setClearPANID(byte val, string ident="")</code>	168

21.2.1.40	<code>msr_setExpirationMask(bool mask, string ident="")</code>	169
21.2.1.41	<code>msr_setSetting(byte setting, byte value, string ident="")</code>	169
21.2.1.42	<code>msr_setSettings(byte setting, byte[] value, string ident="")</code>	169
21.2.1.43	<code>msr_setSwipeEncryption(byte encryption, string ident="")</code>	169
21.2.1.44	<code>msr_setSwipeForcedEncryptionOption(bool track1, bool track2, bool track3, bool track3card0, string ident="")</code>	170
21.2.1.45	<code>msr_setSwipeMaskOption(bool track1, bool track2, bool track3, string ident="")</code>	170
21.2.1.46	<code>msr_setWhiteList(byte[] value, byte[] MAC, string ident="")</code>	170
21.2.1.47	<code>msr_startMSRSwipe(int timeout, string ident="")</code>	171
21.2.1.48	<code>msr_switchUSBInterfaceMode(bool blsUSBKeyboardMode)</code>	171
21.2.1.49	<code>msr_switchUSBInterfaceMode(bool blsUSBKeyboardMode, ref string ident)</code>	171
21.2.1.50	<code>SDK_Version()</code>	172
21.2.1.51	<code>setCallback(CallBack my_Callback)</code>	172
21.2.1.52	<code>setCallback(IntPtr my_Callback, SynchronizationContext context)</code>	172
21.2.1.53	<code>setCommandTimeout(int milliseconds, string ident="")</code>	173
21.2.1.54	<code>useSerialPort(int port)</code>	173
21.2.1.55	<code>useSerialPort(int port, int baud)</code>	173
21.2.1.56	<code>useSerialPortLinux(string path)</code>	173
21.2.1.57	<code>useSerialPortLinux(string path, int baud)</code>	174
21.2.2	Property Documentation	174
21.2.2.1	SharedController	174
21.3	IDTechSDK.IDT_BTPay Class Reference	174
21.3.1	Member Function Documentation	177
21.3.1.1	<code>config_getBeeperController(ref bool firmwareControlBeeper, string ident="")</code>	177
21.3.1.2	<code>config_getEncryptionControl(ref bool msr, ref bool icc, string ident="")</code>	177
21.3.1.3	<code>config_getLEDController(ref bool firmwareControlMSRLED, ref bool firmwareControlICCLEd, string ident="")</code>	178
21.3.1.4	<code>config_getModelNumber(ref string response, string ident="")</code>	178
21.3.1.5	<code>config_getSerialNumber(ref string response, string ident="")</code>	178
21.3.1.6	<code>config_setBeeperController(bool firmwareControlBeeper, string ident="")</code>	179
21.3.1.7	<code>config_setCmdTimeOutDuration(int newTimeOut, string ident="")</code>	179
21.3.1.8	<code>config_setEncryptionControl(bool msr, bool icc, string ident="")</code>	179
21.3.1.9	<code>config_setLEDController(bool firmwareControlMSRLED, bool firmwareControlICCLEd=false, string ident="")</code>	180
21.3.1.10	<code>device_controlBeep(int index, int frequency, int duration, string ident="")</code>	180
21.3.1.11	<code>device_controlLED(byte indexLED, byte control, int intervalOn=500, int intervalOff=500, string ident="")</code>	181
21.3.1.12	<code>device_controlLED_ICC(int controlMode, int interval, string ident="")</code>	181
21.3.1.13	<code>device_controlLED_MSR(byte control, int intervalOn=500, int intervalOff=500, string ident="")</code>	182
21.3.1.14	<code>device_getAnyRKIStatus(bool isMultiKey, ref string status, ref Dictionary< string, RKI_KEY_TYPE > keys, string ident="")</code>	182

21.3.1.15 device_getConfigurationFromMemory(ref string json, string ident="")	183
21.3.1.16 device_getFirmwareVersion(ref string response, string ident="")	183
21.3.1.17 device_getKeyStatus(ref byte[] status, string ident="")	183
21.3.1.18 device_getResponseCodeString(RETURN_CODE eCode)	184
21.3.1.19 device_getRKIStatus(bool isProd, bool isMultiKey, ref string status, ref Dictionary< string, RKI_KEY_TYPE > keys, string ident="")	193
21.3.1.20 device_readConfigurationToMemory(string memo, Configs.InstallRules rules, List< Configs.DeviceCommand > cmds, string ident="", bool isForeground=false)	194
21.3.1.21 device_rebootDevice(string ident="")	194
21.3.1.22 device_RemoteKeyInjection(RKI_KEY_TYPE type, string keyName, string ident="", bool performOnForeground=false)	195
21.3.1.23 device_sendConfiguration(string filename, VC_OPERATION_TYPE type, bool matchFW, string ident="", bool isForeground=false)	195
21.3.1.24 device_sendConfigurationFromZip(byte[] zip, string filename, VC_OPERATION_TYPE type, bool matchFW, string ident="", bool isForeground=false)	196
21.3.1.25 device_sendDataCommand(string cmd, bool calcLRC, ref byte[] response, string ident="")	197
21.3.1.26 device_sendPAE(string command, ref string response, int timeoutstring, string ident="")	197
21.3.1.27 device_setDateTime(string ident="")	197
21.3.1.28 device_startRKI(string ident="")	199
21.3.1.29 emv_addTerminalData(byte[] tlv, string ident="")	199
21.3.1.30 emv_allowFallback(bool allow, string ident="")	199
21.3.1.31 emv_authenticateTransaction(byte[] updatedTLV, string ident="")	200
21.3.1.32 emv_autoAuthenticate(bool authenticate, string ident="")	200
21.3.1.33 emv_autoAuthenticateTags(bool authenticate, byte[] tags, string ident="")	200
21.3.1.34 emv_callbackResponseLCD(EMV_LCD_DISPLAY_MODE type, byte selection, string ident="")	200
21.3.1.35 emv_callbackResponseMSR(byte[] MSR, string ident="")	201
21.3.1.36 emv_callbackResponsePIN(EMV_PIN_MODE type, byte[] KSN, byte[] PIN, string ident="")	201
21.3.1.37 emv_cancelTransaction(string ident="")	202
21.3.1.38 emv_completeTransaction(bool commError, byte[] authCode, byte[] iad, byte[] tlvScripts, byte[] tlv, string ident="")	202
21.3.1.39 emv_getEMVConfigurationCheckValue(ref string response, string ident="")	203
21.3.1.40 emv_getEMVKernelCheckValue(ref string response, string ident="")	203
21.3.1.41 emv_getEMVKernelVersion(ref string response, string ident="")	203
21.3.1.42 emv_removeAllApplicationData(string ident="")	203
21.3.1.43 emv_removeAllCAPK(string ident="")	204
21.3.1.44 emv_removeAllCRL(string ident="")	204
21.3.1.45 emv_removeApplicationData(byte[] AID, string ident="")	204
21.3.1.46 emv_removeCAPK(byte[] capk, string ident="")	204
21.3.1.47 emv_removeCRL(byte[] crlList, string ident="")	205

21.3.1.48 emv_removeTerminalData(string id=""")	205
21.3.1.49 emv_retrieveAIDList(ref byte[] response, string id=""")	205
21.3.1.50 emv_retrieveApplicationData(byte[] AID, ref byte[] tlv, string id=""")	206
21.3.1.51 emv_retrieveCAPK(byte[] capk, ref byte[] key, string id=""")	206
21.3.1.52 emv_retrieveCAPKList(ref byte[] keys, string id=""")	207
21.3.1.53 emv_retrieveCRLList(ref byte[] list, string id=""")	207
21.3.1.54 emv_retrieveTerminalData(ref byte[] tlv, string id=""")	207
21.3.1.55 emv_retrieveTransactionResult(byte[] tags, ref IDTTransactionData tlv, string id=""")	207
21.3.1.56 emv_setApplicationData(byte[] name, byte[] tlv, string id=""")	208
21.3.1.57 emv_setCAPK(byte[] key, string id=""")	208
21.3.1.58 emv_setCRL(byte[] list, string id=""")	209
21.3.1.59 emv_setTerminalData(byte[] tlv, string id=""")	209
21.3.1.60 emv_setTerminalMajorConfiguration(int configuration, string id=""")	209
21.3.1.61 emv_startTransaction(double amount, double amtOther, int exponent, int type, int timeout, byte[] tags, bool forceOnline, bool isFastEMV=false, string id=""")	210
21.3.1.62 emv_trySetTerminalData(byte[] tlv, ref byte[] rejectedTLV, ref byte[] convertedTLV, bool overwrite=false, string id=""")	210
21.3.1.63 getCommandTimeout(string id=""")	211
21.3.1.64 icc_disable(string id=""")	211
21.3.1.65 icc_enable(bool withNotification, string id=""")	211
21.3.1.66 icc_exchangeAPDU(string c_APDU, ref byte[] response, string id=""")	211
21.3.1.67 icc_exchangeEncryptedAPDU(string c_APDU, ref byte[] response, string id=""")	212
21.3.1.68 icc_getAPDU_KSN(ref byte[] ksn, string id=""")	212
21.3.1.69 icc_getFunctionStatus(ref bool enabled, ref bool withNotification, string id=""")	213
21.3.1.70 icc_getICCRReaderStatus(ref byte status, string id=""")	213
21.3.1.71 icc_getKeyFormatForICCDUKPT(ref byte format, string id=""")	213
21.3.1.72 icc_getKeyTypeForICCDUKPT(ref byte type, string id=""")	214
21.3.1.73 icc_powerOffICC(string id=""")	214
21.3.1.74 icc_powerOnICC(ref byte[] ATR, string id=""")	215
21.3.1.75 icc_setKeyFormatForICCDUKPT(byte encryption, string id=""")	215
21.3.1.76 icc_setKeyTypeForICCDUKPT(byte encryption, string id=""")	215
21.3.1.77 msr_cancelMSRSwipe(string id=""")	216
21.3.1.78 msr_captureMode(bool isBufferMode, bool withNotification=false, string id=""")	216
21.3.1.79 msr_disable(string id=""")	216
21.3.1.80 msr_getClearPANID(ref byte value, string id=""")	217
21.3.1.81 msr_getExpirationMask(ref byte value, string id=""")	217
21.3.1.82 msr_getFunctionStatus(ref bool enabled, ref bool isBufferMode, ref bool with← Notification, string id=""")	217
21.3.1.83 msr_getSetting(byte setting, ref byte value, string id=""")	218
21.3.1.84 msr_getSettings(byte setting, ref byte[] value, string id=""")	218

21.3.1.85	<code>msr_getSwipeEncryption(ref byte encryption, string ident="")</code>	218
21.3.1.86	<code>msr_getSwipeForcedEncryptionOption(ref byte option, string ident="")</code>	219
21.3.1.87	<code>msr_getSwipeMaskOption(ref byte option, string ident="")</code>	219
21.3.1.88	<code>msr_RetrieveWhiteList(ref byte[] value, string ident="")</code>	219
21.3.1.89	<code>msr_setClearPANID(byte val, string ident="")</code>	220
21.3.1.90	<code>msr_setExpirationMask(bool mask, string ident="")</code>	220
21.3.1.91	<code>msr_setSetting(byte setting, byte value, string ident="")</code>	220
21.3.1.92	<code>msr_setSettings(byte setting, byte[] value, string ident="")</code>	221
21.3.1.93	<code>msr_setSwipeEncryption(byte encryption, string ident="")</code>	221
21.3.1.94	<code>msr_setSwipeForcedEncryptionOption(bool track1, bool track2, bool track3, bool track3card0, string ident="")</code>	221
21.3.1.95	<code>msr_setSwipeMaskOption(bool track1, bool track2, bool track3, string ident="")</code>	222
21.3.1.96	<code>msr_setWhiteList(byte[] value, byte[] MAC, string ident="")</code>	222
21.3.1.97	<code>msr_startMSRSwipe(int timeout, string ident="")</code>	222
21.3.1.98	<code>msr_switchUSBInterfaceMode(bool blsUSBKeyboardMode)</code>	222
21.3.1.99	<code>msr_switchUSBInterfaceMode(bool blsUSBKeyboardMode, ref string ident)</code>	223
21.3.1.100	<code>SDK_Version()</code>	223
21.3.1.101	<code>setCallback(CallBack my_Callback)</code>	223
21.3.1.102	<code>setCallback(IntPtr my_Callback, SynchronizationContext context)</code>	224
21.3.1.103	<code>setCommandTimeout(int milliseconds, string ident="")</code>	224
21.3.1.104	<code>useSerialPort(int port)</code>	224
21.3.1.105	<code>useSerialPort(int port, int baud)</code>	224
21.3.1.106	<code>useSerialPortLinux(string path)</code>	225
21.3.1.107	<code>useSerialPortLinux(string path, int baud)</code>	225
21.3.2	Property Documentation	225
21.3.2.1	SharedController	225
21.4	IDTechSDK.IDT_CM100 Class Reference	226
21.4.1	Detailed Description	226
21.4.2	Member Function Documentation	226
21.4.2.1	<code>ctls_cancelTransaction()</code>	226
21.4.2.2	<code>ctls_enableL2Application(bool enable)</code>	227
21.4.2.3	<code>ctls_startTransaction(double amount, double amtOther, int exponent, int type, int timeout, byte[] tags, bool isFastEMV=false)</code>	227
21.4.2.4	<code>device_getAnyRKIStatus(bool isMultiKey, ref string status, ref Dictionary< string, RKI_KEY_TYPE > keys, string ident="")</code>	227
21.4.2.5	<code>device_getConfigurationFromMemory(ref string json, string ident="")</code>	228
21.4.2.6	<code>device_getFirmwareVersion(ref string response)</code>	228
21.4.2.7	<code>device_getRKIStatus(bool isProd, bool isMultiKey, ref string status, ref Dictionary< string, RKI_KEY_TYPE > keys, string ident="")</code>	228
21.4.2.8	<code>device_getVersion(byte version, ref string response)</code>	229

21.4.2.9	device_readConfigurationToMemory(string memo, Configs.InstallRules rules, List< Configs.DeviceCommand > cmds, string ident="","", bool isForeground=false)	229
21.4.2.10	device_RemoteKeyInjection(RKI_KEY_TYPE type, string keyName, string ident="","", bool performOnForeground=false)	230
21.4.2.11	device_sendBeep(int time)	230
21.4.2.12	device_sendConfiguration(string filename, VC_OPERATION_TYPE type, bool matchFW, string ident="","", bool isForeground=false)	230
21.4.2.13	device_sendConfigurationFromZip(byte[] zip, string filename, VC_OPERATION_TYPE type, bool matchFW, string ident="","", bool isForeground=false)	231
21.4.2.14	device_sendDataCommand(string cmd, bool calcLRC, ref byte[] response)	232
21.4.2.15	device_sendPAE(string command, ref string response, int timeout, string ip="")	232
21.4.2.16	device_setDateTime()	233
21.4.2.17	getCommandTimeout()	233
21.4.2.18	SDK_Version()	233
21.4.2.19	setCallback(CallBack my_Callback)	233
21.4.2.20	setCallback(IntPtr my_Callback, SynchronizationContext context)	233
21.4.2.21	setCommandTimeout(int milliseconds)	234
21.4.3	Property Documentation	234
21.4.3.1	SharedController	234
21.5	IDTechSDK.IDT_K100 Class Reference	234
21.5.1	Member Function Documentation	235
21.5.1.1	closeSerialPort()	235
21.5.1.2	closeUSB()	235
21.5.1.3	config_getBaudRate(ref int baud)	235
21.5.1.4	config_getModelNumber(ref string response)	236
21.5.1.5	config_setBaudRate(int baud)	236
21.5.1.6	config_setCmdTimeOutDuration(int newTimeOut)	237
21.5.1.7	device_getAnyRKIStatus(bool isMultiKey, ref string status, ref Dictionary< string, RKI_KEY_TYPE > keys, string ident="")	237
21.5.1.8	device_getConfigurationFromMemory(ref string json, string ident="")	237
21.5.1.9	device_getDateTimeString(ref string str)	237
21.5.1.10	device_getFirmwareVersion(ref string response)	238
21.5.1.11	device_getKeyStatus(ref byte[] status)	238
21.5.1.12	device_getRKIStatus(bool isProd, bool isMultiKey, ref string status, ref Dictionary< string, RKI_KEY_TYPE > keys, string ident="")	238
21.5.1.13	device_readConfigurationToMemory(string memo, Configs.InstallRules rules, List< Configs.DeviceCommand > cmds, string ident="","", bool isForeground=false)	239
21.5.1.14	device_rebootDevice()	239
21.5.1.15	device_RemoteKeyInjection(RKI_KEY_TYPE type, string keyName, string ident="","", bool performOnForeground=false)	240
21.5.1.16	device_sendConfiguration(string filename, VC_OPERATION_TYPE type, bool matchFW, string ident="","", bool isForeground=false)	240

21.5.1.17	device_sendConfigurationFromZip(byte[] zip, string filename, VC_OPERATION↵ _TYPE type, bool matchFW, string ident="","", bool isForeground=false)	241
21.5.1.18	device_sendDataCommand(string cmd, bool calcLRC, ref byte[] response)	242
21.5.1.19	device_sendDataCommand_ext(string cmd, bool calcLRC, ref byte[] response, int timeout, bool noResponse)	242
21.5.1.20	device_StartRKI(int type)	242
21.5.1.21	device_startRKI(bool isTest)	243
21.5.1.22	device_updateFirmwareFromZip(byte[] zipfile, string ident="","", bool is↵ Foreground=false)	243
21.5.1.23	getCommandTimeout()	243
21.5.1.24	pin_cancelPINEntry()	244
21.5.1.25	pin_enableKeypad()	244
21.5.1.26	pin_getEncryptedPIN(int keyType, string PAN, string message, int timeout, bool isAES=false, string ident="")	244
21.5.1.27	pin_getFunctionKey()	245
21.5.1.28	pin_sendBeep(int frequency, int duration)	245
21.5.1.29	SDK_Version()	245
21.5.1.30	setCallback(CallBack my_Callback)	246
21.5.1.31	setCallback(IntPtr my_Callback, SynchronizationContext context)	246
21.5.1.32	setCommandTimeout(int milliseconds)	246
21.5.1.33	useSerialPort(int port)	246
21.5.1.34	useSerialPort(int port, int baud)	246
21.5.1.35	useSerialPortLinux(string path)	247
21.5.1.36	useSerialPortLinux(string path, int baud)	247
21.5.2	Property Documentation	247
21.5.2.1	SharedController	247
21.6	IDTechSDK.IDT_KioskIII Class Reference	247
21.6.1	Detailed Description	250
21.6.2	Member Function Documentation	250
21.6.2.1	config_checkDUKPTKey(byte keyindex, ref byte[] val, string ident="")	250
21.6.2.2	config_getDEKVariantType(ref byte type, string ident="")	250
21.6.2.3	config_getDUKPT_DEK_Attribution(byte keyslot, ref byte mode, ref byte output↵ Mode_workingKey, ref byte variant_keyUsage, string ident="")	251
21.6.2.4	config_getDUKPT_KSN(ref byte[] ksn, string ident="")	251
21.6.2.5	config_getDUKPTEncryptionType(ref byte type, string ident="")	252
21.6.2.6	config_getKeyslot_PEK_DEK(ref byte keyslotPEK, ref byte keyslotDEK, string ident="")	252
21.6.2.7	config_getSalt_KCV(ref byte[] kcv, string ident="")	252
21.6.2.8	config_getSerialNumber(ref string response, string ident="")	253
21.6.2.9	config_setBaudRate(int baud, string ident="")	253
21.6.2.10	config_setDEKVariantType(byte type, string ident="")	253

21.6.2.11 config_setDUKPT_DEK_Attribution_AES(byte keyslot, byte workingKey, byte keyUsage, string ident="")	254
21.6.2.12 config_setDUKPT_DEK_Attribution_TDES(byte keyslot, byte outputMode, byte variant, string ident="")	254
21.6.2.13 config_setDUKPTEncryptionType(byte type, string ident="")	255
21.6.2.14 config_setKeyslot_PEK_DEK(bool isPEK, byte keyslot, string ident="")	255
21.6.2.15 config_setRKLEKeys(short keyNumber, byte[] tr31, byte[] nonce, byte[] hmac, ref byte[] kv, ref byte[] nonce_device, ref byte[] hmac_device, string ident="")	255
21.6.2.16 ctls_activateTransaction(int timeout, byte[] tags, bool isFastEMV=false, string ident="")	256
21.6.2.17 ctls_cancelTransaction(string ident="")	257
21.6.2.18 ctls_getAllConfigurationGroups(ref byte[][] response, string ident="")	257
21.6.2.19 ctls_getConfigurationGroup(int group, ref byte[] tlv, string ident="")	258
21.6.2.20 ctls_nfcCommand(byte[] nfcCmdPkt, ref byte[] response, string ident="")	258
21.6.2.21 ctls_removeAllCAPK(string ident="")	259
21.6.2.22 ctls_removeApplicationData(byte[] AID, string ident="")	260
21.6.2.23 ctls_removeCAPK(byte[] capk, string ident="")	260
21.6.2.24 ctls_removeConfigurationGroup(int group, string ident="")	260
21.6.2.25 ctls_retrieveAIDList(ref byte[][] response, string ident="")	261
21.6.2.26 ctls_retrieveApplicationData(byte[] AID, ref byte[] tlv, string ident="")	261
21.6.2.27 ctls_retrieveCAPK(byte[] capk, ref byte[] key, string ident="")	261
21.6.2.28 ctls_retrieveCAPKList(ref byte[] keys, string ident="")	262
21.6.2.29 ctls_retrieveTerminalData(ref byte[] tlv, string ident="")	262
21.6.2.30 ctls_setApplicationData(byte[] tlv, string ident="")	263
21.6.2.31 ctls_setCAPK(byte[] key, string ident="")	263
21.6.2.32 ctls_setConfigurationGroup(byte[] tlv, string ident="")	264
21.6.2.33 ctls_setDefaultConfiguration(string ident="")	264
21.6.2.34 ctls_setTerminalData(byte[] tlv, string ident="")	264
21.6.2.35 ctls_startTransaction(double amount, double amtOther, int exponent, int type, int timeout, byte[] tags, bool isFastEMV=false, string ident="")	264
21.6.2.36 ctls_trySetTerminalData(byte[] tlv, ref byte[] rejectedTLV, ref byte[] convertedTLV, string ident="")	266
21.6.2.37 ctls_updateBalance(byte statusCode, byte[] authCode, byte[] date, byte[] time, string ident="")	266
21.6.2.38 device_activateTransaction(int timeout, byte[] tags, bool isFastEMV=false, string ident="")	267
21.6.2.39 device_cancelTransaction(string ident="")	268
21.6.2.40 device_controlLED(byte indexLED, byte control, string ident="")	268
21.6.2.41 device_controlUserInterface(byte[] values, string ident="")	269
21.6.2.42 device_enablePassThrough(bool enablePassThrough, string ident="")	269
21.6.2.43 device_getAnyRKIStatus(bool isMultiKey, ref string status, ref Dictionary< string, RKI_KEY_TYPE > keys, string ident="")	270
21.6.2.44 device_getConfigurationFromMemory(ref string json, string ident="")	270

21.6.2.45 device_getFirmwareVersion(ref string response, string ident="")	270
21.6.2.46 device_getMerchantRecord(int index, ref byte[] record, string ident="")	271
21.6.2.47 device_getRKIStatus(bool isProd, bool isMultiKey, ref string status, ref Dictionary< string, RKI_KEY_TYPE > keys, string ident="")	271
21.6.2.48 device_getTransactionResults(ref IDTTransactionData results, string ident="")	271
21.6.2.49 device_pingDevice(string ident="")	272
21.6.2.50 device_pollForToken(byte seconds, ref byte card, ref byte[] serialNumber, string ident="")	272
21.6.2.51 device_readConfigurationToMemory(string memo, Configs.InstallRules rules, List< Configs.DeviceCommand > cmds, string ident="", bool isForeground=false)	273
21.6.2.52 device_rebootDevice(string ident="")	273
21.6.2.53 device_RemoteKeyInjection(RKI_KEY_TYPE type, string keyName, string ident="", bool performOnForeground=false)	273
21.6.2.54 device_retrieveAIDList(ref byte[][] response, string ident="")	274
21.6.2.55 device_sendConfiguration(string filename, VC_OPERATION_TYPE type, bool matchFW, string ident="", bool isForeground=false)	274
21.6.2.56 device_sendConfigurationFromZip(byte[] zip, string filename, VC_OPERATION_TYPE type, bool matchFW, string ident="", bool isForeground=false)	275
21.6.2.57 device_sendDataCommand(string cmd, bool calcLRC, ref byte[] response, string ident="")	276
21.6.2.58 device_sendDataCommand_ext(string cmd, bool calcLRC, ref byte[] response, int timeout, bool noResponse, string ident="")	276
21.6.2.59 device_sendPAE(string command, ref string response, int timeout, string ident="")	277
21.6.2.60 device_sendVivoCommandP2(byte command, byte subCommand, byte[] data, ref byte[] response, string ident="")	277
21.6.2.61 device_sendVivoCommandP2_ext(byte command, byte subCommand, byte[] data, ref byte[] response, int timeout, bool noResponse, string ident="")	277
21.6.2.62 device_setBurstMode(byte mode, string ident="")	278
21.6.2.63 device_setMerchantRecord(int index, bool enabled, string merchantID, string merchantURL, string ident="")	278
21.6.2.64 device_setPollMode(byte mode, string ident="")	278
21.6.2.65 device_startRKI(string ident="")	279
21.6.2.66 device_StartRKI(int type, string ident="")	279
21.6.2.67 device_startTransaction(double amount, double amtOther, int exponent, int type, int timeout, byte[] tags, bool isFastEMV=false, string ident="")	279
21.6.2.68 device_updateDeviceFirmware(byte[] firmwareData, string ident="")	281
21.6.2.69 device_updateFirmwareFromZip(byte[] zipfile, string ident="", bool isForeground=false)	282
21.6.2.70 felica_authentication(byte[] key, string ident="")	282
21.6.2.71 felica_read(byte[] serviceCode, int numBlocks, byte[] blockList, ref byte[] blocks, string ident="")	283
21.6.2.72 felica_readWithMac(int numBlocks, byte[] blockList, ref byte[] blocks, string ident="")	283
21.6.2.73 felica_requestService(byte[] nodeCode, ref byte[] response, string ident="")	283

21.6.2.74 felica_SendCommand(byte[] command, ref byte[] response, string ident="")	284
21.6.2.75 felica_write(byte[] serviceCode, int blockCount, byte[] blockList, byte[] data, ref byte[] statusFlag, string ident="")	284
21.6.2.76 felica_writeWithMac(int blockNumber, byte[] data, string ident="")	284
21.6.2.77 getCommandTimeout(string ident="")	285
21.6.2.78 lcd_retrieveMessage(DisplayMessages.DISPLAY_MESSAGE_LANGUAGE ← E lang, DisplayMessages.DISPLAY_MESSAGE_IDENTIFIER id, ref string line1, ref string line2)	285
21.6.2.79 SDK_Version()	285
21.6.2.80 setCallback(CallBack my_Callback)	285
21.6.2.81 setCallback(IntPtr my_Callback, SynchronizationContext context)	286
21.6.2.82 setCommandTimeout(int milliseconds, string ident="")	286
21.6.2.83 useSerialPort(int port)	286
21.6.2.84 useSerialPort(int port, int baud, string ident="")	286
21.6.2.85 useSerialPortLinux(string path)	287
21.6.2.86 useSerialPortLinux(string path, int baud)	287
21.6.3 Property Documentation	287
21.6.3.1 SharedController	287
21.7 IDTechSDK.IDT_L100 Class Reference	287
21.7.1 Member Function Documentation	289
21.7.1.1 closeSerialPort()	289
21.7.1.2 closeUSB(string ident="")	289
21.7.1.3 config_getBaudRate(ref int baud, string ident="")	289
21.7.1.4 config_getEthernetMACAddress(ref byte[] address, string ident="")	290
21.7.1.5 config_getModelNumber(ref string response, string ident="")	290
21.7.1.6 config_getNetworkConfiguration(ref bool isStatic, ref string address, ref string subnet, ref string gateway, ref string dns, string ident="")	290
21.7.1.7 config_getSerialNumber(ref string response, string ident="")	291
21.7.1.8 config_setBaudRate(int baud, string ident="")	291
21.7.1.9 config_setCmdTimeOutDuration(int newTimeOut, string ident="")	292
21.7.1.10 config_setEthernetMACAddress(byte[] address, string ident="")	292
21.7.1.11 config_setNetworkConfiguration(bool isStatic, string address, string subnet, string gateway, string dns, string ident="")	292
21.7.1.12 device_enterStopMode(string ident="")	292
21.7.1.13 device_getAnyRKIStatus(bool isMultiKey, ref string status, ref Dictionary< string, RKI_KEY_TYPE > keys, string ident="")	293
21.7.1.14 device_getConfigurationFromMemory(ref string json, string ident="")	293
21.7.1.15 device_getDateTime(ref byte[] dateTime, string ident="")	293
21.7.1.16 device_getFirmwareVersion(ref string response, string ident="")	294
21.7.1.17 device_getKeyStatus(ref byte[] status, string ident="")	294
21.7.1.18 device_getRKIStatus(bool isProd, bool isMultiKey, ref string status, ref Dictionary< string, RKI_KEY_TYPE > keys, string ident="")	295

21.7.1.19 device_readConfigurationToMemory(string memo, Configs.InstallRules rules, List< Configs.DeviceCommand > cmds, string ident="","", bool isForeground=false)	295
21.7.1.20 device_rebootDevice(string ident="")	295
21.7.1.21 device_RemoteKeyInjection(RKI_KEY_TYPE type, string keyName, string ident="", bool performOnForeground=false)	296
21.7.1.22 device_sendConfiguration(string filename, VC_OPERATION_TYPE type, bool matchFW, string ident="", bool isForeground=false)	296
21.7.1.23 device_sendConfigurationFromZip(byte[] zip, string filename, VC_OPERATION_TYPE type, bool matchFW, string ident="", bool isForeground=false)	297
21.7.1.24 device_sendDataCommand(string cmd, bool calcLRC, ref byte[] response, string ident="")	298
21.7.1.25 device_sendDataCommand_ext(string cmd, bool calcLRC, ref byte[] response, int timeout, bool noResponse, string ident="")	298
21.7.1.26 device_sendPAE(string command, ref string response, int timeout, string ident="")	299
21.7.1.27 device_setDateTime(string ident="")	299
21.7.1.28 device_setSleepModeTime(int time, string ident="")	299
21.7.1.29 device_StartRKI(int type, string ident="")	300
21.7.1.30 device_startRKI(string ident="")	300
21.7.1.31 device_updateDeviceFirmware(byte[] firmwareData, string ident="")	300
21.7.1.32 device_updateFirmwareFromZip(byte[] zipfile, string ident="", bool isForeground=false)	301
21.7.1.33 getCommandTimeout(string ident="")	303
21.7.1.34 lcd_clearAllLines(string ident="")	303
21.7.1.35 lcd_clearDisplay(int lineNumber, string ident="")	303
21.7.1.36 lcd_displayMessage(int lineNumber, string message, string ident="")	304
21.7.1.37 lcd_displayPrompt(int promptNumber, int lineNumber, string ident="")	304
21.7.1.38 lcd_enableBacklight(bool enable, string ident="")	304
21.7.1.39 lcd_getBacklightStatus(ref bool enabled, string ident="")	304
21.7.1.40 lcd_retrieveMessage(DisplayMessages.DISPLAY_MESSAGE_LANGUAGE lang, DisplayMessages.DISPLAY_MESSAGE_IDENTIFIER id, ref string line1, ref string line2)	305
21.7.1.41 lcd_savePrompt(int promptNumber, string prompt, string ident="")	305
21.7.1.42 pin_cancelPINEntry(string ident="")	305
21.7.1.43 pin_getEncryptedPIN(int keyType, string PAN, string message, int timeout, bool isAES=false, string ident="")	306
21.7.1.44 pin_getFunctionKey(string ident="")	306
21.7.1.45 pin_getManualPanEntry(bool csc, bool ADR, bool ZIP, string ident="")	308
21.7.1.46 pin_promptForAmountInput(string message, byte[] signature, bool maskInput, int minLen, int maxLen, string ident="")	308
21.7.1.47 pin_promptForAmountInput(int messageID, int languageID, int minLen, int maxLen)	308
21.7.1.48 pin_promptForKeyInput(string message, byte[] signature, bool maskInput, int minLen, int maxLen, string ident="")	311
21.7.1.49 pin_promptForKeyInput(int messageID, int languageID, bool maskInput, int minLen, int maxLen)	313

21.7.1.50	<code>pin_sendBeep(int frequency, int duration, string id=""")</code>	316
21.7.1.51	<code>pin_setKeyPressCapture(bool showKeyValue, string id=""")</code>	316
21.7.1.52	<code>SDK_Version()</code>	316
21.7.1.53	<code>setCallback(CallBack my_Callback)</code>	316
21.7.1.54	<code>setCallback(IntPtr my_Callback, SynchronizationContext context)</code>	317
21.7.1.55	<code>setCommandTimeout(int milliseconds, string id=""")</code>	317
21.7.1.56	<code>useSerialPort(int port)</code>	317
21.7.1.57	<code>useSerialPort(int port, int baud, string id=""")</code>	317
21.7.1.58	<code>useSerialPortLinux(string path)</code>	317
21.7.1.59	<code>useSerialPortLinux(string path, int baud)</code>	318
21.7.2	Property Documentation	318
21.7.2.1	SharedController	318
21.8	IDTechSDK.IDT_L80 Class Reference	318
21.8.1	Member Function Documentation	320
21.8.1.1	<code>closeSerialPort()</code>	320
21.8.1.2	<code>closeUSB(string id=""")</code>	320
21.8.1.3	<code>config_getBaudRate(ref int baud, string id=""")</code>	320
21.8.1.4	<code>config_getModelNumber(ref string response, string id=""")</code>	321
21.8.1.5	<code>config_getSerialNumber(ref string response, string id=""")</code>	321
21.8.1.6	<code>config_setBaudRate(int baud, string id=""")</code>	321
21.8.1.7	<code>config_setCmdTimeOutDuration(int newTimeOut, string id=""")</code>	322
21.8.1.8	<code>device_enterStopMode(string id=""")</code>	322
21.8.1.9	<code>device_getAnyRKIStatus(bool isMultiKey, ref string status, ref Dictionary< string, RKI_KEY_TYPE > keys, string id=""")</code>	322
21.8.1.10	<code>device_getConfigurationFromMemory(ref string json, string id=""")</code>	323
21.8.1.11	<code>device_getDateTime(ref byte[] dateTime, string id=""")</code>	323
21.8.1.12	<code>device_getFirmwareVersion(ref string response, string id=""")</code>	323
21.8.1.13	<code>device_getKeyStatus(ref byte[] status, string id=""")</code>	324
21.8.1.14	<code>device_getRKIStatus(bool isProd, bool isMultiKey, ref string status, ref Dictionary< string, RKI_KEY_TYPE > keys, string id=""")</code>	324
21.8.1.15	<code>device_PKI_RKI(bool isProduction, string id=""", bool performOn←Foreground=false)</code>	324
21.8.1.16	<code>device_readConfigurationToMemory(string memo, Configs.InstallRules rules, List< Configs.DeviceCommand > cmds, string id=""", bool isForeground=false)</code>	325
21.8.1.17	<code>device_rebootDevice(string id=""")</code>	325
21.8.1.18	<code>device_RemoteKeyInjection(RKI_KEY_TYPE type, string keyName, string id=""", bool performOnForeground=false)</code>	325
21.8.1.19	<code>device_sendConfiguration(string filename, VC_OPERATION_TYPE type, bool matchFW, string id=""", bool isForeground=false)</code>	326
21.8.1.20	<code>device_sendConfigurationFromZip(byte[] zip, string filename, VC_OPERATION←_TYPE type, bool matchFW, string id=""", bool isForeground=false)</code>	327

21.8.1.21	<code>device_sendDataCommand(string cmd, bool calcLRC, ref byte[] response, string ident="")</code>	328
21.8.1.22	<code>device_sendDataCommand_ext(string cmd, bool calcLRC, ref byte[] response, int timeout, bool noResponse, string ident="")</code>	328
21.8.1.23	<code>device_sendPAE(string command, ref string response, int timeout, string ident="")</code>	328
21.8.1.24	<code>device_setDateTime(string ident="")</code>	329
21.8.1.25	<code>device_setSleepModeTime(int time, string ident="")</code>	329
21.8.1.26	<code>device_updateDeviceFirmware(byte[] firmwareData, string ident="")</code>	329
21.8.1.27	<code>device_updateFirmwareFromZip(byte[] zipfile, string ident="", bool isForeground=false)</code>	331
21.8.1.28	<code>getCommandTimeout(string ident="")</code>	331
21.8.1.29	<code>lcd_clearAllLines(string ident="")</code>	331
21.8.1.30	<code>lcd_clearDisplay(int lineNumber, string ident="")</code>	331
21.8.1.31	<code>lcd_displayMessage(int lineNumber, string message, string ident="")</code>	332
21.8.1.32	<code>lcd_displayPrompt(int promptNumber, int lineNumber, string ident="")</code>	332
21.8.1.33	<code>lcd_enableBacklight(bool enable, string ident="")</code>	332
21.8.1.34	<code>lcd_getBacklightStatus(ref bool enabled, string ident="")</code>	333
21.8.1.35	<code>lcd_retrieveMessage(DisplayMessages.DISPLAY_MESSAGE_LANGUAGE lang, DisplayMessages.DISPLAY_MESSAGE_IDENTIFIER id, ref string line1, ref string line2)</code>	333
21.8.1.36	<code>lcd_savePrompt(int promptNumber, string prompt, string ident="")</code>	333
21.8.1.37	<code>pin_cancelPINEntry(string ident="")</code>	334
21.8.1.38	<code>pin_getEncryptedPIN(int keyType, string PAN, string message, int timeout, bool isAES=false, string ident="")</code>	334
21.8.1.39	<code>pin_getFunctionKey(string ident="")</code>	334
21.8.1.40	<code>pin_getManualPanEntry(bool csc, bool ADR, bool ZIP, string ident="")</code>	335
21.8.1.41	<code>pin_promptForAmountInput(string message, byte[] signature, bool maskInput, int minLen, int maxLen, string ident="")</code>	335
21.8.1.42	<code>pin_promptForKeyInput(string message, byte[] signature, bool maskInput, int minLen, int maxLen, string ident="")</code>	336
21.8.1.43	<code>pin_sendBeep(int frequency, int duration, string ident="")</code>	337
21.8.1.44	<code>pin_setKeypressCapture(bool showKeyValue, string ident="")</code>	337
21.8.1.45	<code>SDK_Version()</code>	337
21.8.1.46	<code>setCallback(CallBack my_Callback)</code>	338
21.8.1.47	<code>setCallback(IntPtr my_Callback, SynchronizationContext context)</code>	338
21.8.1.48	<code>setCommandTimeout(int milliseconds, string ident="")</code>	338
21.8.1.49	<code>useSerialPort(int port)</code>	338
21.8.1.50	<code>useSerialPort(int port, int baud, string ident="")</code>	338
21.8.1.51	<code>useSerialPortLinux(string path)</code>	339
21.8.1.52	<code>useSerialPortLinux(string path, int baud)</code>	339
21.8.2	Property Documentation	339
21.8.2.1	SharedController	339

21.9 IDTechSDK.IDT_MiniSmartII Class Reference	339
21.9.1 Detailed Description	342
21.9.2 Member Function Documentation	342
21.9.2.1 config_getModelNumber(ref string response, string ident="")	342
21.9.2.2 config_getSerialNumber(ref string response, string ident="")	342
21.9.2.3 config_setCmdTimeOutDuration(int newTimeOut, string ident="")	342
21.9.2.4 createFastEMVData(ref IDTTransactionData cData, string ident="")	342
21.9.2.5 device_activateTransaction(int timeout, byte[] tags, bool forceOnline, bool isFastEMV=false, string ident="")	343
21.9.2.6 device_enableSecureHeadForMSII(string ident="")	343
21.9.2.7 device_getAnyRKIStatus(bool isMultiKey, ref string status, ref Dictionary< string, RKI_KEY_TYPE > keys, string ident="")	344
21.9.2.8 device_getConfigurationFromMemory(ref string json, string ident="")	344
21.9.2.9 device_getDateTime(ref byte[] dateTime, string ident="")	344
21.9.2.10 device_getFirmwareVersion(ref string response, string ident="")	345
21.9.2.11 device_getKeyTypeForICCDUKPT(ref byte type, string ident="")	345
21.9.2.12 device_getResponseCodeString(RETURN_CODE eCode)	346
21.9.2.13 device_getRKIStatus(bool isProd, bool isMultiKey, ref string status, ref Dictionary< string, RKI_KEY_TYPE > keys, string ident="")	355
21.9.2.14 device_readConfigurationToMemory(string memo, Configs.InstallRules rules, List< Configs.DeviceCommand > cmds, string ident="", bool isForeground=false)	356
21.9.2.15 device_rebootDevice(string ident="")	356
21.9.2.16 device_RemoteKeyInjection(RKI_KEY_TYPE type, string keyName, string ident="", bool performOnForeground=false)	357
21.9.2.17 device_sendConfiguration(string filename, VC_OPERATION_TYPE type, bool matchFW, string ident="", bool isForeground=false)	357
21.9.2.18 device_sendConfigurationFromZip(byte[] zip, string filename, VC_OPERATION_TYPE type, bool matchFW, string ident="", bool isForeground=false)	358
21.9.2.19 device_sendDataCommand(string cmd, bool calcLRC, ref byte[] response, string ident="")	359
21.9.2.20 device_sendDataCommand_ext(string cmd, bool calcLRC, ref byte[] response, int timeout, bool noResponse, string ident="")	359
21.9.2.21 device_sendPAE(string command, ref string response, int timeout, string ident="")	360
21.9.2.22 device_setDateTime(string ident="")	360
21.9.2.23 device_StartRKI(int type, string ident="")	360
21.9.2.24 device_startRKI(string ident="")	360
21.9.2.25 device_startTransaction(double amount, double amtOther, int exponent, int type, int timeout, byte[] tags, bool forceOnline, bool isFastEMV=false, string ident="")	361
21.9.2.26 device_updateDeviceFirmware(byte[] firmwareData, string ident="")	361
21.9.2.27 device_updateFirmwareFromZip(byte[] zipfile, string ident="", bool isForeground=false)	363
21.9.2.28 emv_activateTransaction(int timeout, byte[] tags, bool forceOnline, bool isFastEMV=false, string ident="")	363
21.9.2.29 emv_addTerminalData(byte[] tlv, string ident="")	364

21.9.2.30	<code>emv_allowFallback(bool allow, string ident="")</code>	364
21.9.2.31	<code>emv_authenticateTransaction(byte[] updatedTLV, string ident="")</code>	364
21.9.2.32	<code>emv_autoAuthenticate(bool authenticate, string ident="")</code>	364
21.9.2.33	<code>emv_autoAuthenticateTags(bool authenticate, byte[] tags, string ident="")</code>	365
21.9.2.34	<code>emv_callbackResponseLCD(EMV_LCD_DISPLAY_MODE type, byte selection, string ident="")</code>	365
21.9.2.35	<code>emv_callbackResponseMSR(byte[] MSR, string ident="")</code>	365
21.9.2.36	<code>emv_callbackResponsePIN(EMV_PIN_MODE type, byte[] KSN, byte[] PIN, string ident="")</code>	366
21.9.2.37	<code>emv_cancelTransaction(string ident="")</code>	366
21.9.2.38	<code>emv_completeTransaction(bool commError, byte[] authCode, byte[] iad, byte[] tlvScripts, byte[] tlv, string ident="")</code>	366
21.9.2.39	<code>emv_getEMVConfigurationCheckValue(ref string response, string ident="")</code>	367
21.9.2.40	<code>emv_getEMVKernelCheckValue(ref string response, string ident="")</code>	367
21.9.2.41	<code>emv_getEMVKernelVersion(ref string response, string ident="")</code>	367
21.9.2.42	<code>emv_removeAllApplicationData(string ident="")</code>	368
21.9.2.43	<code>emv_removeAllCAPK(string ident="")</code>	368
21.9.2.44	<code>emv_removeAllCRL(string ident="")</code>	368
21.9.2.45	<code>emv_removeApplicationData(byte[] AID, string ident="")</code>	368
21.9.2.46	<code>emv_removeCAPK(byte[] capk, string ident="")</code>	369
21.9.2.47	<code>emv_removeCRL(byte[] crlList, string ident="")</code>	369
21.9.2.48	<code>emv_removeTerminalData(string ident="")</code>	369
21.9.2.49	<code>emv_retrieveAIDList(ref byte[][] response, string ident="")</code>	370
21.9.2.50	<code>emv_retrieveApplicationData(byte[] AID, ref byte[] tlv, string ident="")</code>	370
21.9.2.51	<code>emv_retrieveCAPK(byte[] capk, ref byte[] key, string ident="")</code>	370
21.9.2.52	<code>emv_retrieveCAPKList(ref byte[] keys, string ident="")</code>	371
21.9.2.53	<code>emv_retrieveCRLList(ref byte[] list, string ident="")</code>	371
21.9.2.54	<code>emv_retrieveTerminalData(ref byte[] tlv, string ident="")</code>	372
21.9.2.55	<code>emv_retrieveTransactionResult(byte[] tags, ref IDTTransactionData tlv, string ident="")</code>	372
21.9.2.56	<code>emv_setApplicationData(byte[] name, byte[] tlv, string ident="")</code>	372
21.9.2.57	<code>emv_setCAPK(byte[] key, string ident="")</code>	373
21.9.2.58	<code>emv_setCRL(byte[] list, string ident="")</code>	373
21.9.2.59	<code>emv_setTerminalData(byte[] tlv, string ident="")</code>	373
21.9.2.60	<code>emv_setTerminalMajorConfiguration(int configuration, string ident="")</code>	374
21.9.2.61	<code>emv_startTransaction(double amount, double amtOther, int exponent, int type, int timeout, byte[] tags, bool forceOnline, bool isFastEMV=false, string ident="")</code>	374
21.9.2.62	<code>emv_trySetTerminalData(byte[] tlv, ref byte[] rejectedTLV, ref byte[] convertedTLV, bool overwrite=false, string ident="")</code>	375
21.9.2.63	<code>getCommandTimeout(string ident="")</code>	375
21.9.2.64	<code>icc_exchangeAPDU(string c_APDU, ref byte[] response, string ident="")</code>	375
21.9.2.65	<code>icc_getAPDU_KSN(ref byte[] ksn, string ident="")</code>	376

21.9.2.66	icc_getClearPANID(ref byte prePAN, ref byte postPAN, string id=""")	376
21.9.2.67	icc_getICCRReaderStatus(ref byte status, string id=""")	376
21.9.2.68	icc_getKeyFormatForICCDUKPT(ref byte format, string id=""")	377
21.9.2.69	icc_getKeyTypeForICCDUKPT(ref byte type, string id=""")	377
21.9.2.70	icc_getReadingCharacteristics(ref string mode, string id=""")	377
21.9.2.71	icc_powerOffICC(string id=""")	378
21.9.2.72	icc_powerOnICC(ref byte[] ATR, string id=""")	378
21.9.2.73	icc_setClearPANID(byte prePAN, byte postPAN, string id=""")	378
21.9.2.74	icc_setKeyFormatForICCDUKPT(byte encryption, string id=""")	379
21.9.2.75	icc_setKeyTypeForICCDUKPT(byte encryption, string id=""")	379
21.9.2.76	icc_setReadingCharacteristics(int mode, string id=""")	379
21.9.2.77	lcd_retrieveMessage(DisplayMessages.DISPLAY_MESSAGE_LANGUAGE← E lang, DisplayMessages.DISPLAY_MESSAGE_IDENTIFIER id, ref string line1, ref string line2)	380
21.9.2.78	SDK_Version()	380
21.9.2.79	setCallback(CallBack my_Callback)	381
21.9.2.80	setCallback(IntPtr my_Callback, SynchronizationContext context)	381
21.9.2.81	setCommandTimeout(int milliseconds, string id=""")	381
21.9.2.82	useSerialPort(int port)	381
21.9.2.83	useSerialPort(int port, int baud)	381
21.9.2.84	useSerialPortLinux(string path)	382
21.9.2.85	useSerialPortLinux(string path, int baud)	382
21.9.3	Property Documentation	382
21.9.3.1	SharedController	382
21.10	IDTechSDK.IDT_NEO2 Class Reference	382
21.10.1	Detailed Description	390
21.10.2	Member Function Documentation	390
21.10.2.1	adf_ApplicationControl(ADF_APP_CONTROL state, string id=""")	390
21.10.2.2	adf_eraseFlash(ADF_TYPE type, string id=""")	391
21.10.2.3	adf_getADFMode(ref bool enable, string id=""")	391
21.10.2.4	adf_getModuleBytes(ADF_TYPE type, ref List< byte[] > adfInfo, string id=""")	391
21.10.2.5	adf_getModuleInfo(ADF_TYPE type, ref List< ADF_Info > adfInfo, string id=""")	392
21.10.2.6	adf_setADFMode(bool enable, string id=""")	392
21.10.2.7	adf_setJTAG(bool enable, string id=""")	392
21.10.2.8	closeSocket(string IP)	392
21.10.2.9	config_checkDUKPTKey(byte keyindex, ref byte[] val, string id=""")	393
21.10.2.10	config_getDEKVariantType(ref byte type, string id=""")	393
21.10.2.11	config_getDUKPT_DEK_Attribution(byte keyslot, ref byte mode, ref byte output← Mode_workingKey, ref byte variant_keyUsage, string id=""")	394
21.10.2.12	config_getDUKPT_KSN(ref byte[] ksn, string id=""")	394
21.10.2.13	config_getDUKPTEncryptionType(ref byte type, string id=""")	395

21.10.2.14	<code>config_getEncryptionControl(ref bool msr, ref bool icc, string ident="")</code>	395
21.10.2.15	<code>config_getEthernetMACAddress(ref byte[] address, string ident="")</code>	395
21.10.2.16	<code>config_getKeyslot_PEK_DEK(ref byte keyslotPEK, ref byte keyslotDEK, string ident="")</code>	396
21.10.2.17	<code>config_getKeySlotInfo(int index, int slot, ref byte[] keyslot, string ident="")</code>	396
21.10.2.18	<code>config_getMasking(ref byte prePAN, ref byte postPAN, ref byte asciiMask, ref byte hexMask, ref bool maskExp, string ident="")</code>	397
21.10.2.19	<code>config_getNetworkConfiguration(ref bool isStatic, ref string address, ref string subnet, ref string gateway, ref string dns, string ident="")</code>	397
21.10.2.20	<code>config_getSalt_KCV(ref byte[] kcv, string ident="")</code>	397
21.10.2.21	<code>config_getSerialNumber(ref string response, string ident="")</code>	398
21.10.2.22	<code>config_getSSLServerEthernet(ref string address, ref int port, string ident="")</code>	398
21.10.2.23	<code>config_getSwipeandDone(ref byte swipeVal, ref byte doneVal, ref byte delay, string ident="")</code>	398
21.10.2.24	<code>config_getTrackFormat(ref byte option, string ident="")</code>	399
21.10.2.25	<code>config_getWhiteList(ref Dictionary< string, string > data, string ident="")</code>	399
21.10.2.26	<code>config_getWifiConfig(ref int mode, ref string ssid, ref string password, ref string ip, ref string netMask, ref string gateway, ref string remoteIP, ref string remotePort, string ident="")</code>	399
21.10.2.27	<code>config_getWirelessWorkMode(ref int mode, string ident="")</code>	400
21.10.2.28	<code>config_sendSSLRequestWiFi(string address, int port, string ident="")</code>	400
21.10.2.29	<code>config_setBaudRate(int baud, string ident="")</code>	400
21.10.2.30	<code>config_setBluetoothParameters(string name, string oldPW, string newPW, string ident="")</code>	401
21.10.2.31	<code>config_setDEKVariantType(byte type, string ident="")</code>	401
21.10.2.32	<code>config_setDUKPT_DEK_Attribution_AES(byte keyslot, byte workingKey, byte keyUsage, string ident="")</code>	401
21.10.2.33	<code>config_setDUKPT_DEK_Attribution_TDES(byte keyslot, byte outputMode, byte variant, string ident="")</code>	402
21.10.2.34	<code>config_setDUKPTEncryptionType(byte type, string ident="")</code>	402
21.10.2.35	<code>config_setEncryptionControl(bool msr, bool icc, string ident="")</code>	403
21.10.2.36	<code>config_setKeyslot_PEK_DEK(bool isPEK, byte keyslot, string ident="")</code>	403
21.10.2.37	<code>config_setMasking(byte prePAN, byte postPAN, byte asciiMask, byte hexMask, bool maskExp, string ident="")</code>	404
21.10.2.38	<code>config_setNetworkConfiguration(bool isStatic, string address, string subnet, string gateway, string dns, string ident="")</code>	404
21.10.2.39	<code>config_setRKLEKeys(short keyNumber, byte[] tr31, byte[] nonce, byte[] hmac, ref byte[] kv, ref byte[] nonce_device, ref byte[] hmac_device, string ident="")</code>	405
21.10.2.40	<code>config_setSSLServerEthernet(string address, int port, string ident="")</code>	405
21.10.2.41	<code>config_setSwipeandDone(byte swipeVal, byte doneVal, byte delay, string ident="")</code>	405
21.10.2.42	<code>config_setTrackFormat(byte option, string ident="")</code>	406
21.10.2.43	<code>config_setWhiteList(byte[] data, string ident="")</code>	406
21.10.2.44	<code>config_setWifiConfig(int mode, string ssid, string password, string ip, string netMask, string gateway, string ident="", string remoteIP=null, string remotePort=null)</code>	406

21.10.2.45	config_setWirelessWorkMode(int mode, string ident="")	407
21.10.2.46	createFastEMVData(ref IDTTransactionData cData, string ident="")	407
21.10.2.47	ctls_activateTransaction(int timeout, byte[] tags, bool forceOnline, bool isFastEMV← MV=false, string ident="")	408
21.10.2.48	ctls_cancelTransaction(string ident="")	409
21.10.2.49	ctls_getAllConfigurationGroups(ref byte[][] response, string ident="")	409
21.10.2.50	ctls_getConfigurationGroup(int group, ref byte[] tlv, string ident="")	409
21.10.2.51	ctls_nfcCommand(byte[] nfcCmdPkt, ref byte[] response, string ident="")	410
21.10.2.52	ctls_removeAllCAPK(string ident="")	411
21.10.2.53	ctls_removeApplicationData(byte[] AID, string ident="")	411
21.10.2.54	ctls_removeCAPK(byte[] capk, string ident="")	411
21.10.2.55	ctls_removeConfigurationGroup(int group, string ident="")	411
21.10.2.56	ctls_resetConfigurationGroup(int group, string ident="")	412
21.10.2.57	ctls_retrieveAIDList(ref byte[][] response, string ident="")	412
21.10.2.58	ctls_retrieveApplicationData(byte[] AID, ref byte[] tlv, string ident="")	412
21.10.2.59	ctls_retrieveCAPK(byte[] capk, ref byte[] key, string ident="")	413
21.10.2.60	ctls_retrieveCAPKList(ref byte[] keys, string ident="")	413
21.10.2.61	ctls_retrieveTerminalData(ref byte[] tlv, string ident="")	413
21.10.2.62	ctls_setApplicationData(byte[] tlv, string ident="")	414
21.10.2.63	ctls_setCAPK(byte[] key, string ident="")	415
21.10.2.64	ctls_setConfigurationGroup(byte[] tlv, string ident="")	415
21.10.2.65	ctls_setDefaultConfiguration(string ident="")	415
21.10.2.66	ctls_setTerminalData(byte[] tlv, string ident="")	416
21.10.2.67	ctls_startTransaction(double amount, double amtOther, int exponent, int type, int timeout, byte[] tags, bool forceOnline, bool isFastEMV=false, string ident="")	416
21.10.2.68	ctls_trySetTerminalData(byte[] tlv, ref byte[] rejectedTLV, ref byte[] convertedTLV, string ident="")	417
21.10.2.69	ctls_updateBalance(byte statusCode, byte[] authCode, byte[] date, byte[] time, string ident="")	418
21.10.2.70	device_activateTransaction(int timeout, byte[] tags, bool isFastEMV=false, string ident="")	418
21.10.2.71	device_buzzer(string ident="")	419
21.10.2.72	device_buzzerOnOff(string ident="")	420
21.10.2.73	device_cancelTransaction(string ident="")	420
21.10.2.74	device_certificateType(ref int type, string ident="")	420
21.10.2.75	device_controlLED(byte indexLED, byte control, string ident="")	421
21.10.2.76	device_controlUserInterface(byte[] values, string ident="")	421
21.10.2.77	device_deleteDirectory(string filename, string ident="")	423
21.10.2.78	device_deleteFile(string filename, bool isSD=false, string ident="")	423
21.10.2.79	device_disBlueLED(string ident="")	423
21.10.2.80	device_enableL100PassThrough(bool enablePassThrough, string ident="")	424

21.10.2.81	<code>device_enableL80PassThrough(bool enablePassThrough, string ident="")</code>	424
21.10.2.82	<code>device_enablePassThrough(bool enablePassThrough, string ident="")</code>	424
21.10.2.83	<code>device_enableBlueLED(byte[] dataCmd, string ident="")</code>	425
21.10.2.84	<code>device_enterStandbyMode(string ident="")</code>	425
21.10.2.85	<code>device_extendedErrorCondition(bool enable, string ident="")</code>	425
21.10.2.86	<code>device_get1050BootloaderVersion(ref string response, string ident="")</code>	426
21.10.2.87	<code>device_get1050FuseStatus(ref byte[] status, string ident="")</code>	426
21.10.2.88	<code>device_getAnyRKIStatus(bool isMultiKey, ref string status, ref Dictionary< string, RKI_KEY_TYPE > keys, string ident="")</code>	426
21.10.2.89	<code>device_getAudioVolume(ref int volume, string ident="")</code>	426
21.10.2.90	<code>device_getBatteryVoltage(ref string voltage, string ident="")</code>	427
21.10.2.91	<code>device_getBLEName(ref string name, string ident="")</code>	427
21.10.2.92	<code>device_getBootloaderVersion(ref string response, string ident="")</code>	428
21.10.2.93	<code>device_getCashTranRiskPara(ref byte[] tlv, string ident="")</code>	428
21.10.2.94	<code>device_getConfigurationFromMemory(ref string json, string ident="")</code>	428
21.10.2.95	<code>device_getDeviceTime(ref DateTime time, string ident="")</code>	428
21.10.2.96	<code>device_getDeviceTreeVersion(ref string response, bool isDeviceTree, string ident="")</code>	429
21.10.2.97	<code>device_getDRS(ref byte[] codeDRS, string ident="")</code>	429
21.10.2.98	<code>device_getFirmwareVersion(ref string response, string ident="")</code>	429
21.10.2.99	<code>device_getHardwareInfor(ref string ascii, string ident="")</code>	430
21.10.2.100	<code>device_getLightSensorVal(ref UInt16 lightVal, string ident="")</code>	430
21.10.2.101	<code>device_getMerchantRecord(int index, ref byte[] record, string ident="")</code>	430
21.10.2.102	<code>device_getModuleVer(ref string moduleVer, string ident="")</code>	431
21.10.2.103	<code>device_getMsrSecurePar(bool b0, bool b1, bool b2, bool b3, ref byte[] tlv, string ident="")</code>	431
21.10.2.104	<code>device_getProcessorType(ref byte[] type, string ident="")</code>	431
21.10.2.105	<code>device_getProductType(ref byte[] type, string ident="")</code>	432
21.10.2.106	<code>device_getRemoteKeyInjectionTO(ref int timeout, string ident="")</code>	432
21.10.2.107	<code>device_getResponseCodeString(RETURN_CODE eCode)</code>	432
21.10.2.108	<code>device_getRKIStatus(bool isProd, bool isMultiKey, ref string status, ref Dictionary< string, RKI_KEY_TYPE > keys, string ident="")</code>	442
21.10.2.109	<code>device_getRT1050FirmwareVersion(ref string response, string ident="")</code>	443
21.10.2.110	<code>device_getSelfCheckTime(ref byte hour, ref byte minutes, string ident="")</code>	443
21.10.2.111	<code>device_getTransactionResults(ref IDTTransactionData results, string ident="")</code>	443
21.10.2.112	<code>device_getTransArmorID(ref string TID, string ident="")</code>	444
21.10.2.113	<code>device_getUIDofMCU(ref string uid, string ident="")</code>	444
21.10.2.114	<code>device_getUsbBootLoader(ref string bootLoader, string ident="")</code>	444
21.10.2.115	<code>device_listDirectory(string directoryName, bool recursive, bool onSD, ref string directory, string ident="")</code>	444
21.10.2.116	<code>device_listenForNotifications(bool enable, string ident="")</code>	445

21.10.2.117	<code>device_loadCertCA(byte certType, byte[] certData, string ident="")</code>	445
21.10.2.118	<code>device_logClear(string ident="")</code>	445
21.10.2.119	<code>device_logEnable(bool enable, string ident="")</code>	446
21.10.2.120	<code>device_logRead(DeviceLogCallback callback, string ident="")</code>	446
21.10.2.121	<code>device_lowPowerMode(bool stopMode, bool wakeOnTrans, string ident="")</code>	446
21.10.2.122	<code>device_offYellowLED(string ident="")</code>	447
21.10.2.123	<code>device_onYellowLED(string ident="")</code>	447
21.10.2.124	<code>device_pingDevice(string ident="")</code>	447
21.10.2.125	<code>device_playAudio(string name, int type, string ident="")</code>	447
21.10.2.126	<code>device_pollForToken(byte seconds, ref byte card, ref byte[] serialNumber, string ident="")</code>	448
21.10.2.127	<code>device_queryFile(string directory, string filename, bool isSD, ref bool exists, ref string timestamp, ref int fileSize, string ident="")</code>	449
21.10.2.128	<code>device_readConfigurationToMemory(string memo, Configs.InstallRules rules, List< Configs.DeviceCommand > cmds, string ident="", bool isForeground=false)</code>	449
21.10.2.129	<code>device_readFileFromSD(string directory, string filename, ref byte[] fileData, string ident="")</code>	450
21.10.2.130	<code>device_rebootDevice(string ident="")</code>	450
21.10.2.131	<code>device_RemoteKeyInjection(RKI_KEY_TYPE type, string keyName, string ident="", bool performOnForeground=false)</code>	451
21.10.2.132	<code>device_resetConfigurationGroup(int group, string ident="")</code>	451
21.10.2.133	<code>device_resetNVM(string ident="")</code>	451
21.10.2.134	<code>device_resetTransaction(string ident="")</code>	452
21.10.2.135	<code>device_retrieveAIDList(ref byte[][] response, string ident="")</code>	452
21.10.2.136	<code>device_retrieveTerminalData(ref byte[] tlv, string ident="")</code>	452
21.10.2.137	<code>device_rrcConnect(string ident="")</code>	452
21.10.2.138	<code>device_rrcDisconnect(string ident="")</code>	453
21.10.2.139	<code>device_rrcDownloadApp(string appName, byte[] appData, string ident="")</code>	453
21.10.2.140	<code>device_rrcInstallApp(string appName, string ident="")</code>	453
21.10.2.141	<code>device_rrcRunApp(string appName, string ident="")</code>	454
21.10.2.142	<code>device_rrcUninstallApp(string appName, string ident="")</code>	454
21.10.2.143	<code>device_sendConfiguration(string filename, VC_OPERATION_TYPE type, bool matchFW, string ident="", bool isForeground=false)</code>	454
21.10.2.144	<code>device_sendConfigurationFromZip(byte[] zip, string filename, VC_OPERATION_TYPE type, bool matchFW, string ident="", bool isForeground=false)</code>	455
21.10.2.145	<code>device_sendDataCommand(string cmd, bool calcLRC, ref byte[] response, string ident="")</code>	456
21.10.2.146	<code>device_sendDataCommand_ext(string cmd, bool calcLRC, ref byte[] response, int timeout, bool noResponse, string ident="")</code>	457
21.10.2.147	<code>device_sendPAE(string command, ref string response, int timeout, string ident="")</code>	457
21.10.2.148	<code>device_sendVivoCommandP2(byte command, byte subCommand, byte[] data, ref byte[] response, string ident="")</code>	457

21.10.2.149	<code>device_sendVivoCommandP2_ext(byte command, byte subCommand, byte[] data, ref byte[] response, int timeout, bool noResponse, string ident="")</code>	458
21.10.2.150	<code>device_sendVivoCommandP3(byte command, byte subCommand, byte[] data, ref byte[] response, string ident="")</code>	458
21.10.2.151	<code>device_sendVivoCommandP3_ext(byte command, byte subCommand, byte[] data, ref byte[] response, int timeout, bool noResponse, string ident="")</code>	459
21.10.2.152	<code>device_sendVivoCommandP4(byte command, byte subCommand, byte[] data, ref byte[] response, string ident="")</code>	459
21.10.2.153	<code>device_sendVivoCommandP4_ext(byte command, byte subCommand, byte[] data, ref byte[] response, int timeout, bool noResponse, string ident="")</code>	459
21.10.2.154	<code>device_setAudioVolume(int volume, string ident="")</code>	460
21.10.2.155	<code>device_setBurstMode(byte mode, string ident="")</code>	460
21.10.2.156	<code>device_setLED(byte indexLED, byte control, string ident="")</code>	460
21.10.2.157	<code>device_setMerchantRecord(int index, bool enabled, string merchantID, string merchantURL, string ident="")</code>	461
21.10.2.158	<code>device_setPollMode(byte mode, string ident="")</code>	461
21.10.2.159	<code>device_setSelfCheckTime(byte hour, byte minutes, string ident="")</code>	462
21.10.2.160	<code>device_setTerminalData(byte[] tlv, string ident="")</code>	462
21.10.2.161	<code>device_setTransArmorEncryption(byte[] cert, string ident="")</code>	462
21.10.2.162	<code>device_setTransArmorID(string TID, string ident="")</code>	462
21.10.2.163	<code>device_StartRKI(int type, string ident="")</code>	463
21.10.2.164	<code>device_startRKI(bool isTest, string ident="")</code>	463
21.10.2.165	<code>device_startTransaction(double amount, double amtOther, int exponent, int type, int timeout, byte[] tags, bool isFastEMV=false, string ident="")</code>	463
21.10.2.166	<code>device_startTransactionCB(double amount, double amtOther, int exponent, int type, int timeout, byte[] tags, CallbackIP callback, string ident="", bool isFastEMV=false)</code>	465
21.10.2.167	<code>device_stopAudio(string ident="")</code>	466
21.10.2.168	<code>device_terminalInfo(ref byte[] tlv, string ident="")</code>	466
21.10.2.169	<code>device_transferFile(string fileName, byte[] file, bool isSD=false, string ident="")</code>	467
21.10.2.170	<code>device_updateDeviceFirmware(byte[] firmwareData, string ident="")</code>	468
21.10.2.171	<code>device_updateDeviceFromManifest(string filepath, string ident="", bool isForeground=false)</code>	469
21.10.2.172	<code>device_updateFirmwareFromZip(byte[] zipfile, string ident="", bool isForeground=false)</code>	469
21.10.2.173	<code>device_updateFirmwareIP(string path, int type, FirmwareUpdateCallback callback, string ip, string ident="", bool performOnForeground=false)</code>	470
21.10.2.174	<code>device_updateFirmwareKernels(string path, FirmwareUpdateCallback callback, string ip="", UInt32 address=0, string ident="")</code>	471
21.10.2.175	<code>device_updateFirmwareType(FIRMWARE_TYPE type, byte[] firmwareData, string ident="", UInt32 address=0, bool performOnForeground=false, int manifestScriptsCount=0, int processedScriptCount=0, string DT_PRJ_filename="")</code>	472
21.10.2.176	<code>device_updateFirmwareType(string path, FIRMWARE_TYPE type, FirmwareUpdateCallback callback, string ident="", UInt32 address=0)</code>	474

21.10.2.177	<code>Device_wakeDevice(string macAddress="","", string ipAddress="")</code>	475
21.10.2.178	<code>mv_activateTransaction(int timeout, byte[] tags, bool forceOnline, bool isFastE← MV=false, string ident="")</code>	476
21.10.2.179	<code>mv_addTerminalData(byte[] tlv, string ident="")</code>	476
21.10.2.180	<code>mv_allowFallback(bool allow, string ident="")</code>	477
21.10.2.181	<code>mv_authenticateTransaction(byte[] updatedTLV, string ident="")</code>	477
21.10.2.182	<code>mv_authenticateTransactionCB(byte[] updatedTLV, CallBackIP callback, string ident="")</code>	477
21.10.2.183	<code>mv_autoAuthenticate(bool authenticate, string ident="")</code>	477
21.10.2.184	<code>mv_autoAuthenticateTags(bool authenticate, byte[] tags, string ident="")</code>	478
21.10.2.185	<code>mv_callbackResponseKSN(byte[] KSN, string ident="")</code>	478
21.10.2.186	<code>mv_callbackResponseLCD(EMV_LCD_DISPLAY_MODE type, byte selection, string ident="")</code>	478
21.10.2.187	<code>mv_callbackResponseMSR(byte[] MSR, string ident="")</code>	479
21.10.2.188	<code>mv_callbackResponsePIN(EMV_PIN_MODE type, byte[] KSN, byte[] PIN, string ident="")</code>	479
21.10.2.189	<code>mv_callbackResponsePIN_ETC(EMV_PIN_MODE type, byte[] KSN, byte[] PIN, string ident="")</code>	479
21.10.2.190	<code>mv_cancelTransaction(string ident="")</code>	480
21.10.2.191	<code>mv_completeTransaction(bool commError, byte[] authCode, byte[] iad, byte[] tlvScripts, byte[] tlv, string ident="")</code>	480
21.10.2.192	<code>mv_completeTransactionCB(bool commError, byte[] authCode, byte[] iad, byte[] tlvScripts, byte[] tlv, CallBackIP callback, string ident="")</code>	481
21.10.2.193	<code>mv_exchangeCerts(ref byte[] cert, ref byte[] nonce, ref byte[] signature, string ident="")</code>	481
21.10.2.194	<code>mv_generateDUKPT(byte[] cert, byte[] signature, ref byte[] key, string ident="")</code>	482
21.10.2.195	<code>mv_getEMVConfigurationCheckValue(ref string response, string ident="")</code>	482
21.10.2.196	<code>mv_getEMVKernelCheckValue(ref string response, string ident="")</code>	482
21.10.2.197	<code>mv_getEMVKernelVersion(ref string response, string ident="")</code>	483
21.10.2.198	<code>mv_getEMVKernelVersionExt(ref string response, string ident="")</code>	483
21.10.2.199	<code>mv_getTerminalMajorConfiguration(ref int configuration, string ident="")</code>	483
21.10.2.200	<code>mv_removeAllApplicationData(string ident="")</code>	483
21.10.2.201	<code>mv_removeAllCAPK(string ident="")</code>	484
21.10.2.202	<code>mv_removeAllCRL(string ident="")</code>	484
21.10.2.203	<code>mv_removeApplicationData(byte[] AID, string ident="")</code>	484
21.10.2.204	<code>mv_removeCAPK(byte[] capk, string ident="")</code>	484
21.10.2.205	<code>mv_removeCRL(byte[] crlList, string ident="")</code>	485
21.10.2.206	<code>mv_removeTerminalData(string ident="")</code>	485
21.10.2.207	<code>mv_resetConfigurationGroup(int group, string ident="")</code>	485
21.10.2.208	<code>mv_retrieveAIDList(ref byte[][] response, string ident="")</code>	486
21.10.2.209	<code>mv_retrieveApplicationData(byte[] AID, ref byte[] tlv, string ident="")</code>	486
21.10.2.210	<code>mv_retrieveCAPK(byte[] capk, ref byte[] key, string ident="")</code>	486

21.10.2.214	<code>mv_retrieveCAPKList(ref byte[] keys, string ident="")</code>	487
21.10.2.215	<code>mv_retrieveCRLList(ref byte[] list, string ident="")</code>	487
21.10.2.216	<code>mv_retrieveTerminalData(ref byte[] tlv, string ident="")</code>	488
21.10.2.217	<code>mv_retrieveTransactionResult(byte[] tags, ref IDTTransactionData tlv, string ident="")</code>	488
21.10.2.218	<code>mv_setApplicationData(byte[] name, byte[] tlv, string ident="")</code>	488
21.10.2.219	<code>mv_setApplicationData(byte[] tlv, string ident="")</code>	489
21.10.2.220	<code>mv_setCAPK(byte[] key, string ident="")</code>	490
21.10.2.221	<code>mv_setCRL(byte[] list, string ident="")</code>	490
21.10.2.222	<code>mv_setTerminalData(byte[] tlv, string ident="")</code>	490
21.10.2.223	<code>mv_setTerminalMajorConfiguration(int configuration, string ident="")</code>	491
21.10.2.224	<code>mv_startTransaction(double amount, double amtOther, int exponent, int type, int timeout, byte[] tags, bool forceOnline, bool isFastEMV=false, string ident="")</code>	491
21.10.2.225	<code>mv_startTransactionCB(double amount, double amtOther, int exponent, int type, int timeout, byte[] tags, bool forceOnline, CallBackIP callback, bool isFastEMV=false, string ident="")</code>	492
21.10.2.226	<code>mv_trySetTerminalData(byte[] tlv, ref byte[] rejectedTLV, ref byte[] convertedTLV, bool overwrite=false, string ident="")</code>	492
21.10.2.227	<code>mv_verifyDUKPTLoaded(byte[] KCV, string ident="")</code>	493
21.10.2.228	<code>olica_authentication(byte[] key, string ident="")</code>	493
21.10.2.229	<code>olica_read(byte[] serviceCode, int numBlocks, byte[] blockList, ref byte[] blocks, string ident="")</code>	493
21.10.2.230	<code>olica_readWithMac(int numBlocks, byte[] blockList, ref byte[] blocks, string ident="")</code>	494
21.10.2.231	<code>olica_requestService(byte[] nodeCode, ref byte[] response, string ident="")</code>	494
21.10.2.232	<code>olica_SendCommand(byte[] command, ref byte[] response, string ident="")</code>	494
21.10.2.233	<code>olica_write(byte[] serviceCode, int blockCount, byte[] blockList, byte[] data, ref byte[] statusFlag, string ident="")</code>	495
21.10.2.234	<code>olica_writeWithMac(int blockNumber, byte[] data, string ident="")</code>	495
21.10.2.235	<code>getCommandTimeout(string ident="")</code>	495
21.10.2.236	<code>getLastErrorString(string ident="")</code>	496
21.10.2.237	<code>c_exchangeAPDU(string c_APDU, ref byte[] response, string ident="")</code>	496
21.10.2.238	<code>c_getICCRReaderStatus(ref byte status, string ident="")</code>	496
21.10.2.239	<code>c_powerOffICC(string ident="")</code>	496
21.10.2.240	<code>c_powerOnICC(ref byte[] ATR, byte interfaces, string ident="")</code>	497
21.10.2.241	<code>p_autoConnectToSocket()</code>	497
21.10.2.242	<code>p_connectToSocket(string IP, bool isSecure=false)</code>	498
21.10.2.243	<code>p_firstConnectToSocket(string IP)</code>	498
21.10.2.244	<code>p_getSocketList()</code>	499
21.10.2.245	<code>p_isConnected(string ip, int attempts=1, bool isSecure=false)</code>	499
21.10.2.246	<code>p_monitorSocketConnectionStatus(bool enable, bool monitorConnect, int interval, int retryCount)</code>	499
21.10.2.247	<code>p_switchToSocket(string IP)</code>	500

21.10.2.245d_addButton(string screenName, string buttonName, byte type, byte alignment, UInt16 xCord, UInt16 yCord, string label, ref lcdItem returnItem, buttonCallback callback, string ident="")	500
21.10.2.246d_addEthernet(string screenName, string objectName, byte alignment, UInt16 xCord, UInt16 yCord, ref lcdItem returnItem, string ip, string ident="")	501
21.10.2.247d_addImage(string screenName, string objectName, byte alignment, UInt16 xCord, UInt16 yCord, string filename, ref lcdItem returnItem, string ident="")	502
21.10.2.248d_addLED(string screenName, string objectName, byte alignment, UInt16 xCord, UInt16 yCord, ref lcdItem returnItem, byte[] LED, string ident="")	502
21.10.2.249d_addText(string screenName, string objectName, byte alignment, UInt16 xCord, UInt16 yCord, UInt16 width, UInt16 height, byte fontID, byte[] color, string label, ref lcdItem returnItem, string ident="")	503
21.10.2.250d_clearAllLines(string ident="")	505
21.10.2.251d_clearDisplay(string ident="")	506
21.10.2.252d_clearScreenInfo(string ident="")	506
21.10.2.253d_cloneScreen(string screenName, string cloneName, ref UInt16 cloneID, string ident="")	506
21.10.2.254d_createScreen(string screenName, ref UInt16 screenID, string ident="")	507
21.10.2.255d_destroyScreen(string screenName, string ident="")	507
21.10.2.256d_displayMessage(int lineNumber, string message, string ident="")	507
21.10.2.257d_getActiveScreen(ref string screenName, string ident="")	508
21.10.2.258d_getAllObjects(string screenName, ref byte objectNumbers, ref Dictionary< UInt16, string > returnObjects, string ident="")	508
21.10.2.259d_getAllScreens(ref byte screenNumbers, ref Dictionary< UInt16, string > returnScreens, string ident="")	508
21.10.2.260d_getAudioVolume(ref int volume, string ident="")	509
21.10.2.261d_getButtonEvent(ref UInt16 screenID, ref UInt16 objectID, ref string screenName, ref string objectName, ref bool isLongPress, string ident="")	509
21.10.2.262d_linkUIWithTransactionMessageId(byte messageID, string screenName, string ident="")	509
21.10.2.263d_loadScreenInfo(string ident="")	510
21.10.2.264d_playAudio(string name, int type, string ident="")	510
21.10.2.265d_queryObjectbyID(UInt16 objectID, ref byte objectNumbers, ref List< string > returnItems, string ident="")	510
21.10.2.266d_queryObjectbyName(string objectName, ref byte objectNumbers, ref List< string > returnItems, string ident="")	511
21.10.2.267d_queryScreenbyID(UInt16 screenID, ref byte result, ref string screenName, string ident="")	511
21.10.2.268d_queryScreenbyName(string screenName, ref byte result, string ident="")	511
21.10.2.269d_removeItem(string screenName, string objectName, string ident="")	512
21.10.2.270d_retrieveMessage(DisplayMessages.DISPLAY_MESSAGE_LANGUAGE lang, DisplayMessages.DISPLAY_MESSAGE_IDENTIFIER id, ref string line1, ref string line2)	512
21.10.2.271d_setAudioVolume(int volume, string ident="")	512
21.10.2.272d_setBacklight(bool isBacklightOn, byte backlightVal, string ident="")	514

21.10.2.273d_setButtonCallback(string screenName, string buttonName, buttonCallback callback, string ip, string ident="")	514
21.10.2.274d_setPinCancelPromptCallback(CancelPromptCallback callback, string ident="")	514
21.10.2.275d_setPinFailureCallback(FailureCallback callback, string ident="")	515
21.10.2.276d_setPinInputCallback(SwipeCallback callback, string ident="")	515
21.10.2.277d_setPinSwipeCallback(SwipeCallback callback, string ident="")	515
21.10.2.278d_setPinTimeoutCallback(TimeoutCallback callback, string ident="")	516
21.10.2.279d_showScreen(string screenName, string ident="")	516
21.10.2.280d_startCameraCapture(ushort timeout, string ident="")	516
21.10.2.281d_startScanQR(ushort timeout, string ident="")	516
21.10.2.282d_startScanQR_ext(ushort timeout, ushort xcord, ushort ycord, ushort width, ushort height, string ident="")	517
21.10.2.283d_startScreenSaver(string name, string ident="")	517
21.10.2.284d_stopAudio(string ident="")	518
21.10.2.285d_stopCameraCapture(string ident="")	518
21.10.2.286d_stopScanQR(string ident="")	518
21.10.2.287d_storeScreenInfo(string ident="")	518
21.10.2.288d_updateColor(string screenName, string objectName, byte[] color, string ident="")	519
21.10.2.289d_updateLabel(string screenName, string objectName, string label, string ident="")	519
21.10.2.290d_updatePosition(string screenName, string objectName, byte alignment, UInt16 new_xCord, UInt16 new_yCord, string ident="")	520
21.10.2.291monitorNetworkForDevices(bool monitorON, int port=0, bool authClient=false, int sendTimeout=6000, int receiveTimeout=6000)	520
21.10.2.292sr_cancelMSRSwipe(string ident="")	521
21.10.2.293sr_getConfiguration(ref byte[] config, string ident="")	521
21.10.2.294sr_getMSRTrack(ref int val, string ident="")	521
21.10.2.295sr_setConfiguration(byte[] config, string ident="")	522
21.10.2.296sr_setMSRTrack(int val, string ip, string ident="")	522
21.10.2.297sr_startMSRSwipe(int timeout, string ident="")	523
21.10.2.298sr_startMSRSwipe_ext(int timeout, string ident, SwipeCallback swipeCallback, TimeoutCallback timeoutCallback, FailureCallback failureCallback)	523
21.10.2.299in_cancelPINEntry(string ident="")	524
21.10.2.300in_capturePin(int timeout, int type, string PAN, int minPIN, int maxPIN, string message, string ident="")	524
21.10.2.301in_capturePin_ext(int timeout, int type, string PAN, int minPIN, int maxPIN, string message, string ident, SwipeCallback inputCallback, FailureCallback failureCallback, TimeoutCallback timeoutCallback, CancelPromptCallback cancelPromptCallback)	525
21.10.2.302in_capturePinExt(int timeout, int type, string PAN, int minPIN, int maxPIN, string message1, string message2, string verify1, string verify2, string ident="")	526
21.10.2.303in_getFunctionKey(int timeout, string ident="")	528

21.10.2.304	<code>in_getFunctionKey_ext(int timeout, string ip, SwipeCallback inputCallback, FailureCallback failureCallback, TimeoutCallback timeoutCallback, CancelPromptCallback cancelPromptCallback)</code>	528
21.10.2.305	<code>in_getPanEntry(bool csc, bool expDate, bool ADR, bool ZIP, bool mod10, UInt16 timeout, bool encPANOnly=false, string ident="")</code>	528
21.10.2.306	<code>in_getPanEntry_ext(bool csc, bool expDate, bool ADR, bool ZIP, bool mod10, UInt16 timeout, bool encPANOnly, string ip, SwipeCallback swipeCallback, TimeoutCallback timeoutCallback, CancelPromptCallback cancelPromptCallback)</code>	529
21.10.2.307	<code>in_promptForAmount(int timeout, int minLen, int maxLen, string message, byte[] signature, string ident="")</code>	530
21.10.2.308	<code>in_promptForAmount_ext(int timeout, int minLen, int maxLen, string message, byte[] signature, string ident, SwipeCallback inputCallback, FailureCallback failureCallback, TimeoutCallback timeoutCallback, CancelPromptCallback cancelPromptCallback)</code>	530
21.10.2.309	<code>in_promptForInput(int messageID, short timeout, string ip=null, string ident="")</code>	531
21.10.2.310	<code>in_promptForInput_ext(int messageID, short timeout, string ident, SwipeCallback inputCallback, SwipeCallback swipeCallback, FailureCallback failureCallback, TimeoutCallback timeoutCallback, CancelPromptCallback cancelPromptCallback)</code>	531
21.10.2.311	<code>in_promptForNumericKeyWithSwipe(short timeout, byte function, int minLen, int maxLen, string line1, string line2, byte[] signature, string ident="")</code>	532
21.10.2.312	<code>in_promptForNumericKeyWithSwipe_ext(short timeout, byte function, int minLen, int maxLen, string line1, string line2, byte[] signature, string ident, SwipeCallback inputCallback, SwipeCallback swipeCallback, FailureCallback failureCallback, TimeoutCallback timeoutCallback, CancelPromptCallback cancelPromptCallback)</code>	533
21.10.2.313	<code>in_sendBeep(string ident="")</code>	534
21.10.2.314	<code>IDK_Version()</code>	534
21.10.2.315	<code>SetCallback(CallBack my_Callback)</code>	534
21.10.2.316	<code>SetCallback(IntPtr my_Callback, SynchronizationContext context)</code>	534
21.10.2.317	<code>SetCallbackIP(CallBackIP my_Callback, string ident="")</code>	535
21.10.2.318	<code>SetCommandTimeout(int milliseconds, string ident="")</code>	535
21.10.2.319	<code>SetLongPressCallback(longPressCallback callback, string ident="")</code>	535
21.10.2.320	<code>UseSerialPort(int port)</code>	535
21.10.2.321	<code>UseSerialPort(int port, int baud)</code>	536
21.10.2.322	<code>UseSerialPortLinux(string path)</code>	536
21.10.2.323	<code>UseSerialPortLinux(string path, int baud)</code>	536
21.10.3	Property Documentation	536
21.10.3.1	SharedController	537
21.11	IDTechSDK.IDT_PIP Class Reference	537
21.11.1	Detailed Description	538
21.11.2	Member Function Documentation	538
21.11.2.1	<code>config_getSerialNumber(ref string response, string ident="")</code>	538
21.11.2.2	<code>ctls_activateTransaction(int timeout, byte[] tags, bool forceOnline=false, bool isFastEMV=false, string ident="")</code>	539

21.11.2.3	<code>ctls_cancelTransaction(string ident="")</code>	540
21.11.2.4	<code>ctls_getAllConfigurationGroups(ref byte[] response, string ident="")</code>	540
21.11.2.5	<code>ctls_getConfigurationGroup(int group, ref byte[] tlv, string ident="")</code>	541
21.11.2.6	<code>ctls_nfcCommand(byte[] nfcCmdPkt, ref byte[] response, string ident="")</code>	541
21.11.2.7	<code>ctls_removeApplicationData(byte[] AID, string ident="")</code>	542
21.11.2.8	<code>ctls_removeConfigurationGroup(int group, string ident="")</code>	543
21.11.2.9	<code>ctls_retrieveAIDList(ref byte[] response, string ident="")</code>	543
21.11.2.10	<code>ctls_retrieveApplicationData(byte[] AID, ref byte[] tlv, string ident="")</code>	543
21.11.2.11	<code>ctls_retrieveTerminalData(ref byte[] tlv, string ident="")</code>	544
21.11.2.12	<code>ctls_setApplicationData(byte[] tlv, string ident="")</code>	544
21.11.2.13	<code>ctls_setConfigurationGroup(byte[] tlv, string ident="")</code>	544
21.11.2.14	<code>ctls_setDefaultConfiguration(string ident="")</code>	545
21.11.2.15	<code>ctls_setTerminalData(byte[] tlv, string ident="")</code>	545
21.11.2.16	<code>ctls_startTransaction(double amount, double amtOther, int exponent, int type, int timeout, byte[] tags, bool forceOnline=false, bool isFastEMV=false, string ident="")</code>	545
21.11.2.17	<code>ctls_trySetTerminalData(byte[] tlv, ref byte[] rejectedTLV, ref byte[] convertedTLV, string ident="")</code>	547
21.11.2.18	<code>ctls_updateBalance(byte statusCode, byte[] authCode, byte[] date, byte[] time, string ident="")</code>	547
21.11.2.19	<code>device_activateTransaction(int timeout, byte[] tags, byte[] options=null, bool isFastEMV=false, string ident="")</code>	547
21.11.2.20	<code>device_controlUserInterface(byte[] values, string ident="")</code>	549
21.11.2.21	<code>device_enablePassThrough(bool enablePassThrough, string ident="")</code>	549
21.11.2.22	<code>device_getAnyRKIStatus(bool isMultiKey, ref string status, ref Dictionary< string, RKI_KEY_TYPE > keys, string ident="")</code>	550
21.11.2.23	<code>device_getConfigurationFromMemory(ref string json, string ident="")</code>	550
21.11.2.24	<code>device_getFirmwareVersion(ref string response, string ident="")</code>	550
21.11.2.25	<code>device_getMerchantRecord(int index, ref byte[] record, string ident="")</code>	551
21.11.2.26	<code>device_getPIPMODE(ref int mode, string ident="")</code>	551
21.11.2.27	<code>device_getRKIStatus(bool isProd, bool isMultiKey, ref string status, ref Dictionary< string, RKI_KEY_TYPE > keys, string ident="")</code>	551
21.11.2.28	<code>device_getTransactionResults(ref IDTTTransactionData results, string ident="")</code>	552
21.11.2.29	<code>device_pingDevice(string ident="")</code>	552
21.11.2.30	<code>device_pollForToken(byte seconds, ref byte card, ref byte[] serialNumber, string ident="")</code>	552
21.11.2.31	<code>device_readConfigurationToMemory(string memo, Configs.InstallRules rules, List< Configs.DeviceCommand > cmds, string ident="", bool isForeground=false)</code>	553
21.11.2.32	<code>device_RemoteKeyInjection(RKI_KEY_TYPE type, string keyName, string ident="", bool performOnForeground=false)</code>	554
21.11.2.33	<code>device_retrieveAIDList(ref byte[] response, string ident="")</code>	554
21.11.2.34	<code>device_sendConfiguration(string filename, VC_OPERATION_TYPE type, bool matchFW, string ident="", bool isForeground=false)</code>	554

21.11.2.35	device_sendConfigurationFromZip(byte[] zip, string filename, VC_OPERATION← _TYPE type, bool matchFW, string ident="")	555
21.11.2.36	device_sendDataCommand(string cmd, bool calcLRC, ref byte[] response, string ident="")	556
21.11.2.37	device_sendDataCommand_ext(string cmd, bool calcLRC, ref byte[] response, int timeout, bool noResponse, string ident="")	556
21.11.2.38	device_sendVivoCommandP2(byte command, byte subCommand, byte[] data, ref byte[] response, string ident="")	557
21.11.2.39	device_sendVivoCommandP2_ext(byte command, byte subCommand, byte[] data, ref byte[] response, int timeout, bool noResponse, string ident="")	557
21.11.2.40	device_setBurstMode(byte mode, string ident="")	557
21.11.2.41	device_setMerchantRecord(int index, bool enabled, string merchantID, string merchantURL, string ident="")	558
21.11.2.42	device_setPIPMODE(int mode, string ident="")	558
21.11.2.43	device_setPollMode(byte mode, string ident="")	558
21.11.2.44	device_startTransaction(double amount, double amtOther, int exponent, int type, int timeout, byte[] tags, byte[] options=null, bool isFastEMV=false, int trans← Mode=0, string ident="")	559
21.11.2.45	device_updateDeviceFirmware(byte[] firmwareData, string ident="")	560
21.11.2.46	device_updateFirmwareFromZip(byte[] zipfile, string ident="", bool is← Foreground=false)	561
21.11.2.47	felica_requestService(byte[] nodeCode, ref byte[] response, string ident="")	562
21.11.2.48	getCommandTimeout(string ident="")	562
21.11.2.49	gcd_retrieveMessage(DisplayMessages.DISPLAY_MESSAGE_LANGUAG← E lang, DisplayMessages.DISPLAY_MESSAGE_IDENTIFIER id, ref string line1, ref string line2)	562
21.11.2.50	SDK_Version()	562
21.11.2.51	setCallback(CallBack my_Callback)	563
21.11.2.52	setCallback(IntPtr my_Callback, SynchronizationContext context)	563
21.11.2.53	setCommandTimeout(int milliseconds, string ident="")	563
21.11.3	Property Documentation	563
21.11.3.1	SharedController	563
21.12	IDTechSDK.IDT_SecureKey Class Reference	563
21.12.1	Detailed Description	565
21.12.2	Member Function Documentation	565
21.12.2.1	config_getModelNumber(ref string response, string ident="")	565
21.12.2.2	config_getSerialNumber(ref string response, string ident="")	565
21.12.2.3	device_capturePINFromLast12(int min, int max, int timeout, string last12, string ident="")	565
21.12.2.4	device_getAnyRKIStatus(bool isMultiKey, ref string status, ref Dictionary< string, RKI_KEY_TYPE > keys, string ident="")	566
21.12.2.5	device_getConfigurationFromMemory(ref string json, string ident="")	566
21.12.2.6	device_getFirmwareVersion(ref string response, string ident="")	566
21.12.2.7	device_getOutputType(bool isSwipe, ref int response, string ident="")	566

21.12.2.8	<code>device_getResponseCodeString(RETURN_CODE eCode)</code>	567
21.12.2.9	<code>device_getRKIStatus(bool isProd, bool isMultiKey, ref string status, ref Dictionary< string, RKL_KEY_TYPE > keys, string ident="")</code>	577
21.12.2.10	<code>device_readConfigurationToMemory(string memo, Configs.InstallRules rules, List< Configs.DeviceCommand > cmds, string ident="", bool isForeground=false)</code>	577
21.12.2.11	<code>device_RemoteKeyInjection(RKL_KEY_TYPE type, string keyName, string ident="", bool performOnForeground=false)</code>	578
21.12.2.12	<code>device_sendConfiguration(string filename, VC_OPERATION_TYPE type, bool matchFW, string ident="", bool isForeground=false)</code>	578
21.12.2.13	<code>device_sendConfigurationFromZip(byte[] zip, string filename, VC_OPERATION_TYPE type, bool matchFW, string ident="", bool isForeground=false)</code>	579
21.12.2.14	<code>device_sendDataCommand(string cmd, bool calcLRC, ref byte[] response, string ident="")</code>	580
21.12.2.15	<code>device_sendDataCommand_ext(string cmd, bool calcLRC, ref byte[] response, int timeout, bool noResponse, string ident="")</code>	580
21.12.2.16	<code>device_sendPAE(string command, ref string response, int timeout, string ident="")</code>	581
21.12.2.17	<code>device_sendVivoCommandP2(byte command, byte subCommand, byte[] data, ref byte[] response, string ident="")</code>	581
21.12.2.18	<code>device_sendVivoCommandP2_ext(byte command, byte subCommand, byte[] data, ref byte[] response, int timeout, bool noResponse, string ident="")</code>	581
21.12.2.19	<code>device_sendVivoCommandP3(byte command, byte subCommand, byte[] data, ref byte[] response, string ident="")</code>	582
21.12.2.20	<code>device_sendVivoCommandP3_ext(byte command, byte subCommand, byte[] data, ref byte[] response, int timeout, bool noResponse, string ident="")</code>	582
21.12.2.21	<code>device_setOutputType(bool isSwipe, int value, string ident="")</code>	583
21.12.2.22	<code>device_StartRKI(int type, string ident="")</code>	583
21.12.2.23	<code>device_updateFirmwareFromZip(byte[] zipfile, string ident="", bool isForeground=false)</code>	583
21.12.2.24	<code>getCommandTimeout(string ident="")</code>	584
21.12.2.25	<code>cd_retrieveMessage(DisplayMessages.DISPLAY_MESSAGE_LANGUAGE lang, DisplayMessages.DISPLAY_MESSAGE_IDENTIFIER id, ref string line1, ref string line2)</code>	584
21.12.2.26	<code>msr_switchUSBInterfaceMode(bool blsUSBKeyboardMode)</code>	584
21.12.2.27	<code>msr_switchUSBInterfaceMode(bool blsUSBKeyboardMode, ref string ident)</code>	585
21.12.2.28	<code>SDK_Version()</code>	585
21.12.2.29	<code>setCallback(CallBack my_Callback)</code>	585
21.12.2.30	<code>setCallback(IntPtr my_Callback, SynchronizationContext context)</code>	585
21.12.2.31	<code>setCommandTimeout(int milliseconds, string ident="")</code>	586
21.12.3	Property Documentation	586
21.12.3.1	SharedController	586
21.13	IDTechSDK.IDT_SecureMag Class Reference	586
21.13.1	Detailed Description	587
21.13.2	Member Function Documentation	588
21.13.2.1	<code>config_getModelNumber(ref string response, string ident="")</code>	588

21.13.2.2 config_getSerialNumber(ref string response, string ident="")	588
21.13.2.3 device_getAnyRKIStatus(bool isMultiKey, ref string status, ref Dictionary< string, RKI_KEY_TYPE > keys, string ident="")	588
21.13.2.4 device_getConfigurationFromMemory(ref string json, string ident="")	588
21.13.2.5 device_getFirmwareVersion(ref string response, string ident="")	589
21.13.2.6 device_getResponseCodeString(RETURN_CODE eCode)	589
21.13.2.7 device_getRKIStatus(bool isProd, bool isMultiKey, ref string status, ref Dictionary< string, RKI_KEY_TYPE > keys, string ident="")	599
21.13.2.8 device_readConfigurationToMemory(string memo, Configs.InstallRules rules, List< Configs.DeviceCommand > cmds, string ident="", bool isForeground=false)	599
21.13.2.9 device_RemoteKeyInjection(RKI_KEY_TYPE type, string keyName, string ident="", bool performOnForeground=false)	600
21.13.2.10 device_sendConfiguration(string filename, VC_OPERATION_TYPE type, bool matchFW, string ident="", bool isForeground=false)	600
21.13.2.11 device_sendConfigurationFromZip(byte[] zip, string filename, VC_OPERATION_TYPE type, bool matchFW, string ident="", bool isForeground=false)	601
21.13.2.12 device_sendDataCommand(string cmd, bool calcLRC, ref byte[] response, string ident="")	602
21.13.2.13 device_sendDataCommand_ext(string cmd, bool calcLRC, ref byte[] response, int timeout, bool noResponse, string ident="")	602
21.13.2.14 device_sendPAE(string command, ref string response, int timeout, string ident="")	603
21.13.2.15 device_StartRKI(string ident="")	603
21.13.2.16 device_StartRKI(int type, string ident="")	604
21.13.2.17 device_updateFirmwareFromZip(byte[] zipfile, string ident="", bool isForeground=false)	604
21.13.2.18 getCommandTimeout(string ident="")	604
21.13.2.19 cd_retrieveMessage(DisplayMessages.DISPLAY_MESSAGE_LANGUAGE lang, DisplayMessages.DISPLAY_MESSAGE_IDENTIFIER id, ref string line1, ref string line2)	605
21.13.2.20 msr_cancelMSRSwipe(string ident="")	605
21.13.2.21 msr_captureMode(bool isBufferMode, bool withNotification=false, string ident="")	605
21.13.2.22 msr_disable(string ident="")	606
21.13.2.23 msr_getClearPANID(ref byte value, string ident="")	606
21.13.2.24 msr_getExpirationMask(ref byte value, string ident="")	606
21.13.2.25 msr_getFunctionStatus(ref bool enabled, ref bool isBufferMode, ref bool withNotification, string ident="")	606
21.13.2.26 msr_getSetting(byte setting, ref byte value, string ident="")	607
21.13.2.27 msr_getSettings(byte setting, ref byte[] value, string ident="")	607
21.13.2.28 msr_getSwipeForcedEncryptionOption(ref byte option, string ident="")	608
21.13.2.29 msr_getSwipeMaskOption(ref byte option, string ident="")	608
21.13.2.30 msr_RetrieveWhiteList(ref byte[] value, string ident="")	608
21.13.2.31 msr_setClearPANID(byte val, string ident="")	609
21.13.2.32 msr_setExpirationMask(bool mask, string ident="")	609
21.13.2.33 msr_setSettings(byte setting, byte[] value, string ident="")	609

21.13.2.34	<code>msr_setSwipeEncryption(byte encryption, string ident="")</code>	610
21.13.2.35	<code>msr_setSwipeForcedEncryptionOption(bool track1, bool track2, bool track3, bool track3card0, string ident="")</code>	610
21.13.2.36	<code>msr_setSwipeMaskOption(bool track1, bool track2, bool track3, string ident="")</code>	610
21.13.2.37	<code>msr_setWhiteList(byte[] value, byte[] MAC, string ident="")</code>	611
21.13.2.38	<code>msr_startMSRSwipe(int timeout, string ident="")</code>	611
21.13.2.39	<code>SDK_Version()</code>	611
21.13.2.40	<code>setCallback(CallBack my_Callback)</code>	611
21.13.2.41	<code>setCallback(IntPtr my_Callback, SynchronizationContext context)</code>	612
21.13.2.42	<code>setCommandTimeout(int milliseconds, string ident="")</code>	612
21.13.3	Property Documentation	612
21.13.3.1	SharedController	612
21.14	IDTechSDK.IDT_SpectrumPro Class Reference	612
21.14.1	Detailed Description	615
21.14.2	Member Function Documentation	615
21.14.2.1	<code>config_getModelNumber(ref string response, string ident="")</code>	615
21.14.2.2	<code>config_getSerialNumber(ref string response, string ident="")</code>	615
21.14.2.3	<code>createFastEMVData(ref IDTTransactionData cData, string ident="")</code>	615
21.14.2.4	<code>device_activateTransaction(int timeout, byte[] tags, bool forceOnline, bool isFast← EMV=false, string ident="")</code>	616
21.14.2.5	<code>device_cardNotification(bool enable, byte option, bool captureMSR, string ident="")</code>	616
21.14.2.6	<code>device_controlUserInterface(byte[] values, string ident="")</code>	617
21.14.2.7	<code>device_getAnyRKIStatus(bool isMultiKey, ref string status, ref Dictionary< string, RKI_KEY_TYPE > keys, string ident="")</code>	617
21.14.2.8	<code>device_getConfigurationFromMemory(ref string json, string ident="")</code>	618
21.14.2.9	<code>device_getFirmwareVersion(ref string response, string ident="")</code>	618
21.14.2.10	<code>device_getNonce(byte[] hostnonce, ref byte[] devicenonce, string ident="")</code>	618
21.14.2.11	<code>device_getResponseCodeString(RETURN_CODE eCode)</code>	619
21.14.2.12	<code>device_getRKIStatus(bool isProd, bool isMultiKey, ref string status, ref Dictionary< string, RKI_KEY_TYPE > keys, string ident="")</code>	628
21.14.2.13	<code>device_getSpectrumProKSN(int type, ref byte[] KSN, string ident="")</code>	629
21.14.2.14	<code>device_getVersions(ref SpectrumInfoExt info, string ident="")</code>	629
21.14.2.15	<code>device_pollCardReader(ref byte[] status, string ident="")</code>	630
21.14.2.16	<code>device_readConfigurationToMemory(string memo, Configs.InstallRules rules, List< Configs.DeviceCommand > cmds, string ident="", bool isForeground=false)</code>	631
21.14.2.17	<code>device_rebootDevice(string ident="")</code>	631
21.14.2.18	<code>device_RemoteKeyInjection(RKI_KEY_TYPE type, string keyName, string ident="", bool performOnForeground=false)</code>	631
21.14.2.19	<code>device_sendConfiguration(string filename, VC_OPERATION_TYPE type, bool matchFW, string ident="", bool isForeground=false)</code>	632
21.14.2.20	<code>device_sendConfigurationFromZip(byte[] zip, string filename, VC_OPERATION← _TYPE type, bool matchFW, string ident="", bool isForeground=false)</code>	633

21.14.2.21	device_sendDataCommand(string cmd, bool calcLRC, ref byte[] response, string ident="")	634
21.14.2.22	device_sendDataCommand_ext(string cmd, bool calcLRC, ref byte[] response, int timeout, bool noResponse, string ident="")	634
21.14.2.23	device_sendMacDataCommand(byte taskID, byte[] functionID, byte[] data, bool macData, ref byte[] response, string ident="")	634
21.14.2.24	device_sendPAE(string command, ref string response, int timeout, string ident="")	635
21.14.2.25	device_setSpectrumProBDK(string BDK, string ident="")	635
21.14.2.26	device_setUID(string UID, int keyType, ref SpectrumInfo info, int session← Timeout=0x0064, string ident="")	635
21.14.2.27	device_startRKI(bool isTest, string ident="")	636
21.14.2.28	device_StartRKI(int type, string ident="")	636
21.14.2.29	device_startTransaction(double amount, double amtOther, int exponent, int type, int timeout, byte[] tags, bool forceOnline, bool isFastEMV=false, string ident="")	637
21.14.2.30	device_updateFirmware(byte[] firmwareData, string firmwareName, int encryptionType, byte[] keyBlob, string ident="")	637
21.14.2.31	emv_activateTransaction(int timeout, byte[] tags, bool forceOnline, bool isFastE← MV=false, string ident="")	638
21.14.2.32	emv_addTerminalData(byte[] tlv, string ident="")	639
21.14.2.33	emv_allowFallback(bool allow, string ident="")	639
21.14.2.34	emv_authenticateTransaction(byte[] updatedTLV, string ident="")	639
21.14.2.35	emv_autoAuthenticate(bool authenticate, string ident="")	640
21.14.2.36	emv_autoAuthenticateTags(bool authenticate, byte[] tags, string ident="")	640
21.14.2.37	emv_callbackResponseLCD(EMV_LCD_DISPLAY_MODE type, byte selection, string ident="")	640
21.14.2.38	emv_callbackResponseMSR(byte[] MSR, string ident="")	640
21.14.2.39	emv_callbackResponsePIN(EMV_PIN_MODE type, byte[] KSN, byte[] PIN, string ident="")	641
21.14.2.40	emv_cancelTransaction(string ident="")	641
21.14.2.41	emv_clearTransactionLog(string ident="")	641
21.14.2.42	emv_completeTransaction(bool commError, byte[] authCode, byte[] iad, byte[] tlvScripts, byte[] tlv, string ident="")	642
21.14.2.43	emv_getEMVConfigurationCheckValue(ref string response, string ident="")	642
21.14.2.44	emv_getEMVKernelCheckValue(ref string response, string ident="")	643
21.14.2.45	emv_getEMVKernelVersion(ref string response, string ident="")	643
21.14.2.46	emv_getTerminalID(ref string response, string ident="")	643
21.14.2.47	emv_getTerminalMajorConfiguration(ref int configuration, string ident="")	643
21.14.2.48	emv_removeAllApplicationData(string ident="")	644
21.14.2.49	emv_removeAllCAPK(string ident="")	644
21.14.2.50	emv_removeAllCRL(string ident="")	644
21.14.2.51	emv_removeApplicationData(byte[] AID, string ident="")	645
21.14.2.52	emv_removeCAPK(byte[] capk, string ident="")	645
21.14.2.53	emv_removeCRL(byte[] crlList, string ident="")	645

21.14.2.54	<code>emv_removeTerminalData(string ident="")</code>	646
21.14.2.55	<code>emv_retrieveAIDList(ref byte[] response, string ident="")</code>	646
21.14.2.56	<code>emv_retrieveApplicationData(byte[] AID, ref byte[] tlv, string ident="")</code>	646
21.14.2.57	<code>emv_retrieveCAPK(byte[] capk, ref byte[] key, string ident="")</code>	646
21.14.2.58	<code>emv_retrieveCAPKList(ref byte[] keys, string ident="")</code>	647
21.14.2.59	<code>emv_retrieveCRLList(ref byte[] list, string ident="")</code>	647
21.14.2.60	<code>emv_retrieveTerminalData(ref byte[] tlv, string ident="")</code>	648
21.14.2.61	<code>emv_retrieveTransactionResult(byte[] tags, ref IDTTransactionData tlv, string ident="")</code>	648
21.14.2.62	<code>emv_setApplicationData(byte[] name, byte[] tlv, string ident="")</code>	648
21.14.2.63	<code>emv_setCAPK(byte[] key, string ident="")</code>	649
21.14.2.64	<code>emv_setCRL(byte[] list, string ident="")</code>	649
21.14.2.65	<code>emv_setTerminalData(byte[] tlv, string ident="")</code>	649
21.14.2.66	<code>emv_setTerminalID(string terminalID, string ident="")</code>	650
21.14.2.67	<code>emv_setTerminalMajorConfiguration(int configuration, string ident="")</code>	650
21.14.2.68	<code>emv_startTransaction(double amount, double amtOther, int exponent, int type, int timeout, byte[] tags, bool forceOnline, bool isFastEMV=false, string ident="")</code>	650
21.14.2.69	<code>emv_trySetTerminalData(byte[] tlv, ref byte[] rejectedTLV, ref byte[] convertedTLV, bool overwrite=false, string ident="")</code>	651
21.14.2.70	<code>getCommandTimeout(string ident="")</code>	651
21.14.2.71	<code>icc_getICCRReaderStatus(ref byte status, string ident="")</code>	652
21.14.2.72	<code>icc_getICCStatus(ref byte[] status, ref byte[] atr, string ident="")</code>	652
21.14.2.73	<code>icc_powerOffICC(string ident="")</code>	653
21.14.2.74	<code>icc_powerOnICC(ref byte[] ATR, string ident="")</code>	654
21.14.2.75	<code>cd_retrieveMessage(DisplayMessages.DISPLAY_MESSAGE_LANGUAGE ← E lang, DisplayMessages.DISPLAY_MESSAGE_IDENTIFIER id, ref string line1, ref string line2)</code>	654
21.14.2.76	<code>msr_cancelMSRSwipe(string ident="")</code>	654
21.14.2.77	<code>msr_clearMSRData(string ident="")</code>	654
21.14.2.78	<code>msr_getMSRData(ref IDTTransactionData card, string ident="")</code>	655
21.14.2.79	<code>msr_startMSRSwipe(int timeout, string ident="")</code>	655
21.14.2.80	<code>pin_cancelPINEntry(string ident="")</code>	655
21.14.2.81	<code>pin_getPIN(int mode, int PANSource, string iccPAN, int startTimeout, int entry ← Timeout, string language, string ident="")</code>	656
21.14.2.82	<code>pin_passThroughMode(bool enable, string ident="")</code>	656
21.14.2.83	<code>SDK_Version()</code>	657
21.14.2.84	<code>setCallback(CallBack my_Callback)</code>	657
21.14.2.85	<code>setCallback(IntPtr my_Callback, SynchronizationContext context)</code>	657
21.14.2.86	<code>setCardNotificationCallback(CardNotificationCallback my_Callback)</code>	657
21.14.2.87	<code>setCommandTimeout(int milliseconds, string ident="")</code>	657
21.14.2.88	<code>useSerialPort(int port)</code>	658
21.14.2.89	<code>useSerialPort(int port, int baud)</code>	658

21.14.2.90	useSerialPortLinux(string path)	658
21.14.2.91	useSerialPortLinux(string path, int baud)	658
21.14.3	Property Documentation	659
21.14.3.1	SharedController	659
21.15	IDTechSDK.IDT_SREDKey2 Class Reference	659
21.15.1	Detailed Description	661
21.15.2	Member Function Documentation	661
21.15.2.1	config_getModelNumber(ref string response, string ident="")	661
21.15.2.2	config_getSerialNumber(ref string response, string ident="")	661
21.15.2.3	config_getStatusKeySlots(ref byte[] keyslots, string ident="")	661
21.15.2.4	device_certificateType(ref int type, string ident="")	662
21.15.2.5	device_enableAdminKey(bool enable, string ident="")	662
21.15.2.6	device_getAnyRKIStatus(bool isMultiKey, ref string status, ref Dictionary< string, RKI_KEY_TYPE > keys, string ident="")	662
21.15.2.7	device_getConfigurationFromMemory(ref string json, string ident="")	663
21.15.2.8	device_getFirmwareVersion(ref string response, string ident="")	663
21.15.2.9	device_getOutputType(bool isSwipe, ref int response, string ident="")	663
21.15.2.10	device_getResponseCodeString(RETURN_CODE eCode)	664
21.15.2.11	device_getRKIStatus(bool isProd, bool isMultiKey, ref string status, ref Dictionary< string, RKI_KEY_TYPE > keys, string ident="")	673
21.15.2.12	device_getTransArmorID(ref string TID, string ident="")	674
21.15.2.13	device_readConfigurationToMemory(string memo, Configs.InstallRules rules, List< Configs.DeviceCommand > cmds, string ident="", bool isForeground=false)	674
21.15.2.14	device_rebootDevice(string ident="")	675
21.15.2.15	device_RemoteKeyInjection(RKI_KEY_TYPE type, string keyName, string ident="", bool performOnForeground=false)	675
21.15.2.16	device_sendConfiguration(string filename, VC_OPERATION_TYPE type, bool matchFW, string ident="", bool isForeground=false)	675
21.15.2.17	device_sendConfigurationFromZip(byte[] zip, string filename, VC_OPERATION_TYPE type, bool matchFW, string ident="", bool isForeground=false)	676
21.15.2.18	device_sendDataCommand(string cmd, bool calcLRC, ref byte[] response, string ident="")	677
21.15.2.19	device_sendDataCommand_ext(string cmd, bool calcLRC, ref byte[] response, int timeout, bool noResponse, string ident="")	677
21.15.2.20	device_sendVivoCommandP2(byte command, byte subCommand, byte[] data, ref byte[] response, string ident="")	678
21.15.2.21	device_sendVivoCommandP2_ext(byte command, byte subCommand, byte[] data, ref byte[] response, int timeout, bool noResponse, string ident="")	678
21.15.2.22	device_sendVivoCommandP3(byte command, byte subCommand, byte[] data, ref byte[] response, string ident="")	679
21.15.2.23	device_sendVivoCommandP3_ext(byte command, byte subCommand, byte[] data, ref byte[] response, int timeout, bool noResponse, string ident="")	679
21.15.2.24	device_sendVivoCommandP4(byte command, byte subCommand, byte[] data, ref byte[] response, string ident="")	679

21.15.2.25	<code>device_sendVivoCommandP4_ext(byte command, byte subCommand, byte[] data, ref byte[] response, int timeout, bool noResponse, string ident="")</code>	680
21.15.2.26	<code>device_setLanguage(int val, string ident="")</code>	680
21.15.2.27	<code>device_setOutputType(bool isSwipe, int value, string ident="")</code>	681
21.15.2.28	<code>device_setPollMode(byte mode, string ident="")</code>	681
21.15.2.29	<code>device_setTransArmorEncryption(byte[] cert, string ident="")</code>	681
21.15.2.30	<code>device_setTransArmorID(string TID, string ident="")</code>	682
21.15.2.31	<code>device_startRKI(string ident="")</code>	682
21.15.2.32	<code>device_updateDeviceFirmware(byte[] firmwareData, string ident="")</code>	682
21.15.2.33	<code>device_updateDeviceFromManifest(string filepath, string ident="", bool isForeground=false)</code>	683
21.15.2.34	<code>device_updateFirmwareFromZip(byte[] zipfile, string ident="", bool isForeground=false)</code>	684
21.15.2.35	<code>device_updateFirmwareType(FIRMWARE_TYPE type, byte[] firmwareData, string ident="", UInt32 address=0, bool performOnForeground=false, int manifestScriptsCount=0, int processedScriptCount=0, string DT_PRJ_filename="")</code>	684
21.15.2.36	<code>getCommandTimeout(string ident="")</code>	686
21.15.2.37	<code>cd_retrieveMessage(DisplayMessages.DISPLAY_MESSAGE_LANGUAGE lang, DisplayMessages.DISPLAY_MESSAGE_IDENTIFIER id, ref string line1, ref string line2)</code>	687
21.15.2.38	<code>msr_cancelTransaction(string ident="")</code>	687
21.15.2.39	<code>msr_disable(string ident="")</code>	687
21.15.2.40	<code>msr_enable(string ident="")</code>	687
21.15.2.41	<code>msr_getClearPANID(ref byte value, string ident="")</code>	688
21.15.2.42	<code>msr_getConfiguration(ref byte[] config, string ident="")</code>	688
21.15.2.43	<code>msr_getExpirationMask(ref byte value, string ident="")</code>	688
21.15.2.44	<code>msr_getFunctionStatus(ref bool enabled, ref bool isBufferMode, ref bool withNotification, string ident="")</code>	689
21.15.2.45	<code>msr_getSwipeEncryption(ref byte encryption, string ident="")</code>	689
21.15.2.46	<code>msr_getSwipeForcedEncryptionOption(ref byte option, string ident="")</code>	689
21.15.2.47	<code>msr_getSwipeMaskOption(ref byte option, string ident="")</code>	690
21.15.2.48	<code>msr_getTerminalData(byte[] tag, ref byte[] tlv, string ident="")</code>	690
21.15.2.49	<code>msr_resetTerminalData(string ident="")</code>	691
21.15.2.50	<code>msr_setClearPANID(byte val, string ident="")</code>	691
21.15.2.51	<code>msr_setConfiguration(byte[] config, string ident="")</code>	691
21.15.2.52	<code>msr_setExpirationMask(bool mask, string ident="")</code>	691
21.15.2.53	<code>msr_setSwipeForcedEncryptionOption(bool track1, bool track2, bool track3, bool track3card0, string ident="")</code>	692
21.15.2.54	<code>msr_setSwipeMaskOption(bool track1, bool track2, bool track3, string ident="")</code>	692
21.15.2.55	<code>msr_setTerminalData(byte[] tlv, string ident="")</code>	692
21.15.2.56	<code>msr_startKeyedTransaction(int timeout, bool ZIP, bool CSC, bool ADR, bool Mod10Check, string ident="")</code>	693

21.15.2.57	<code>msr_startSwipeTransaction(int timeout, byte[] tlv, string ident="")</code>	693
21.15.2.58	<code>msr_switchUSBInterfaceMode(bool blsUSBKeyboardMode)</code>	693
21.15.2.59	<code>msr_switchUSBInterfaceMode(bool blsUSBKeyboardMode, ref string ident)</code>	694
21.15.2.60	<code>SDK_Version()</code>	694
21.15.2.61	<code>setCallback(CallBack my_Callback)</code>	694
21.15.2.62	<code>setCallback(IntPtr my_Callback, SynchronizationContext context)</code>	695
21.15.2.63	<code>setCommandTimeout(int milliseconds, string ident="")</code>	695
21.15.3	Property Documentation	695
21.15.3.1	SharedController	695
21.16	IDTechSDK.IDT_UniPay Class Reference	695
21.16.1	Detailed Description	697
21.16.2	Member Function Documentation	697
21.16.2.1	<code>config_getModelNumber(ref string response, string ident="")</code>	697
21.16.2.2	<code>config_getSerialNumber(ref string response, string ident="")</code>	697
21.16.2.3	<code>config_setCmdTimeOutDuration(int newTimeOut, string ident="")</code>	697
21.16.2.4	<code>device_getAnyRKIStatus(bool isMultiKey, ref string status, ref Dictionary< string, RKI_KEY_TYPE > keys, string ident="")</code>	698
21.16.2.5	<code>device_getBatteryVoltage(ref string voltage, string ident="")</code>	698
21.16.2.6	<code>device_getBootloaderVersion(ref string response, string ident="")</code>	699
21.16.2.7	<code>device_getConfigurationFromMemory(ref string json, string ident="")</code>	699
21.16.2.8	<code>device_getFirmwareVersion(ref string response, string ident="")</code>	699
21.16.2.9	<code>device_getRKIStatus(bool isProd, bool isMultiKey, ref string status, ref Dictionary< string, RKI_KEY_TYPE > keys, string ident="")</code>	699
21.16.2.10	<code>device_readConfigurationToMemory(string memo, Configs.InstallRules rules, List< Configs.DeviceCommand > cmds, string ident="", bool isForeground=false)</code>	700
21.16.2.11	<code>device_rebootDevice(string ident="")</code>	700
21.16.2.12	<code>device_RemoteKeyInjection(RKI_KEY_TYPE type, string keyName, string ident="", bool performOnForeground=false)</code>	701
21.16.2.13	<code>device_sendConfiguration(string filename, VC_OPERATION_TYPE type, bool matchFW, string ident="", bool isForeground=false)</code>	701
21.16.2.14	<code>device_sendConfigurationFromZip(byte[] zip, string filename, VC_OPERATION_TYPE type, bool matchFW, string ident="", bool isForeground=false)</code>	702
21.16.2.15	<code>device_sendDataCommand(string cmd, bool calcLRC, ref byte[] response, string ident="")</code>	703
21.16.2.16	<code>device_sendPAE(string command, ref string response, int timeout, string ident="")</code>	703
21.16.2.17	<code>device_startRKI(string ident="")</code>	703
21.16.2.18	<code>device_updateFirmwareFromZip(byte[] zipfile, string ident="", bool isForeground=false)</code>	705
21.16.2.19	<code>getCommandTimeout(string ident="")</code>	705
21.16.2.20	<code>icc_exchangeAPDU(string c_APDU, ref byte[] response, string ident="")</code>	705
21.16.2.21	<code>icc_getAPDU_KSN(ref byte[] ksn, string ident="")</code>	706
21.16.2.22	<code>icc_getICCRReaderStatus(ref byte status, string ident="")</code>	706

21.16.2.23	<code>cc_getKeyFormatForICCDUKPT(ref byte format, string ident="")</code>	706
21.16.2.24	<code>cc_getKeyTypeForICCDUKPT(ref byte type, string ident="")</code>	707
21.16.2.25	<code>cc_powerOffICC(string ident="")</code>	707
21.16.2.26	<code>cc_powerOnICC(ref byte[] ATR, string ident="")</code>	707
21.16.2.27	<code>cc_setKeyFormatForICCDUKPT(byte encryption, string ident="")</code>	708
21.16.2.28	<code>cc_setKeyTypeForICCDUKPT(byte encryption, string ident="")</code>	708
21.16.2.29	<code>msr_cancelMSRSwipe(string ident="")</code>	708
21.16.2.30	<code>msr_getClearPANID(ref byte value, string ident="")</code>	709
21.16.2.31	<code>msr_getExpirationMask(ref byte value, string ident="")</code>	709
21.16.2.32	<code>msr_getSetting(byte setting, ref byte value, string ident="")</code>	709
21.16.2.33	<code>msr_getSwipeEncryption(ref byte encryption, string ident="")</code>	709
21.16.2.34	<code>msr_getSwipeForcedEncryptionOption(ref byte option, string ident="")</code>	710
21.16.2.35	<code>msr_getSwipeMaskOption(ref byte option, string ident="")</code>	710
21.16.2.36	<code>msr_setClearPANID(byte val, string ident="")</code>	711
21.16.2.37	<code>msr_setExpirationMask(bool mask, string ident="")</code>	711
21.16.2.38	<code>msr_setSetting(byte setting, byte value, string ident="")</code>	711
21.16.2.39	<code>msr_setSwipeEncryption(byte encryption, string ident="")</code>	711
21.16.2.40	<code>msr_setSwipeForcedEncryptionOption(bool track1, bool track2, bool track3, bool track3card0, string ident="")</code>	712
21.16.2.41	<code>msr_setSwipeMaskOption(bool track1, bool track2, bool track3, string ident="")</code>	712
21.16.2.42	<code>msr_startMSRSwipe(int timeout, string ident="")</code>	712
21.16.2.43	<code>SDK_Version()</code>	713
21.16.2.44	<code>setCallback(CallBack my_Callback)</code>	713
21.16.2.45	<code>setCallback(IntPtr my_Callback, SynchronizationContext context)</code>	713
21.16.2.46	<code>setCommandTimeout(int milliseconds, string ident="")</code>	713
21.16.3	Property Documentation	714
21.16.3.1	SharedController	714
21.17	IDTechSDK.IDT_UniPayl_V Class Reference	714
21.17.1	Detailed Description	716
21.17.2	Member Function Documentation	716
21.17.2.1	<code>config_getEncryptionControl(ref bool msr, ref bool icc, string ident="")</code>	716
21.17.2.2	<code>config_getSerialNumber(ref string response, string ident="")</code>	717
21.17.2.3	<code>config_setEncryptionControl(bool msr, bool icc, string ident="")</code>	717
21.17.2.4	<code>device_activateTransaction(int timeout, byte[] tags, bool forceOnline, bool isFast← EMV=false, string ident="")</code>	717
21.17.2.5	<code>device_enablePassThrough(bool enablePassThrough, string ident="")</code>	719
21.17.2.6	<code>device_getAnyRKIStatus(bool isMultiKey, ref string status, ref Dictionary< string, RKI_KEY_TYPE > keys, string ident="")</code>	719
21.17.2.7	<code>device_getConfigurationFromMemory(ref string json, string ident="")</code>	720
21.17.2.8	<code>device_getDRS(ref byte[] codeDRS, string ident="")</code>	720
21.17.2.9	<code>device_getFirmwareVersion(ref string response, string ident="")</code>	720

21.17.2.10	<code>device_getResponseCodeString(RETURN_CODE eCode)</code>	721
21.17.2.11	<code>device_getRKIStatus(bool isProd, bool isMultiKey, ref string status, ref Dictionary< string, RKI_KEY_TYPE > keys, string ident="")</code>	730
21.17.2.12	<code>device_pingDevice(string ident="")</code>	731
21.17.2.13	<code>device_readConfigurationToMemory(string memo, Configs.InstallRules rules, List< Configs.DeviceCommand > cmds, string ident="", bool isForeground=false)</code>	731
21.17.2.14	<code>device_RemoteKeyInjection(RKI_KEY_TYPE type, string keyName, string ident="", bool performOnForeground=false)</code>	732
21.17.2.15	<code>device_retrieveTerminalData(ref byte[] tlv, string ident="")</code>	732
21.17.2.16	<code>device_selfCheck(string ident="")</code>	732
21.17.2.17	<code>device_sendConfiguration(string filename, VC_OPERATION_TYPE type, bool matchFW, string ident="", bool isForeground=false)</code>	733
21.17.2.18	<code>device_sendConfigurationFromZip(byte[] zip, string filename, VC_OPERATION_TYPE type, bool matchFW, string ident="", bool isForeground=false)</code>	733
21.17.2.19	<code>device_sendDataCommand(string cmd, bool calcLRC, ref byte[] response, string ident="")</code>	734
21.17.2.20	<code>device_sendDataCommand_ext(string cmd, bool calcLRC, ref byte[] response, int timeout, bool noResponse, string ident="")</code>	735
21.17.2.21	<code>device_sendPAE(string command, ref string response, int timeout, string ident="")</code>	735
21.17.2.22	<code>device_sendVivoCommandP2(byte command, byte subCommand, byte[] data, ref byte[] response, string ident="")</code>	735
21.17.2.23	<code>device_sendVivoCommandP2_ext(byte command, byte subCommand, byte[] data, ref byte[] response, int timeout, bool noResponse, string ident="")</code>	736
21.17.2.24	<code>device_setBurstMode(byte mode, string ident="")</code>	736
21.17.2.25	<code>device_setPollMode(byte mode, string ident="")</code>	737
21.17.2.26	<code>device_setTerminalData(byte[] tlv, string ident="")</code>	737
21.17.2.27	<code>device_startRKI(string ident="")</code>	738
21.17.2.28	<code>device_StartRKI(int type, string ident="")</code>	738
21.17.2.29	<code>device_startTransaction(double amount, double amtOther, int exponent, int type, int timeout, byte[] tags, bool forceOnline, bool isFastEMV=false, string ident="")</code>	738
21.17.2.30	<code>device_updateDeviceFirmware(byte[] firmwareData, string ident="")</code>	739
21.17.2.31	<code>device_updateFirmwareFromZip(byte[] zipfile, string ident="", bool isForeground=false)</code>	740
21.17.2.32	<code>emv_activateTransaction(int timeout, byte[] tags, bool forceOnline, bool isFastEMV=false, string ident="")</code>	740
21.17.2.33	<code>emv_addTerminalData(byte[] tlv, string ident="")</code>	741
21.17.2.34	<code>emv_allowFallback(bool allow, string ident="")</code>	741
21.17.2.35	<code>emv_authenticateTransaction(byte[] updatedTLV, string ident="")</code>	741
21.17.2.36	<code>emv_autoAuthenticate(bool authenticate, string ident="")</code>	742
21.17.2.37	<code>emv_autoAuthenticateTags(bool authenticate, byte[] tags, string ident="")</code>	742
21.17.2.38	<code>emv_callbackResponseLCD(EMV_LCD_DISPLAY_MODE type, byte selection, string ident="")</code>	742
21.17.2.39	<code>emv_callbackResponseMSR(byte[] MSR, string ident="")</code>	743

21.17.2.40	<code>emv_callbackResponsePIN(EMV_PIN_MODE type, byte[] KSN, byte[] PIN, string ident="")</code>	743
21.17.2.41	<code>emv_cancelTransaction(string ident="")</code>	743
21.17.2.42	<code>emv_completeTransaction(bool commError, byte[] authCode, byte[] iad, byte[] tlvScripts, byte[] tlv, string ident="")</code>	744
21.17.2.43	<code>emv_getEMVConfigurationCheckValue(ref string response, string ident="")</code>	744
21.17.2.44	<code>emv_getEMVKernelCheckValue(ref string response, string ident="")</code>	744
21.17.2.45	<code>emv_getEMVKernelVersion(ref string response, string ident="")</code>	745
21.17.2.46	<code>emv_getEMVKernelVersionExt(ref string response, string ident="")</code>	745
21.17.2.47	<code>emv_removeAllApplicationData(string ident="")</code>	745
21.17.2.48	<code>emv_removeAllCAPK(string ident="")</code>	746
21.17.2.49	<code>emv_removeAllCRL(string ident="")</code>	746
21.17.2.50	<code>emv_removeApplicationData(byte[] AID, string ident="")</code>	746
21.17.2.51	<code>emv_removeCAPK(byte[] capk, string ident="")</code>	746
21.17.2.52	<code>emv_removeCRL(byte[] crlList, string ident="")</code>	747
21.17.2.53	<code>emv_removeTerminalData(string ident="")</code>	747
21.17.2.54	<code>emv_retrieveAIDList(ref byte[][] response, string ident="")</code>	747
21.17.2.55	<code>emv_retrieveApplicationData(byte[] AID, ref byte[] tlv, string ident="")</code>	747
21.17.2.56	<code>emv_retrieveCAPK(byte[] capk, ref byte[] key, string ident="")</code>	748
21.17.2.57	<code>emv_retrieveCAPKList(ref byte[] keys, string ident="")</code>	748
21.17.2.58	<code>emv_retrieveCRLList(ref byte[] list, string ident="")</code>	749
21.17.2.59	<code>emv_retrieveTerminalData(ref byte[] tlv, string ident="")</code>	749
21.17.2.60	<code>emv_retrieveTransactionResult(byte[] tags, ref IDTTransactionData tlv, string ident="")</code>	749
21.17.2.61	<code>emv_setApplicationData(byte[] name, byte[] tlv, string ident="")</code>	750
21.17.2.62	<code>emv_setCAPK(byte[] key, string ident="")</code>	750
21.17.2.63	<code>emv_setCRL(byte[] list, string ident="")</code>	750
21.17.2.64	<code>emv_setTerminalData(byte[] tlv, string ident="")</code>	751
21.17.2.65	<code>emv_setTerminalMajorConfiguration(int configuration, string ident="")</code>	751
21.17.2.66	<code>emv_startTransaction(double amount, double amtOther, int exponent, int type, int timeout, byte[] tags, bool forceOnline, bool isFastEMV=false, string ident="")</code>	751
21.17.2.67	<code>emv_trySetTerminalData(byte[] tlv, ref byte[] rejectedTLV, ref byte[] convertedTLV, bool overwrite=false, string ident="")</code>	752
21.17.2.68	<code>getCommandTimeout(string ident="")</code>	752
21.17.2.69	<code>icc_exchangeAPDU(string c_APDU, ref byte[] response, string ident="")</code>	753
21.17.2.70	<code>icc_getICCRReaderStatus(ref byte status, string ident="")</code>	753
21.17.2.71	<code>icc_powerOffICC(string ident="")</code>	753
21.17.2.72	<code>icc_powerOnICC(ref byte[] ATR, byte interfaces, string ident="")</code>	754
21.17.2.73	<code>cd_retrieveMessage(DisplayMessages.DISPLAY_MESSAGE_LANGUAGE ← E lang, DisplayMessages.DISPLAY_MESSAGE_IDENTIFIER id, ref string line1, ref string line2)</code>	754
21.17.2.74	<code>msr_cancelMSRSwipe(string ident="")</code>	754

21.17.2.75	<code>msr_startMSRSwipe(int timeout, string ident=(""))</code>	754
21.17.2.76	<code>msr_switchUSBInterfaceMode(bool blsUSBKeyboardMode)</code>	756
21.17.2.77	<code>msr_switchUSBInterfaceMode(bool blsUSBKeyboardMode, ref string ident)</code>	756
21.17.2.78	<code>SDK_Version()</code>	756
21.17.2.79	<code>setCallback(CallBack my_Callback)</code>	757
21.17.2.80	<code>setCallback(IntPtr my_Callback, SynchronizationContext context)</code>	757
21.17.2.81	<code>setCommandTimeout(int milliseconds, string ident=(""))</code>	757
21.17.3	Property Documentation	757
21.17.3.1	SharedController	757
21.18	IDTechSDK.IDT_Vendi Class Reference	757
21.18.1	Member Function Documentation	759
21.18.1.1	<code>config_getSerialNumber(ref string response, string ident=(""))</code>	759
21.18.1.2	<code>config_setBaudRate(int baud, string ident=(""))</code>	759
21.18.1.3	<code>ctls_activateTransaction(int timeout, byte[] tags, bool isFastEMV=false, string ident=(""))</code>	760
21.18.1.4	<code>ctls_cancelTransaction(string ident=(""))</code>	761
21.18.1.5	<code>ctls_getAllConfigurationGroups(ref byte[][] response, string ident=(""))</code>	761
21.18.1.6	<code>ctls_getConfigurationGroup(int group, ref byte[] tlv, string ident=(""))</code>	762
21.18.1.7	<code>ctls_removeAllCAPK(string ident=(""))</code>	762
21.18.1.8	<code>ctls_removeApplicationData(byte[] AID, string ident=(""))</code>	762
21.18.1.9	<code>ctls_removeCAPK(byte[] capk, string ident=(""))</code>	762
21.18.1.10	<code>ctls_removeConfigurationGroup(int group, string ident=(""))</code>	763
21.18.1.11	<code>ctls_retrieveAIDList(ref byte[][] response, string ident=(""))</code>	763
21.18.1.12	<code>ctls_retrieveApplicationData(byte[] AID, ref byte[] tlv, string ident=(""))</code>	763
21.18.1.13	<code>ctls_retrieveCAPK(byte[] capk, ref byte[] key, string ident=(""))</code>	764
21.18.1.14	<code>ctls_retrieveCAPKList(ref byte[] keys, string ident=(""))</code>	764
21.18.1.15	<code>ctls_retrieveTerminalData(ref byte[] tlv, string ident=(""))</code>	764
21.18.1.16	<code>ctls_setApplicationData(byte[] tlv, string ident=(""))</code>	765
21.18.1.17	<code>ctls_setCAPK(byte[] key, string ident=(""))</code>	765
21.18.1.18	<code>ctls_setConfigurationGroup(byte[] tlv, string ident=(""))</code>	766
21.18.1.19	<code>ctls_setDefaultConfiguration(string ident=(""))</code>	766
21.18.1.20	<code>ctls_setTerminalData(byte[] tlv, string ident=(""))</code>	767
21.18.1.21	<code>ctls_startTransaction(double amount, double amtOther, int exponent, int type, int timeout, byte[] tags, bool isFastEMV=false, string ident=(""))</code>	767
21.18.1.22	<code>ctls_trySetTerminalData(byte[] tlv, ref byte[] rejectedTLV, ref byte[] convertedTLV, string ident=(""))</code>	768
21.18.1.23	<code>device_activateTransaction(int timeout, byte[] tags, bool isFastEMV=false, string ident=(""))</code>	769
21.18.1.24	<code>device_controlLED(byte indexLED, byte control, string ident=(""))</code>	770
21.18.1.25	<code>device_controlUserInterface(byte[] values, string ident=(""))</code>	770

21.18.1.26	<code>device_getAnyRKIStatus(bool isMultiKey, ref string status, ref Dictionary< string, RKI_KEY_TYPE > keys, string ident="")</code>	771
21.18.1.27	<code>device_getConfigurationFromMemory(ref string json, string ident="")</code>	772
21.18.1.28	<code>device_getFirmwareVersion(ref string response, string ident="")</code>	772
21.18.1.29	<code>device_getMerchantRecord(int index, ref byte[] record, string ident="")</code>	772
21.18.1.30	<code>device_getRKIStatus(bool isProd, bool isMultiKey, ref string status, ref Dictionary< string, RKI_KEY_TYPE > keys, string ident="")</code>	772
21.18.1.31	<code>device_getTransactionResults(ref IDTTransactionData results, string ident="")</code>	773
21.18.1.32	<code>device_pingDevice(string ident="")</code>	773
21.18.1.33	<code>device_readConfigurationToMemory(string memo, Configs.InstallRules rules, List< Configs.DeviceCommand > cmds, string ident="", bool isForeground=false)</code>	773
21.18.1.34	<code>device_RemoteKeyInjection(RKI_KEY_TYPE type, string keyName, string ident="", bool performOnForeground=false)</code>	774
21.18.1.35	<code>device_retrieveAIDList(ref byte[][] response, string ident="")</code>	774
21.18.1.36	<code>device_sendConfiguration(string filename, VC_OPERATION_TYPE type, bool matchFW, string ident="", bool isForeground=false)</code>	775
21.18.1.37	<code>device_sendConfigurationFromZip(byte[] zip, string filename, VC_OPERATION_TYPE type, bool matchFW, string ident="", bool isForeground=false)</code>	775
21.18.1.38	<code>device_sendDataCommand(string cmd, bool calcLRC, ref byte[] response, string ident="")</code>	776
21.18.1.39	<code>device_sendDataCommand_ext(string cmd, bool calcLRC, ref byte[] response, int timeout, bool noResponse, string ident="")</code>	777
21.18.1.40	<code>device_sendPAE(string command, ref string response, int timeout, string ident="")</code>	777
21.18.1.41	<code>device_sendVivoCommandP2(byte command, byte subCommand, byte[] data, ref byte[] response, string ident="")</code>	777
21.18.1.42	<code>device_sendVivoCommandP2_ext(byte command, byte subCommand, byte[] data, ref byte[] response, int timeout, bool noResponse, string ident="")</code>	778
21.18.1.43	<code>device_setBurstMode(byte mode, string ident="")</code>	778
21.18.1.44	<code>device_setMerchantRecord(int index, bool enabled, string merchantID, string merchantURL, string ident="")</code>	779
21.18.1.45	<code>device_setPollMode(byte mode, string ident="")</code>	779
21.18.1.46	<code>device_startRKI(string ident="")</code>	779
21.18.1.47	<code>device_startTransaction(double amount, double amtOther, int exponent, int type, int timeout, byte[] tags, bool isFastEMV=false, string ident="")</code>	780
21.18.1.48	<code>device_updateDeviceFirmware(byte[] firmwareData, string ident="")</code>	781
21.18.1.49	<code>device_updateFirmwareFromZip(byte[] zipfile, string ident="", bool isForeground=false)</code>	782
21.18.1.50	<code>getCommandTimeout(string ident="")</code>	782
21.18.1.51	<code>lcd_retrieveMessage(DisplayMessages.DISPLAY_MESSAGE_LANGUAGE lang, DisplayMessages.DISPLAY_MESSAGE_IDENTIFIER id, ref string line1, ref string line2)</code>	783
21.18.1.52	<code>msr_cancelMSRSwipe(string ident="")</code>	783
21.18.1.53	<code>msr_startMSRSwipe(int timeout, string ident="")</code>	783
21.18.1.54	<code>SDK_Version()</code>	783

21.18.1.55	setCallback(CallBack my_Callback)	784
21.18.1.56	setCallback(IntPtr my_Callback, SynchronizationContext context)	784
21.18.1.57	setCommandTimeout(int milliseconds, string ident="")	784
21.18.1.58	useSerialPort(int port)	784
21.18.1.59	useSerialPort(int port, int baud)	785
21.18.1.60	useSerialPortLinux(string path)	785
21.18.1.61	useSerialPortLinux(string path, int baud)	785
21.18.2	Property Documentation	785
21.18.2.1	SharedController	785
21.19	IDTechSDK.IDT_VendIII Class Reference	786
21.19.1	Member Function Documentation	787
21.19.1.1	config_getSerialNumber(ref string response, string ident="")	787
21.19.1.2	config_setBaudRate(int baud, string ident="")	788
21.19.1.3	ctls_activateTransaction(int timeout, byte[] tags, bool isFastEMV=false, string ident="")	788
21.19.1.4	ctls_cancelTransaction(string ident="")	789
21.19.1.5	ctls_getAllConfigurationGroups(ref byte[][] response, string ident="")	790
21.19.1.6	ctls_getConfigurationGroup(int group, ref byte[] tlv, string ident="")	790
21.19.1.7	ctls_removeAllCAPK(string ident="")	790
21.19.1.8	ctls_removeApplicationData(byte[] AID, string ident="")	791
21.19.1.9	ctls_removeCAPK(byte[] capk, string ident="")	791
21.19.1.10	ctls_removeConfigurationGroup(int group, string ident="")	791
21.19.1.11	ctls_retrieveAIDList(ref byte[][] response, string ident="")	792
21.19.1.12	ctls_retrieveApplicationData(byte[] AID, ref byte[] tlv, string ident="")	792
21.19.1.13	ctls_retrieveCAPK(byte[] capk, ref byte[] key, string ident="")	792
21.19.1.14	ctls_retrieveCAPKList(ref byte[] keys, string ident="")	793
21.19.1.15	ctls_retrieveTerminalData(ref byte[] tlv, string ident="")	793
21.19.1.16	ctls_setApplicationData(byte[] tlv, string ident="")	794
21.19.1.17	ctls_setCAPK(byte[] key, string ident="")	794
21.19.1.18	ctls_setConfigurationGroup(byte[] tlv, string ident="")	795
21.19.1.19	ctls_setDefaultConfiguration(string ident="")	795
21.19.1.20	ctls_setTerminalData(byte[] tlv, string ident="")	795
21.19.1.21	ctls_startTransaction(double amount, double amtOther, int exponent, int type, int timeout, byte[] tags, bool isFastEMV=false, string ident="")	796
21.19.1.22	ctls_trySetTerminalData(byte[] tlv, ref byte[] rejectedTLV, ref byte[] convertedTLV, string ident="")	797
21.19.1.23	device_activateTransaction(int timeout, byte[] tags, bool isFastEMV=false, string ident="")	797
21.19.1.24	device_controlLED(byte indexLED, byte control, string ident="")	798
21.19.1.25	device_controlUserInterface(byte[] values, string ident="")	798

21.19.1.26	<code>device_getAnyRKIStatus(bool isMultiKey, ref string status, ref Dictionary< string, RKI_KEY_TYPE > keys, string ident=(""))</code>	799
21.19.1.27	<code>device_getConfigurationFromMemory(ref string json, string ident=(""))</code>	799
21.19.1.28	<code>device_getFirmwareVersion(ref string response, string ident=(""))</code>	800
21.19.1.29	<code>device_getMerchantRecord(int index, ref byte[] record, string ident=(""))</code>	800
21.19.1.30	<code>device_getRKIStatus(bool isProd, bool isMultiKey, ref string status, ref Dictionary< string, RKI_KEY_TYPE > keys, string ident=(""))</code>	800
21.19.1.31	<code>device_getTransactionResults(ref IDTTransactionData results, string ident=(""))</code>	800
21.19.1.32	<code>device_pingDevice(string ident=(""))</code>	801
21.19.1.33	<code>device_readConfigurationToMemory(string memo, Configs.InstallRules rules, List< Configs.DeviceCommand > cmds, string ident= "", bool isForeground=false)</code>	801
21.19.1.34	<code>device_RemoteKeyInjection(RKI_KEY_TYPE type, string keyName, string ident= "", bool performOnForeground=false)</code>	802
21.19.1.35	<code>device_retrieveAIDList(ref byte[][] response, string ident= "")</code>	802
21.19.1.36	<code>device_sendConfiguration(string filename, VC_OPERATION_TYPE type, bool matchFW, string ident= "", bool isForeground=false)</code>	802
21.19.1.37	<code>device_sendConfigurationFromZip(byte[] zip, string filename, VC_OPERATION_TYPE type, bool matchFW, string ident= "", bool isForeground=false)</code>	803
21.19.1.38	<code>device_sendDataCommand(string cmd, bool calcLRC, ref byte[] response, string ident= "")</code>	804
21.19.1.39	<code>device_sendDataCommand_ext(string cmd, bool calcLRC, ref byte[] response, int timeout, bool noResponse, string ident= "")</code>	804
21.19.1.40	<code>device_sendPAE(string command, ref string response, int timeout, string ident= "")</code>	805
21.19.1.41	<code>device_sendVivoCommandP2(byte command, byte subCommand, byte[] data, ref byte[] response, string ident= "")</code>	805
21.19.1.42	<code>device_sendVivoCommandP2_ext(byte command, byte subCommand, byte[] data, ref byte[] response, int timeout, bool noResponse, string ident= "")</code>	805
21.19.1.43	<code>device_setBurstMode(byte mode, string ident= "")</code>	806
21.19.1.44	<code>device_setMerchantRecord(int index, bool enabled, string merchantID, string merchantURL, string ident= "")</code>	806
21.19.1.45	<code>device_setPollMode(byte mode, string ident= "")</code>	806
21.19.1.46	<code>device_startRKI(string ident= "")</code>	807
21.19.1.47	<code>device_startTransaction(double amount, double amtOther, int exponent, int type, int timeout, byte[] tags, bool isFastEMV=false, string ident= "")</code>	807
21.19.1.48	<code>device_updateDeviceFirmware(byte[] firmwareData, string ident= "")</code>	808
21.19.1.49	<code>device_updateFirmwareFromZip(byte[] zipfile, string ident= "", bool isForeground=false)</code>	809
21.19.1.50	<code>emv_activateTransaction(int timeout, byte[] tags, bool forceOnline, byte[] returnTags, bool isFastEMV=false, string ident= "")</code>	809
21.19.1.51	<code>emv_completeTransaction(bool commError, byte[] authCode, byte[] iad, byte[] tlvScripts, byte[] tlv, string ident= "")</code>	810
21.19.1.52	<code>emv_startTransaction(double amount, double amtOther, int exponent, int type, int timeout, byte[] tags, bool forceOnline, byte[] returnTags, bool isFastEMV=false, string ident= "")</code>	810

21.19.1.53	<code>cd_retrieveMessage(DisplayMessages.DISPLAY_MESSAGE_LANGUAGE lang, DisplayMessages.DISPLAY_MESSAGE_IDENTIFIER id, ref string line1, ref string line2)</code>	811
21.19.1.54	<code>msr_cancelMSRSwipe(string ident="")</code>	811
21.19.1.55	<code>msr_startMSRSwipe(int timeout, string ident="")</code>	811
21.19.1.56	<code>SDK_Version()</code>	812
21.19.1.57	<code>setCallback(CallBack my_Callback)</code>	812
21.19.1.58	<code>setCallback(IntPtr my_Callback, SynchronizationContext context)</code>	812
21.19.1.59	<code>useSerialPort(int port)</code>	812
21.19.1.60	<code>useSerialPort(int port, int baud)</code>	812
21.19.1.61	<code>useSerialPortLinux(string path)</code>	813
21.19.1.62	<code>useSerialPortLinux(string path, int baud)</code>	813
21.19.2	Property Documentation	813
21.19.2.1	SharedController	813
21.20	IDTechSDK.IDT_VP3300 Class Reference	814
21.20.1	Detailed Description	816
21.20.2	Member Function Documentation	817
21.20.2.1	<code>config_getDate(ref int year, ref int month, ref int day, string ident="")</code>	817
21.20.2.2	<code>config_getEncryptionControl(ref bool msr, ref bool icc, string ident="")</code>	818
21.20.2.3	<code>config_getSerialNumber(ref string response, string ident="")</code>	818
21.20.2.4	<code>config_getTime(ref int hour, ref int minute, string ident="")</code>	819
21.20.2.5	<code>config_setDate(int year, int month, int day, string ident="")</code>	819
21.20.2.6	<code>config_setEncryptionControl(bool msr, bool icc, string ident="")</code>	819
21.20.2.7	<code>createFastEMVData(ref IDTTransactionData cData, string ident="")</code>	820
21.20.2.8	<code>ctls_activateTransaction(int timeout, byte[] tags, bool forceOnline, bool isFastEMV=false, string ident="")</code>	820
21.20.2.9	<code>ctls_cancelTransaction(string ident="")</code>	822
21.20.2.10	<code>ctls_getAllConfigurationGroups(ref byte[][] response, string ident="")</code>	822
21.20.2.11	<code>ctls_getConfigurationGroup(int group, ref byte[] tlv, string ident="")</code>	822
21.20.2.12	<code>ctls_removeAllCAPK(string ident="")</code>	823
21.20.2.13	<code>ctls_removeApplicationData(byte[] AID, string ident="")</code>	823
21.20.2.14	<code>ctls_removeCAPK(byte[] capk, string ident="")</code>	823
21.20.2.15	<code>ctls_removeConfigurationGroup(int group, string ident="")</code>	823
21.20.2.16	<code>ctls_retrieveAIDList(ref byte[][] response, string ident="")</code>	824
21.20.2.17	<code>ctls_retrieveApplicationData(byte[] AID, ref byte[] tlv, string ident="")</code>	824
21.20.2.18	<code>ctls_retrieveCAPK(byte[] capk, ref byte[] key, string ident="")</code>	824
21.20.2.19	<code>ctls_retrieveCAPKList(ref byte[] keys, string ident="")</code>	825
21.20.2.20	<code>ctls_retrieveTerminalData(ref byte[] tlv, string ident="")</code>	825
21.20.2.21	<code>ctls_setApplicationData(byte[] tlv, string ident="")</code>	826
21.20.2.22	<code>ctls_setCAPK(byte[] key, string ident="")</code>	826
21.20.2.23	<code>ctls_setConfigurationGroup(byte[] tlv, string ident="")</code>	827

21.20.2.24	ctls_setDefaultConfiguration(string ident="")	827
21.20.2.25	ctls_setTerminalData(byte[] tlv, string ident="")	827
21.20.2.26	ctls_startTransaction(double amount, double amtOther, int exponent, int type, int timeout, byte[] tags, bool forceOnline, bool isFastEMV=false, string ident="")	828
21.20.2.27	ctls_trySetTerminalData(byte[] tlv, ref byte[] rejectedTLV, ref byte[] convertedTLV, string ident="")	829
21.20.2.28	device_activateTransaction(int timeout, byte[] tags, bool isFastEMV=false, string ident="")	829
21.20.2.29	device_buzzer(string ident="")	831
21.20.2.30	device_cancelTransaction(string ident="")	831
21.20.2.31	device_controlLED(byte indexLED, byte control, string ident="")	831
21.20.2.32	device_controlUserInterface(byte[] values, string ident="")	832
21.20.2.33	device_enablePassThrough(bool enablePassThrough, string ident="")	834
21.20.2.34	device_getAnyRKIStatus(bool isMultiKey, ref string status, ref Dictionary< string, RKI_KEY_TYPE > keys, string ident="")	834
21.20.2.35	device_getCashTranRiskPara(ref byte[] tlv, string ident="")	834
21.20.2.36	device_getConfigurationFromMemory(ref string json, string ident="")	834
21.20.2.37	device_getDateTime(ref byte[] dateTime, string ident="")	835
21.20.2.38	device_getDrlReaderRiskPara(byte index, ref byte[] tlv, string ident="")	835
21.20.2.39	device_getFirmwareVersion(ref string response, string ident="")	836
21.20.2.40	device_getHardwareInfor(ref string ascii, string ident="")	836
21.20.2.41	device_getMerchantRecord(int index, ref byte[] record, string ident="")	836
21.20.2.42	device_getModuleVer(ref string moduleVer, string ident="")	837
21.20.2.43	device_getPollMode(ref byte mode, string ident="")	837
21.20.2.44	device_getProductType(ref byte[] type, string ident="")	837
21.20.2.45	device_getResponseCodeString(RETURN_CODE eCode)	838
21.20.2.46	device_getRKIStatus(bool isProd, bool isMultiKey, ref string status, ref Dictionary< string, RKI_KEY_TYPE > keys, string ident="")	847
21.20.2.47	device_getTransactionResults(ref IDTTransactionData results, string ident="")	848
21.20.2.48	device_getUIDofMCU(ref string uid, string ident="")	848
21.20.2.49	device_getUsbBootLoader(ref string bootLoader, string ident="")	848
21.20.2.50	device_pingDevice(string ident="")	849
21.20.2.51	device_readConfigurationToMemory(string memo, Configs.InstallRules rules, List< Configs.DeviceCommand > cmds, string ident="", bool isForeground=false)	849
21.20.2.52	device_rebootDevice(string ident="")	849
21.20.2.53	device_RemoteKeyInjection(RKI_KEY_TYPE type, string keyName, string ident="", bool performOnForeground=false)	850
21.20.2.54	device_sendConfiguration(string filename, VC_OPERATION_TYPE type, bool matchFW, string ident="", bool isForeground=false)	850
21.20.2.55	device_sendConfigurationFromZip(byte[] zip, string filename, VC_OPERATION_TYPE type, bool matchFW, string ident="", bool isForeground=false)	851
21.20.2.56	device_sendDataCommand(string cmd, bool calcLRC, ref byte[] response, string ident="")	852

21.20.2.57	device_sendDataCommand_ext(string cmd, bool calcLRC, ref byte[] response, int timeout, bool noResponse, string ident="")	852
21.20.2.58	device_sendPAE(string command, ref string response, int timeout, string ident="")	853
21.20.2.59	device_sendVivoCommandP2(byte command, byte subCommand, byte[] data, ref byte[] response, string ident="")	853
21.20.2.60	device_sendVivoCommandP2_ext(byte command, byte subCommand, byte[] data, ref byte[] response, int timeout, bool noResponse, string ident="")	853
21.20.2.61	device_setBurstMode(byte mode, string ident="")	854
21.20.2.62	device_setDateTime(string ident="")	854
21.20.2.63	device_setLED(byte indexLED, byte control, string ident="")	854
21.20.2.64	device_setMerchantRecord(int index, bool enabled, string merchantID, string merchantURL, string ident="")	855
21.20.2.65	device_setPollMode(byte mode, string ident="")	855
21.20.2.66	device_setQuickChipHIDMode(byte mode, string ident="")	855
21.20.2.67	device_startRKI(string ident="")	856
21.20.2.68	device_StartRKI(int type, string ident="")	856
21.20.2.69	device_startTransaction(double amount, double amtOther, int exponent, int type, int timeout, byte[] tags, bool isFastEMV=false, string ident="")	856
21.20.2.70	device_updateDeviceFirmware(byte[] firmwareData, string ident="")	858
21.20.2.71	device_updateFirmwareFromZip(byte[] zipfile, string ident="", bool isForeground=false)	859
21.20.2.72	emv_activateTransaction(int timeout, byte[] tags, bool forceOnline, bool isFastEMV=false, string ident="")	859
21.20.2.73	emv_addTerminalData(byte[] tlv, string ident="")	860
21.20.2.74	emv_allowFallback(bool allow, string ident="")	860
21.20.2.75	emv_authenticateTransaction(byte[] updatedTLV, string ident="")	860
21.20.2.76	emv_autoAuthenticate(bool authenticate, string ident="")	860
21.20.2.77	emv_autoAuthenticateTags(bool authenticate, byte[] tags, string ident="")	861
21.20.2.78	emv_callbackResponseLCD(EMV_LCD_DISPLAY_MODE type, byte selection, string ident="")	861
21.20.2.79	emv_callbackResponseMSR(byte[] MSR, string ident="")	861
21.20.2.80	emv_callbackResponsePIN(EMV_PIN_MODE type, byte[] KSN, byte[] PIN, string ident="")	862
21.20.2.81	emv_cancelTransaction(string ident="")	862
21.20.2.82	emv_completeTransaction(bool commError, byte[] authCode, byte[] iad, byte[] tlvScripts, byte[] tlv, string ident="")	862
21.20.2.83	emv_getEMVConfigurationCheckValue(ref string response, string ident="")	863
21.20.2.84	emv_getEMVKernelCheckValue(ref string response, string ident="")	863
21.20.2.85	emv_getEMVKernelVersion(ref string response, string ident="")	863
21.20.2.86	emv_getEMVKernelVersionExt(ref string response, string ident="")	864
21.20.2.87	emv_getTerminalMajorConfiguration(ref int configuration, string ident="")	864
21.20.2.88	emv_removeAllApplicationData(string ident="")	864
21.20.2.89	emv_removeAllCAPK(string ident="")	865

21.20.2.90	<code>emv_removeAllCRL(string ident="")</code>	865
21.20.2.91	<code>emv_removeApplicationData(byte[] AID, string ident="")</code>	865
21.20.2.92	<code>emv_removeCAPK(byte[] capk, string ident="")</code>	865
21.20.2.93	<code>emv_removeCRL(byte[] crlList, string ident="")</code>	866
21.20.2.94	<code>emv_removeTerminalData(string ident="")</code>	866
21.20.2.95	<code>emv_retrieveAIDList(ref byte[][] response, string ident="")</code>	866
21.20.2.96	<code>emv_retrieveApplicationData(byte[] AID, ref byte[] tlv, string ident="")</code>	867
21.20.2.97	<code>emv_retrieveCAPK(byte[] capk, ref byte[] key, string ident="")</code>	867
21.20.2.98	<code>emv_retrieveCAPKList(ref byte[] keys, string ident="")</code>	868
21.20.2.99	<code>emv_retrieveCRLList(ref byte[] list, string ident="")</code>	868
21.20.2.100	<code>emv_retrieveTerminalData(ref byte[] tlv, string ident="")</code>	868
21.20.2.101	<code>emv_retrieveTransactionResult(byte[] tags, ref IDTTransactionData tlv, string ident="")</code>	868
21.20.2.102	<code>emv_setApplicationData(byte[] name, byte[] tlv, string ident="")</code>	869
21.20.2.103	<code>emv_setCAPK(byte[] key, string ident="")</code>	869
21.20.2.104	<code>emv_setCRL(byte[] list, string ident="")</code>	870
21.20.2.105	<code>emv_setTerminalData(byte[] tlv, string ident="")</code>	870
21.20.2.106	<code>emv_setTerminalMajorConfiguration(int configuration, string ident="")</code>	870
21.20.2.107	<code>emv_startTransaction(double amount, double amtOther, int exponent, int type, int timeout, byte[] tags, bool forceOnline, bool isFastEMV=false, string ident="")</code>	871
21.20.2.108	<code>emv_trySetTerminalData(byte[] tlv, ref byte[] rejectedTLV, ref byte[] convertedTLV, bool overwrite=false, string ident="")</code>	871
21.20.2.109	<code>GetCommandTimeout(string ident="")</code>	872
21.20.2.110	<code>c_exchangeAPDU(string c_APDU, ref byte[] response, string ident="")</code>	872
21.20.2.111	<code>c_getICCRReaderStatus(ref byte status, string ident="")</code>	872
21.20.2.112	<code>c_powerOffICC(string ident="")</code>	872
21.20.2.113	<code>c_powerOnICC(ref byte[] ATR, byte interfaces, string ident="")</code>	873
21.20.2.114	<code>d_retrieveMessage(DisplayMessages.DISPLAY_MESSAGE_LANGUAGE lang, DisplayMessages.DISPLAY_MESSAGE_IDENTIFIER id, ref string line1, ref string line2)</code>	873
21.20.2.115	<code>msr_cancelMSRSwipe(string ident="")</code>	873
21.20.2.116	<code>msr_startMSRSwipe(int timeout, string ident="")</code>	874
21.20.2.117	<code>SDK_Version()</code>	874
21.20.2.118	<code>SetCallback(CallBack my_Callback)</code>	874
21.20.2.119	<code>SetCallback(IntPtr my_Callback, SynchronizationContext context)</code>	874
21.20.2.120	<code>SetCommandTimeout(int milliseconds, string ident="")</code>	875
21.20.2.121	<code>UseSerialPort(int port)</code>	875
21.20.2.122	<code>UseSerialPort(int port, int baud)</code>	875
21.20.3	Property Documentation	875
21.20.3.1	SharedController	875
21.21	IDTechSDK.IDT_VP8800 Class Reference	876

21.21.1 Member Function Documentation	879
21.21.1.1 config_getDataKeySlot(ref int slot, string ident="")	879
21.21.1.2 config_getEthernetMACAddress(ref byte[] address, string ident="")	880
21.21.1.3 config_getKeySlotInfo(int index, int slot, ref byte[] keyslot, string ident="")	880
21.21.1.4 config_getNetworkConfiguration(ref bool isStatic, ref string address, ref string subnet, ref string gateway, ref string dns, string ident="")	880
21.21.1.5 config_getPINKeySlot(ref int slot, string ident="")	881
21.21.1.6 config_getSerialNumber(ref string response, string ident="")	881
21.21.1.7 config_getStatusKeySlots(ref byte[] keyslots, string ident="")	881
21.21.1.8 config_setDataKeySlot(int slot, string ident="")	882
21.21.1.9 config_setEthernetMACAddress(byte[] address, string ident="")	882
21.21.1.10 config_setNetworkConfiguration(bool isStatic, string address, string subnet, string gateway, string dns, string ident="")	882
21.21.1.11 config_setPINKeySlot(int slot, string ident="")	883
21.21.1.12 createFastEMVData(ref IDTTTransactionData cData, string ident="")	883
21.21.1.13 ctls_activateTransaction (int timeout, byte[] tags, bool forceOnline, bool isFastEMV←MV=false, string ident="")	883
21.21.1.14 ctls_cancelTransaction (string ident="")	884
21.21.1.15 ctls_displayOnlineAuthResult (bool isOK, byte[] TLV, string ident="")	885
21.21.1.16 ctls_displayOnlineAuthResult_ext (byte statusCode, byte[] TLV, string ident="")	885
21.21.1.17 ctls_getConfigurationGroup (int group, ref byte[] tlv, string ident="")	885
21.21.1.18 ctls_nfcCommand (byte[] nfcCmdPkt, ref byte[] response, string ident="")	886
21.21.1.19 ctls_removeAllCAPK (string ident="")	887
21.21.1.20 ctls_removeApplicationData (byte[] AID, string ident="")	888
21.21.1.21 ctls_removeCAPK (byte[] capk, string ident="")	888
21.21.1.22 ctls_removeConfigurationGroup (int group, string ident="")	888
21.21.1.23 ctls_resetConfigurationGroup (int group, string ident="")	889
21.21.1.24 ctls_retrieveAIDList (ref byte[][] response, string ident="")	889
21.21.1.25 ctls_retrieveApplicationData (byte[] AID, ref byte[] tlv, string ident="")	889
21.21.1.26 ctls_retrieveCAPK (byte[] capk, ref byte[] key, string ident="")	889
21.21.1.27 ctls_retrieveCAPKList (ref byte[] keys, string ident="")	890
21.21.1.28 ctls_retrieveTerminalData (ref byte[] tlv, string ident="")	890
21.21.1.29 ctls_setApplicationData (byte[] tlv, string ident="")	891
21.21.1.30 ctls_setCAPK (byte[] key, string ident="")	891
21.21.1.31 ctls_setConfigurationGroup (byte[] tlv, string ident="")	892
21.21.1.32 ctls_setDefaultConfiguration (string ident="")	892
21.21.1.33 ctls_setTerminalData (byte[] tlv, string ident="")	893
21.21.1.34 ctls_startTransaction (double amount, double amtOther, int exponent, int type, int timeout, byte[] tags, bool forceOnline, bool isFastEMV=false, string ident="")	893
21.21.1.35 ctls_trySetTerminalData (byte[] tlv, ref byte[] rejectedTLV, ref byte[] convertedTLV, string ident="")	894

21.21.1.36	<code>device_activateTransaction(int timeout, byte[] tags, byte[] options, bool isFastE← MV=false, string ident="")</code>	895
21.21.1.37	<code>device_buzzer(string ident="")</code>	896
21.21.1.38	<code>device_calibrateParameters(byte delta, string ident="")</code>	896
21.21.1.39	<code>device_cancelTransaction(string ident="")</code>	896
21.21.1.40	<code>device_controlIndicator(int indicator, bool enable, string ident="")</code>	897
21.21.1.41	<code>device_controlLED(byte indexLED, byte control, string ident="")</code>	897
21.21.1.42	<code>device_controlUserInterface(byte[] values, string ident="")</code>	897
21.21.1.43	<code>device_createDirectory(string directoryName, string ident="")</code>	899
21.21.1.44	<code>device_deleteDirectory(string filename, string ident="")</code>	899
21.21.1.45	<code>device_deleteFile(string filename, bool isSD=false, string ident="")</code>	899
21.21.1.46	<code>device_enablePassThrough(bool enablePassThrough, string ident="")</code>	900
21.21.1.47	<code>device_getConfigurationFromMemory(ref string json, string ident="")</code>	900
21.21.1.48	<code>device_getConfigurationGroup(int group, ref byte[] tlv, string ident="")</code>	900
21.21.1.49	<code>device_getDriveFreeSpace(ref int free, ref int used, string ident="")</code>	900
21.21.1.50	<code>device_getFirmwareVersion(ref string response, string ident="")</code>	901
21.21.1.51	<code>device_getMerchantRecord(int index, ref byte[] record, string ident="")</code>	901
21.21.1.52	<code>device_getResponseCodeString(RETURN_CODE eCode)</code>	901
21.21.1.53	<code>device_getTransactionResults(ref IDTTransactionData results, string ident="")</code>	911
21.21.1.54	<code>device_listDirectory(string directoryName, bool recursive, bool onSD, ref string directory, string ident="")</code>	912
21.21.1.55	<code>device_pingDevice(string ident="")</code>	912
21.21.1.56	<code>device_readConfigurationToMemory(string memo, Configs.InstallRules rules, List< Configs.DeviceCommand > cmds, string ident="", bool isForeground=false)</code>	912
21.21.1.57	<code>device_RemoteKeyInjection(RKI_KEY_TYPE type, string keyName, string ident="", bool performOnForeground=false)</code>	913
21.21.1.58	<code>device_removeConfigurationGroup(int group, string ident="")</code>	913
21.21.1.59	<code>device_resetConfigurationGroup(int group, string ident="")</code>	913
21.21.1.60	<code>device_retrieveAIDList(ref byte[][] response, string ident="")</code>	914
21.21.1.61	<code>device_sendConfiguration(string filename, VC_OPERATION_TYPE type, bool matchFW, string ident="", bool isForeground=false)</code>	914
21.21.1.62	<code>device_sendConfigurationFromZip(byte[] zip, string filename, VC_OPERATION← _TYPE type, bool matchFW, string ident="", bool isForeground=false)</code>	915
21.21.1.63	<code>device_sendDataCommand(string cmd, bool calcLRC, ref byte[] response, string ident="")</code>	916
21.21.1.64	<code>device_sendDataCommand_ext(string cmd, bool calcLRC, ref byte[] response, int timeout, bool noResponse, string ident="")</code>	916
21.21.1.65	<code>device_sendPAE(string command, ref string response, int timeout, string ident="")</code>	916
21.21.1.66	<code>device_sendVivoCommandP2(byte command, byte subCommand, byte[] data, ref byte[] response, string ident="")</code>	917
21.21.1.67	<code>device_sendVivoCommandP2_ext(byte command, byte subCommand, byte[] data, ref byte[] response, int timeout, bool noResponse, string ident="")</code>	917

21.21.1.68	device_sendVivoCommandP3(byte command, byte subCommand, byte[] data, ref byte[] response, string ident="")	918
21.21.1.69	device_sendVivoCommandP3_ext(byte command, byte subCommand, byte[] data, ref byte[] response, int timeout, bool noResponse, string ident="")	918
21.21.1.70	device_setBurstMode(byte mode, string ident="")	918
21.21.1.71	device_setBuzzerLED(byte buzzer, byte led, bool ledON, string ident="")	919
21.21.1.72	device_setMerchantRecord(int index, bool enabled, string merchantID, string merchantURL, string ident="")	919
21.21.1.73	device_setPollMode(byte mode, string ident="")	920
21.21.1.74	device_startRKI(bool isTest, string ident="")	920
21.21.1.75	device_StartRKI(int type, string ident="")	920
21.21.1.76	device_startTransaction(double amount, double amtOther, int exponent, int type, int timeout, byte[] tags, byte[] options, bool isFastEMV=false, int transMode=0, string ident="")	921
21.21.1.77	device_transferFile(string fileName, byte[] file, bool isSD=false, string ident="")	922
21.21.1.78	device_updateDeviceFirmware(byte[] firmwareData, string ident="")	922
21.21.1.79	device_updateFirmwareFromZip(byte[] zipfile, string ident="", bool isForeground=false)	924
21.21.1.80	emv_activateTransaction(int timeout, byte[] tags, bool forceOnline, bool isFastEMV=false, string ident="")	924
21.21.1.81	emv_addTerminalData(byte[] tlv, string ident="")	925
21.21.1.82	emv_allowFallback(bool allow, string ident="")	925
21.21.1.83	emv_autoAuthenticate(bool authenticate, string ident="")	925
21.21.1.84	emv_autoAuthenticateTags(bool authenticate, byte[] tags, string ident="")	925
21.21.1.85	emv_callbackResponseLCD(EMV_LCD_DISPLAY_MODE type, byte selection, string ident="")	926
21.21.1.86	emv_callbackResponseMSR(byte[] MSR, string ident="")	926
21.21.1.87	emv_callbackResponsePIN(EMV_PIN_MODE type, byte[] KSN, byte[] PIN, string ident="")	926
21.21.1.88	emv_cancelTransaction(string ident="")	927
21.21.1.89	emv_clearTransactionLog(string ident="")	927
21.21.1.90	emv_completeTransaction(bool commError, byte[] authCode, byte[] iad, byte[] tlvScripts, byte[] tlv, string ident="")	927
21.21.1.91	emv_continueTransactionForCV(int result, byte[] pinblock, string ident="")	928
21.21.1.92	emv_getConfigurationGroup(int group, ref byte[] tlv, string ident="")	928
21.21.1.93	emv_getCRLStatus(ref byte[] status, string ident="")	929
21.21.1.94	emv_getEMVConfigurationCheckValue(ref string response, string ident="")	929
21.21.1.95	emv_getEMVKernelCheckValue(ref string response, string ident="")	929
21.21.1.96	emv_getEMVKernelVersion(ref string response, string ident="")	930
21.21.1.97	emv_getEMVKernelVersionExt(ref string response, string ident="")	930
21.21.1.98	emv_getExemptionLogStatus(ref byte[] status, string ident="")	930
21.21.1.99	emv_getTransactionLogRecord(ref byte[] record, ref int remaining, string ident="")	930
21.21.1.100	emv_getTransactionLogStatus(ref byte[] status, string ident="")	932

21.21.1.101	<code>mv_removeAllCAPK(string ident="")</code>	932
21.21.1.102	<code>mv_removeAllCRL(string ident="")</code>	932
21.21.1.103	<code>mv_removeAllExceptions(string ident="")</code>	933
21.21.1.104	<code>mv_removeApplicationData(byte[] AID, string ident="")</code>	933
21.21.1.105	<code>mv_removeCAPK(byte[] capk, string ident="")</code>	933
21.21.1.106	<code>mv_removeConfigurationGroup(int group, string ident="")</code>	933
21.21.1.107	<code>mv_removeCRL(byte[] crl, string ident="")</code>	934
21.21.1.108	<code>mv_removeException(byte[] exception, string ident="")</code>	934
21.21.1.109	<code>mv_removeTerminalData(string ident="")</code>	934
21.21.1.110	<code>mv_removeTransactionAmountLog(string ident="")</code>	935
21.21.1.111	<code>mv_resetConfigurationGroup(int group, string ident="")</code>	935
21.21.1.112	<code>mv_retrieveAIDList(ref byte[][] response, string ident="")</code>	935
21.21.1.113	<code>mv_retrieveApplicationData(byte[] AID, ref byte[] tlv, string ident="")</code>	936
21.21.1.114	<code>mv_retrieveCAPK(byte[] capk, ref byte[] key, string ident="")</code>	936
21.21.1.115	<code>mv_retrieveCAPKList(ref byte[] keys, string ident="")</code>	936
21.21.1.116	<code>mv_retrieveCRLList(ref byte[] list, string ident="")</code>	937
21.21.1.117	<code>mv_retrieveExceptionList(ref byte[] list, string ident="")</code>	937
21.21.1.118	<code>mv_retrieveTerminalData(ref byte[] tlv, string ident="")</code>	937
21.21.1.119	<code>mv_retrieveTransactionResult(byte[] tags, ref IDTTransactionData tlv, string ident="")</code>	938
21.21.1.120	<code>mv_setApplicationData(byte[] tlv, string ident="")</code>	938
21.21.1.121	<code>mv_setCAPK(byte[] key, string ident="")</code>	939
21.21.1.122	<code>mv_setCRL(byte[] crl, string ident="")</code>	939
21.21.1.123	<code>mv_setException(byte[] exception, string ident="")</code>	940
21.21.1.124	<code>mv_setTerminalDataVP8800(byte[] tlv, int config, string ident="")</code>	940
21.21.1.125	<code>mv_startTransaction(double amount, double amtOther, int exponent, int type, int timeout, byte[] tags, bool forceOnline, string ident="")</code>	940
21.21.1.126	<code>mv_trySetTerminalData(byte[] tlv, ref byte[] rejectedTLV, ref byte[] convertedTLV, bool overwrite=false, string ident="")</code>	941
21.21.1.127	<code>lica_authentication(byte[] key, string ident="")</code>	941
21.21.1.128	<code>lica_read(byte[] serviceCode, int numBlocks, byte[] blockList, ref byte[] blocks, string ident="")</code>	942
21.21.1.129	<code>lica_readWithMac(int numBlocks, byte[] blockList, ref byte[] blocks, string ident="")</code>	942
21.21.1.130	<code>lica_requestService(byte[] nodeCode, ref byte[] response, string ident="")</code>	943
21.21.1.131	<code>lica_SendCommand(byte[] command, ref byte[] response, string ident="")</code>	943
21.21.1.132	<code>lica_write(byte[] serviceCode, int blockCount, byte[] blockList, byte[] data, ref byte[] statusFlag, string ident="")</code>	943
21.21.1.133	<code>lica_writeWithMac(int blockNumber, byte[] data, string ident="")</code>	944
21.21.1.134	<code>letCommandTimeout(string ident="")</code>	944
21.21.1.135	<code>c_exchangeAPDU(string c_APDU, ref byte[] response, string ident="")</code>	944
21.21.1.136	<code>c_powerOffICC(string ident="")</code>	944

21.21.1.137c_powerOnICC(ref byte[] ATR, byte interfaces, string ident="")	945
21.21.1.138d_captureSignature(int timeout, int format, string ident="")	945
21.21.1.139d_clearDisplay(string ident="")	945
21.21.1.140d_clearInputEvents(string ident="")	946
21.21.1.141d_customDisplayMode(bool enable, string ident="")	946
21.21.1.142d_displayButton(int xCord, int yCord, int xWidth, int yWidth, int font, int style, int displayProp, string text, ref byte[] identifier, byte[] foreRGB=null, byte[] backRGB=null, string ident="")	946
21.21.1.143d_displayText(int xCord, int yCord, int xWidth, int yWidth, int font, int style, int displayProp, string text, ref byte[] identifier, string ident="")	948
21.21.1.144d_getInputEvent(byte timeout, string ident="")	949
21.21.1.145d_resetInitialState(string ident="")	950
21.21.1.146d_retrieveMessage(DisplayMessages.DISPLAY_MESSAGE_LANGUAGE lang, DisplayMessages.DISPLAY_MESSAGE_IDENTIFIER id, ref string line1, ref string line2)	950
21.21.1.147d_setBackgroundImage(string file, bool enable, string ident="")	951
21.21.1.148d_setDisplayImage(string file, int posX, int posY, int posMode, bool touchEnable, bool clearScreen, string ident="")	951
21.21.1.149d_setForeBackColor(byte[] foreRGB, byte[] backRGB, string ident="")	951
21.21.1.150d_startSlideShow(string files, int posX, int posY, int posMode, bool touchEnable, bool recursion, bool touchTerminate, int delay, int loops, bool clearScreen, string ident="")	952
21.21.1.151msr_cancelMSRSwipe(string ident="")	952
21.21.1.152msr_flushTrackData(string ident="")	953
21.21.1.153msr_startMSRSwipe(int timeout, string ident="")	953
21.21.1.154pin_getEncryptedOnlinePIN(int keyType, int timeout, string ident="")	953
21.21.1.155pin_getPAN(bool getCSC, int timeout, string ident="")	954
21.21.1.156pin_promptCreditDebit(object currencySymbol, string displayAmount, int timeout, string ident="")	954
21.21.1.157DK_Version()	954
21.21.1.158SetCallback(CallBack my_Callback)	954
21.21.1.159SetCallback(IntPtr my_Callback, SynchronizationContext context)	955
21.21.1.160GetCommandTimeout(int milliseconds, string ident="")	955
21.21.1.161UseSerialPort(int port)	955
21.21.1.162UseSerialPort(int port, int baud, string ident="")	955
21.21.1.163UseSerialPortLinux(string path)	956
21.21.1.164UseSerialPortLinux(string path, int baud)	956
21.21.2 Property Documentation	956
21.21.2.1 SharedController	956

Chapter 1

IDTech Windows SDK Reference Guide for Visual Studio Development

IDTechSDK.dll is a Windows dynamic link libraries that will be provided by IDTech as the main interface between Windows Application (.Net Based) and applications, respectively, the IDTech family of devices and payment processing solutions.

The purpose of this document is to describe the requirements of the API as well as the interface definitions and requirements needed for a Visual Studio .Net based application wishing to deploy with the payment application.

- [SDK Class Organization](#)
- [Connecting To IDTech Devices](#)
- [Core Implementation: WinForms](#)
- [Important Security Notice](#)
- [Main Transaction Commands](#)
- [EMV Callback](#)
- [EMV Tag Reference](#)
- [Enumeration Reference](#)
- [Error Code Reference](#)
- [LCD Foreign Language Mapping Table](#)
- [L100 Pass-Through Mode](#)
- [Virtual Device Operation](#)
- [Using SDK on iOS with Xamarin](#)
- [Using SDK on Android with Xamarin](#)
- [Using SDK on iOS/Android with MAUI](#)
- [Using ZMQ Server/Client](#)

Chapter 2

Using ZMQ Server/Client

Zero Messaging Queue (ZMQ) allows you to turn your PC into a server that can service remote requests from IDTech ZMQ Clients.

IDTech ZMQ Server SDK can be utilized to create a server app that controls locally connected IDTech devices (usually USB connected). The ZMQ Server will spin up a TCP/IP listener socket that ZMQ Clients can connect to. This not only allows you ZMQ Clients to be located in remote locations, but also give you the ability to control the IDTech device connected to the ZMQ Server from multiple locations/clients.

2.1 Setting Up ZMQ Server App

Source code and compiled server app can be found at IDTech Knowledge Base at the following link:

https://atlassian.idtechproducts.com/confluence/download/attachments/30479625/IDTech_ZMQ_Server_App_Source_Code.zip?api=v2

For your project, import from NuGet the library IDTechZMQ_Server.

After the library is linked, add it's reference to your form/project

```
using IDTechZMQ_Server;
```

If you are building a GUI interface and would like the server app to display/report information, you have the following four callback you can optionally utilize:

```
void deviceListChanged(List<string> connectedDevices)
{
    //whenever devices are attached/removed, this callback will be updated with the connected devices
}

void clientListChanged(List<string> connectedClients)
{
    //whenever clients are connected to the server, this callback will be updated with the connected clients
}

void deviceStateChanged(string ident, string client, int state)
{
    //whenever device stat changes, this callback will report the client and new state
}

public void communicationCallback(bool incoming, string json)
{
    //This callback will report all the data going to/from the devices. Used for debugging/extended logging
}
```

You set the callback usually upon form/project initialization by utilizing the RemoteAPI class

```
public Form1()
{
    InitializeComponent();
```

```

RemoteAPI.setDeviceListChangedCallback(deviceListChanged);

RemoteAPI.setClientListChangedCallback(clientListChanged);

RemoteAPI.setDeviceStateChangedCallback(deviceStateChanged);

RemoteAPI.setCommunicationCallback(communicationCallback);
}

```

To start the ZMQ server, you utilize the following API from the RemoteAPI class:

```

public bool ZMQServer(bool enable, string identifier = "ZMQ_SERVER", string address = "127.0.0.1", int port
    = 13599);

```

- enable: True = turn ON server, False = Stop Server
- identifier: The name to call the server instance
- address: The IP address to bind the listening socket to
- port: The port to bind the listening socket to

The RemoteAPI class has a singleton instance called SharedController.

Example to start and stop the ZMQ Server:

```

//Start Server
RemoteAPI.SharedController.ZMQServer(true); //spins up server using name "ZMQ_SERVER", address 127.0.0.1
and port 13599

//Stop Server
RemoteAPI.SharedController.ZMQServer(false);

```

2.2 Setting Up ZMQ Client App

The ZMQ Client SDK device API are in the IDT_Device class which mirrors the same methods/signatures as the IDT_Device class in our main IDTechSDK_STD SDK. This means the implementation is identical to using IDTechSDK_STD SDK (when using IDT_Device class and not individual device classes like IDT_VP330 and IDT_NEO2), except for the device connection.

When using IDTechSDK_STD, you enable USB and devices automatically appear. When using ZMQ Client SDK, you must first connect to the ZMQ Server app, and once that is established, any devices connected to ZMQ Server app will be passed to the ZMQ Client app as if they are locally attached.

You establish your application as you would if using the regular IDTechSDK_STD, but you must only send device commands using IDT_Device class.

Establish/test your application for local device access using IDTechSDK_STD. Once you are happy with the operation and ready to convert it to a ZMQ Client app, remove the link to IDTechSDK_STD, and instead import from NuGet IDTechZMQ_Client.

There should be no error in your app after importing the client SDK. If there are any, it probably is because your app using individual device class calls (IDT_VP330.SharedController.device_getFirmwareVersion(ref ver, ident)), instead of the IDT_Device version (IDT_Device.SharedController.device_getFirmwareVersion(ref ver, ident)). Please update any of your existing methods if this is the case.

To connect/disconnect your client to the server app, you use the following API call

```

//IDT_Device class
public static bool client_remoteCommandClient(bool enable, string identifier = "", string address = "
    127.0.0.1", int port = 13599)

```

- enable: True = turn ON client, False = Stop Client
- identifier: The name to call your client instance

- address: The IP address where the server is running at
- port: The port where the server is running at

Example to start and stop the ZMQ Client:

```
//Start Client
IDT_Device.client_remoteCommandClient(true,"Store1"); //spins up client using name "Store", and attempts
to connect to server at address 127.0.0.1 and port 13559

//Stop Client
IDT_Device.client_remoteCommandClient(false);
```

Get/Set Wait Time

This function will allow you to define how long the client is willing to wait for a command to execute when the server reports the device is busy from a request from another client.

```
//IDT_Device Class

public static void client_setWaitTime(int time)
public static int client_getWaitTime()

//NOTE: time is in milliseconds
```

2.3 ZMQ Server / Client Example

The following steps will allow you to execute a ZMQ Server/Client environment using sample code provided by IDTech. This requires you to open IDTech uDemo source code in Visual Studio and adjust the imported SDK.

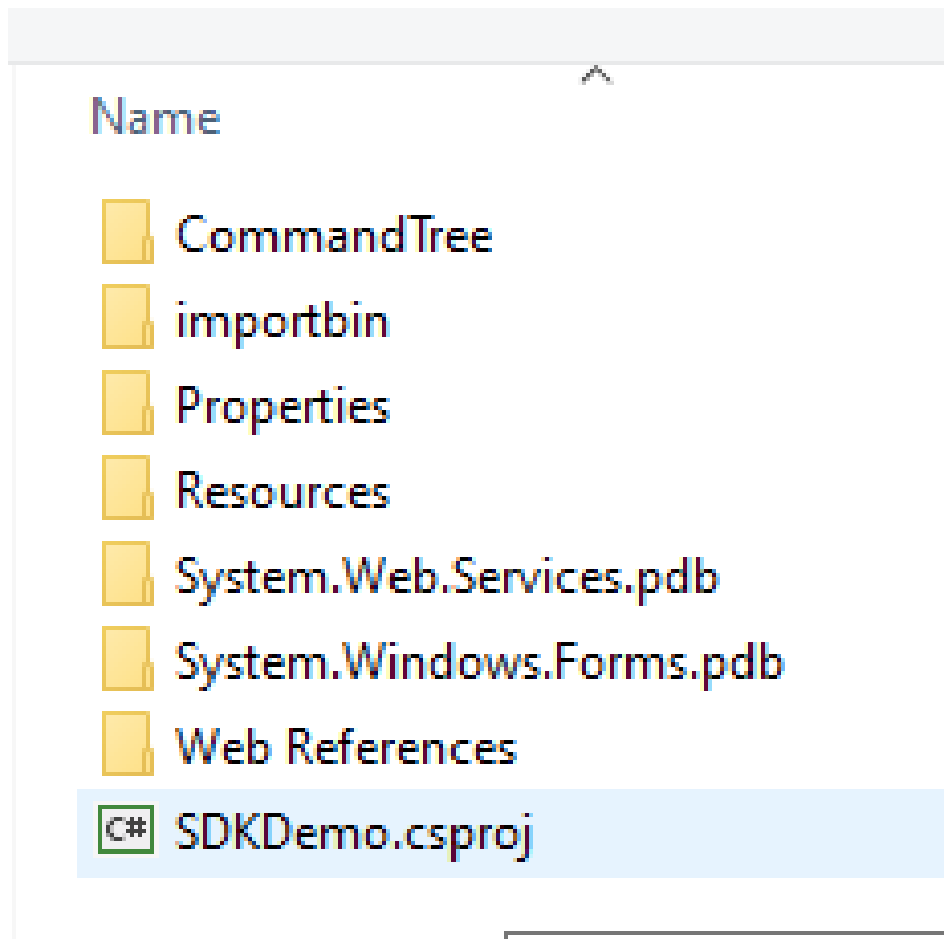
Please download and unzip the ZMQ Server App

https://atlassian.idtechproducts.com/confluence/download/attachments/30479625/IDTech_ZMQ_Server_App_Source_Code.zip?api=v2

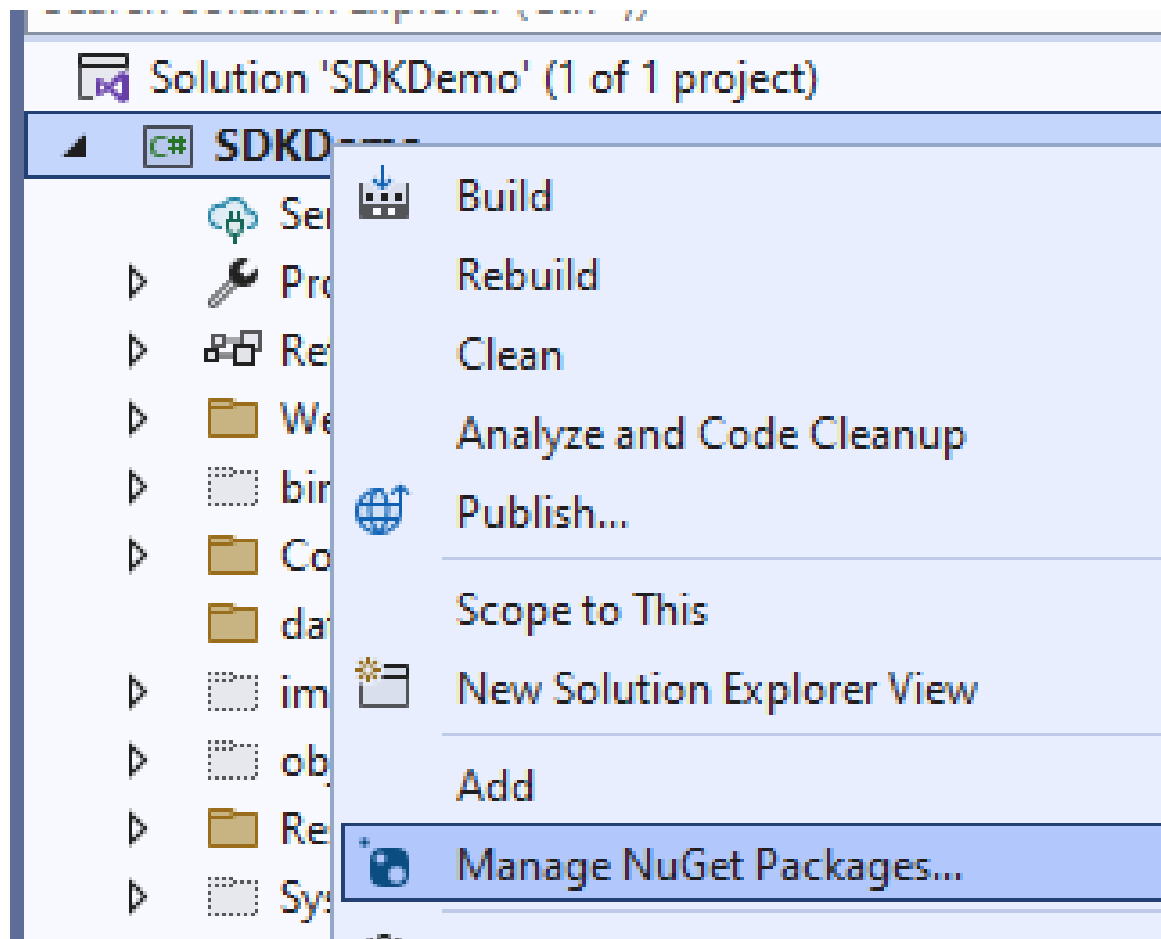
Please download and unzip IDTech uDemo source code

<https://atlassian.idtechproducts.com/confluence/download/attachments/30479625/dot%20NET%20SDK%20Demo%20Source%20Code.zip?api=v2>

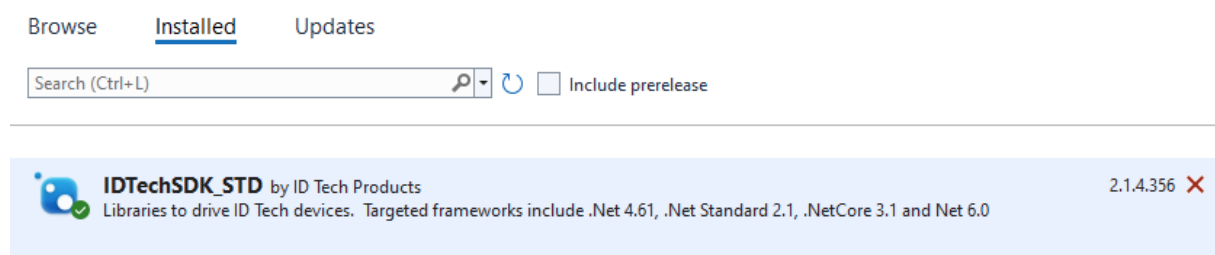
In Visual Studio, open uDemo with file SDKDemo.csproj



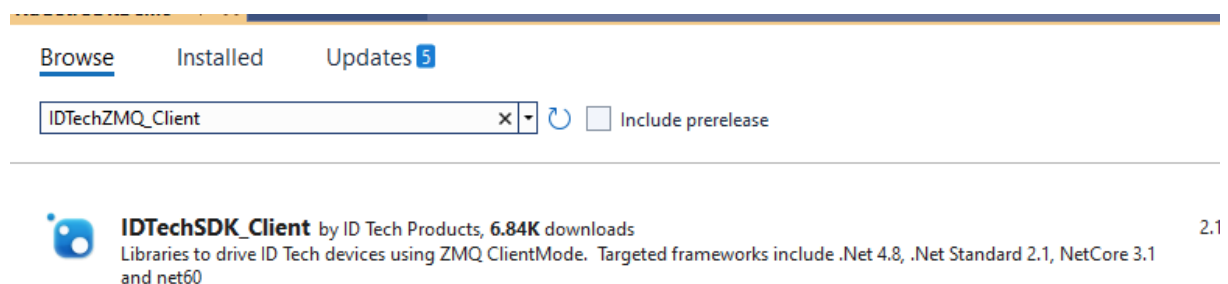
Right-click on SDKDemo project, and Manage Nuget Packages...



Under the Installed tab, click the red X to REMOVE IDTechSDK_STD

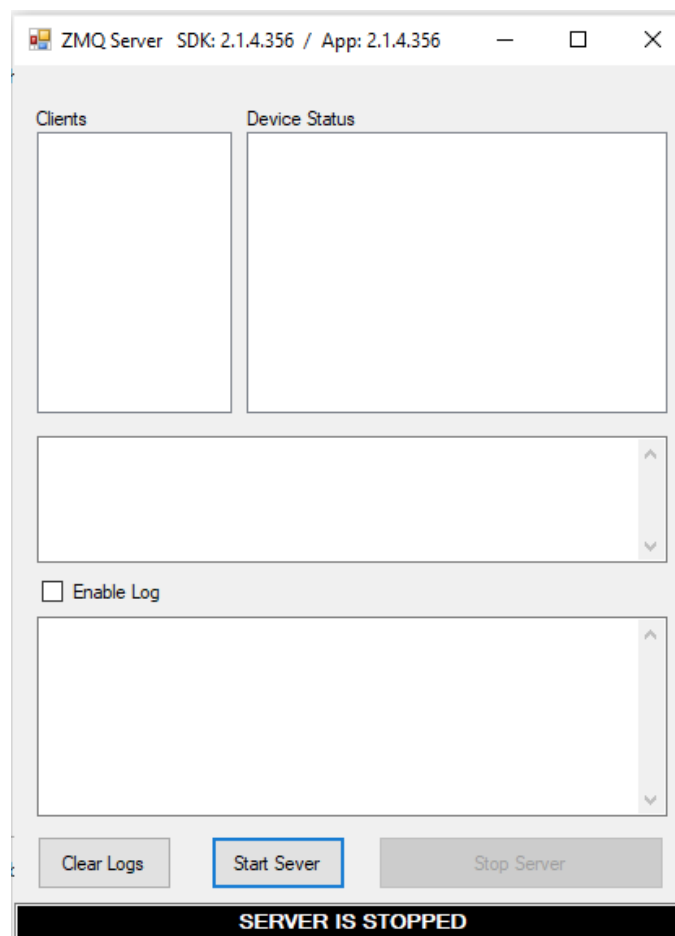
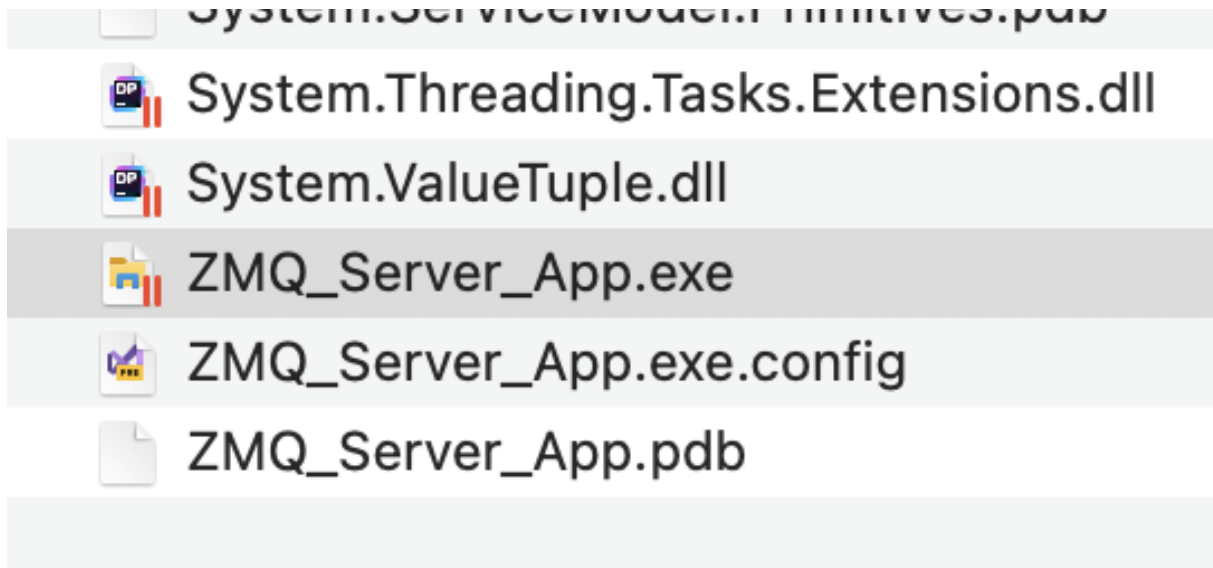


Under the Browse tab, Browse and install IDTechSDK_Client

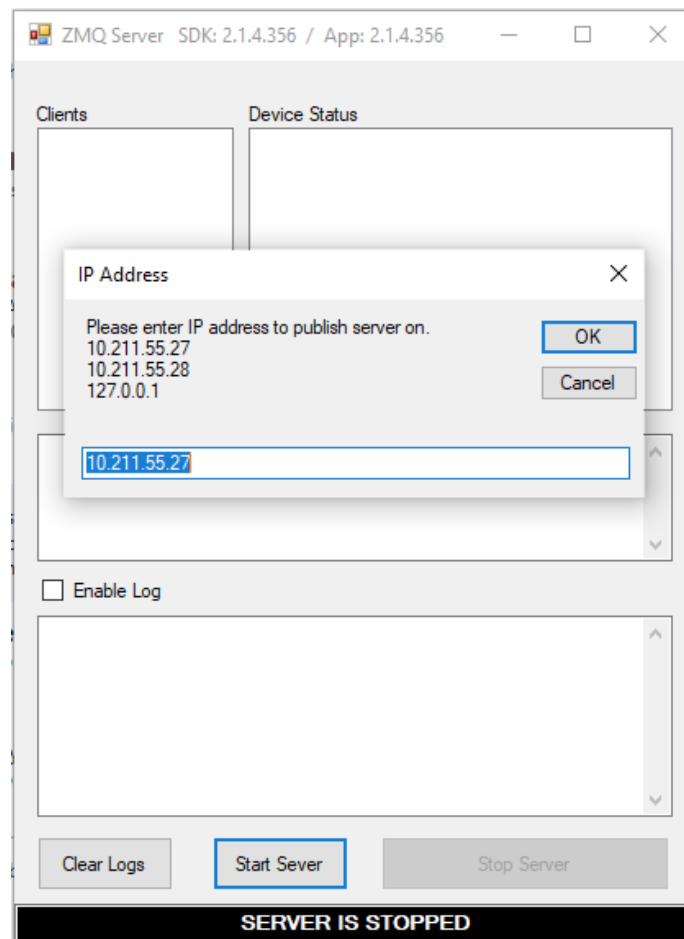


Plug a device into your PC, example VP3350

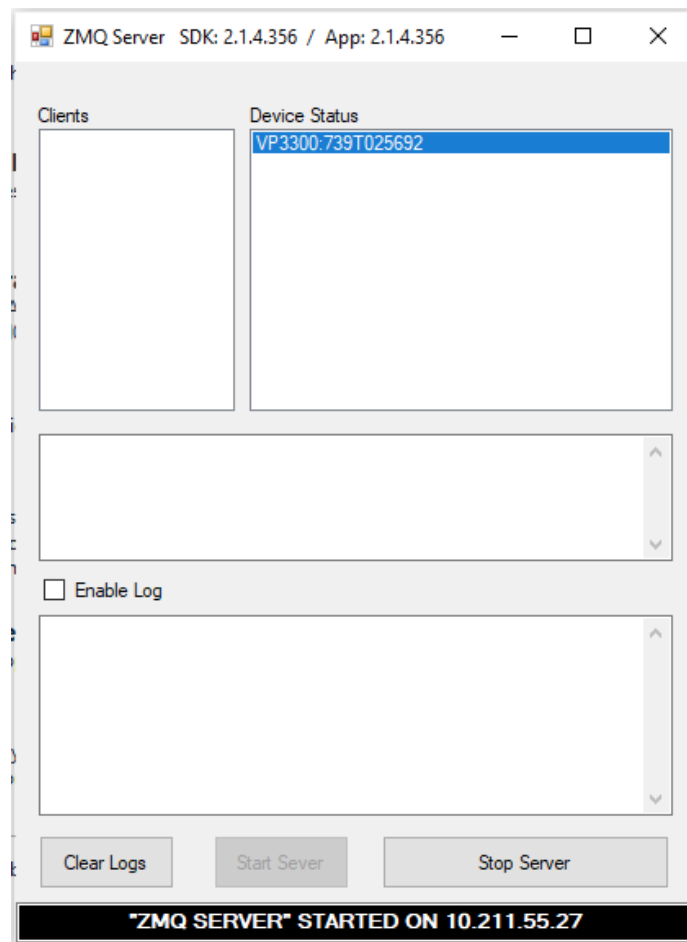
In the ZMQ Sever App bin/Release folder, run the server app ZMQ_Server_App.exe



Click "Start Server" and enter one of the valid IP address the server NIC has access to. In our example, we will use 10.211.55.27



You should see the server successfully start, and the VP3300 as connected device



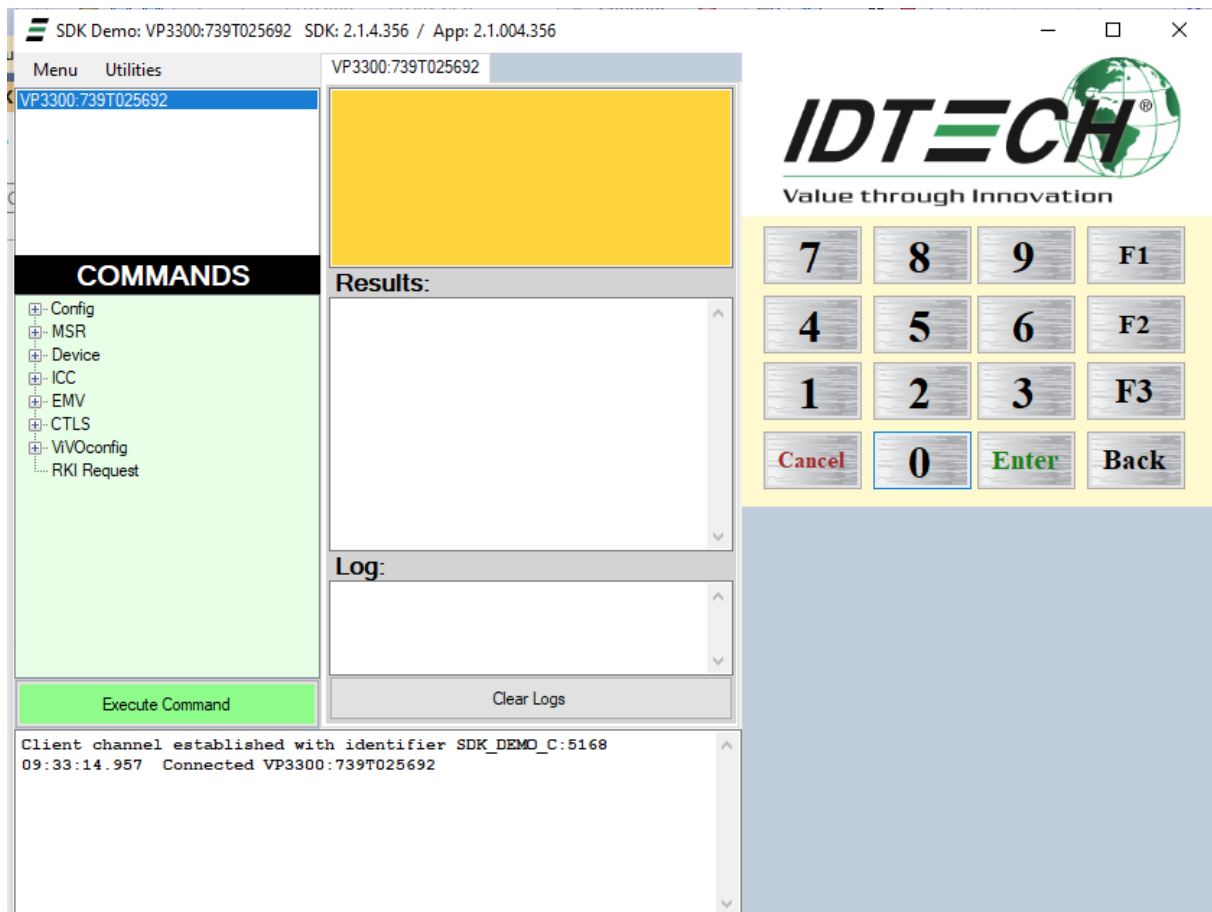
Back in Visual Studio, run uDemo. Select Menu->Enable Client.

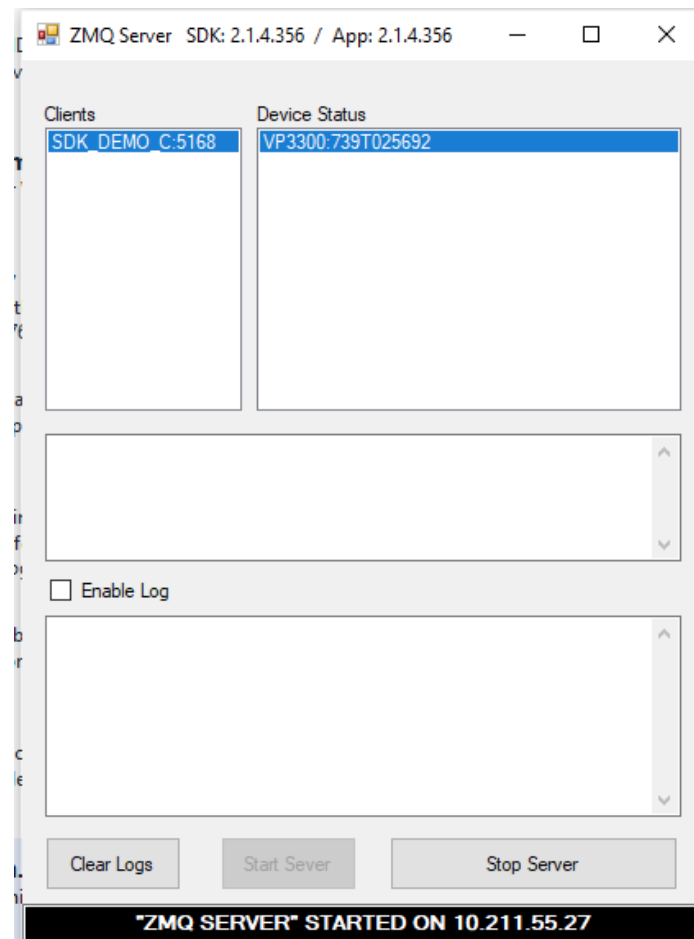
You will be asked to provide a name of your client. Example "SDK_DEMO".

You will be asked the address of the server. In our example, we are using 10.211.55.27

You will now see uDemo connect the client, and you will see the server app serving the client.

At this point, you can now send commands to the VP3300.





Chapter 3

Using SDK on iOS/Android with MAUI

The IDTech SDK can be use on iOS and Android platform when coding with MAUI in Visual Studio.

Android/iOS sample code can be found here:

https://atlassian.idtechproducts.com/confluence/download/attachments/30479625/MAUI_SDKuDemo.zip?api=v2

Once you have established your MAUI project, right-click on your project and select "Manage NuGet Packages..." import IDTech.Maui.Comm from NuGet. Note, this will automatically import the other libraries, including the iOS and Android binding libraries, and IDTechSDK.dll. There is no need to separately import IDTechSDK_STD from NuGet

NOTE: You need to save your project as a solution (.sln) before you will be able to manage NuGet packages.

3.1 Setting Up iOS Platform Bindings

In the Main routine in Program.cs in the Platforms/iOS, execute IDTech.Maui.Comm.IDTechBinding.Init(); BEFORE the app delegate.

```
public class Program
{
    // This is the main entry point of the application.
    static void Main(string[] args)
    {
        IDTech.Maui.Comm.IDTechBinding.Init();
        // if you want to use a different Application Delegate class from "AppDelegate"
        // you can specify it here.
        UIApplication.Main(args, null, typeof(AppDelegate));
    }
}
```

Add permissions to your info.plist

If you are planning on using the lightning connector and External Accessory Protocol, include the protocol string for the product(s) you would like to use. NOTE: Your app will need approval to use any listed devices before publishing to App Store, so only include devices you plan to use. If you don't plan on using any of the Lightning Connector devices, exclude this key from your .plist

```
VP3350 = com.idtechproducts.neo
BTMag = com.idtechproducts.BTMag
BTPay = com.idtechproducts.BTPay 200
iMag = com.idtechproducts.reader

<key>UISupportedExternalAccessoryProtocols</key>
<array>
    <string>com.idtechproducts.neo</string>
    <string>com.idtechproducts.reader</string>
    <string>com.idtechproducts.BTPay 200</string>
    <string>com.idtechproducts.BTMag</string>
</array>
```

If you are planning on using BLE devices, add the following:


```
<key>NSBluetoothAlwaysUsageDescription</key>
<string>Captures Payment Information</string>
<key>NSBluetoothPeripheralUsageDescription</key>
<string>Captures Payment Information</string>
```

If you are planning on using Audio Jack devices, add the following:

```
<key>NSMicrophoneUsageDescription</key>
<string>Captures Payment Information</string>
```

3.2 Setting Up Android Platform Bindings

In the MainActivity onCreate routine for Android, execute IDTech.Maui.Comm.IDTechBinding.Init(this) BEFORE base.OnCreate.

In the MainActivity onCreate routine for Android, set this.RequestPermissions AFTER LoadApplication.

After the RequestPermission are established, enable the USB drivers in IDTechBinding by executing IDTech.Maui.Comm.IDTechBinding.enableUSB();

```
protected override void OnCreate(Bundle savedInstanceState)
{
    IDTech.Maui.Comm.IDTechBinding.Init(this);

    SharedController = this;

    base.OnCreate(savedInstanceState);

    Platform.Init(this, savedInstanceState);
    this.RequestPermissions(new[]
    {
        Manifest.Permission.AccessCoarseLocation,
        Manifest.Permission.AccessFineLocation,
        Manifest.Permission.AccessNetworkState,
        Manifest.Permission.BluetoothPrivileged,
        Manifest.Permission.Bluetooth,
        Manifest.Permission.BluetoothAdmin,
        Manifest.Permission.ModifyAudioSettings,
        Manifest.Permission.RecordAudio
    }, 0);
    IDTech.Maui.Comm.IDTechBinding.enableUSB();
}
```

In your manifest of your Android Project, set the required permissions. You can set the manifest by editing Android Project Options ->Android Application->Required Permissions and select the following

```
- AccessCoarseLocation,
- AccessFineLocation,
- AccessNetworkState,
- BluetoothPrivileged,
- Bluetooth,
- BluetoothAdmin,
- ModifyAudioSettings,
- RecordAudio
```

3.3 Setting Up Main (Shared) Platform Bindings

IDTech SDK was imported when you imported the IDTech.Maui.Comm library into the main project.

Other than special considerations for connecting to the devices on the mobile platform (specified below), follow the standard project development procedures by references in the other sections of this help document and also Windows Forms IDTech SDK uDemo source code "dot NET SDK Demo Source Code.zip". Standard development consists of setting the callback, and then executing API commands using a shared controller. The project page with these files can be found at: <https://atlassian.idtechproducts.com/confluence/display/~KB/Universal+Library+for+Visual+Studio+-+Home>

3.4 Communicating with Devices

There are four groups of devices that can be used on Android: Bluetooth Low Energy, Audio Jack, USB, Audio Jack and Serial

There are three groups of devices that can be used on iOS: Bluetooth Low Energy, Lightning Connector, and Audio Jack

Bluetooth Low Energy (Android and iOS).

This includes the VP3300, and NEOII devices, such as VP3310, VP3320, VP3350, and VP3600.

You connect to a BLE device using the following method:

```
IDT_Device.startBLEScan(IDT_DEVICE_Types type, string name);
```

The first parameter is the device type. This can have one of three possible values: IDT_DEVICE_Types.IDT_DEVICE_VP3300 - VP3300, VP8300 IDT_DEVICE_Types.IDT_DEVICE_NEO2 - VP3600, VP3320, VP3350 (VP6800 BLE is currently not supported) IDT_DEVICE_Types.IDT_DEVICE_NONE - scan for name when no name provided

The second parameter is the device name that you would like to connect to.

If you don't specify a name (and set device type to IDT_DEVICE_Types.IDT_DEVICE_NONE), the SDK will scan the area and return a list of all found names. You can then use one of those names to attempt to connect to that device. When the SDK finishes scanning the area (2-3 seconds), it will return a callback DeviceState.BTListReady. You can then execute IDT_Device.getBLEDeviceList() to retrieve the list of device names found in the vicinity.

```
case DeviceState.BTListReady:
    List<string> btDevices = null;

    btDevices = IDT_Device.getBLEDeviceList();
    displayBLEDevices(btDevices);
    break;
```

USB (Android Only).

USB device detection is automatic. The SDK callback will inform when devices are attached or detached from the system. No special considerations or coding needed

Serial (Android Only).

Using a Serial to USB adapter, you can attempt to connect to a serial device with the following method:

```
IDT_Device.useSerialAdapter(IDT_DEVICE_Types type, int baud);
```

The first parameter is the type of device you would like to connect to. If you would like to connect to a MiniSmartII, you would pass IDT_DEVICE_Types.IDT_DEVICE_MINISMARTII

The second parameter is the baud rate. If you pass a value of 0, it will use the factory default baud rate. Otherwise, if the baud rate was changed, you specify it here (example 9600)

Audio Jack (Android/iOS).

For Audio Jack, you first need to tell the SDK which device it need to configure the audio drivers for, and then monitor the Headphone Jack plug/unplug events, and when a plug event happens, send a command to connect to the audio jack reader.

Here are the available devices to configure the SDK for the correct model audio jack product

```
IDT_Device.startAudioJack(IDT_DEVICE_Types.IDT_DEVICE_VP3300);
IDT_Device.startAudioJack(IDT_DEVICE_Types.IDT_DEVICE_UNIPAYI_V);
IDT_Device.startAudioJack(IDT_DEVICE_Types.IDT_DEVICE_UNIPAY);
IDT_Device.startAudioJack(IDT_DEVICE_Types.IDT_DEVICE_UNIMAG);
IDT_Device.startAudioJack(IDT_DEVICE_Types.IDT_DEVICE_UNIMAG_PRO);
```

Then when a plug insertion event is detected, you power up the device:

```
case DeviceState.PlugInserted:
    appendMessageToResults("Plug Inserted");
    IDT_Device.connectToAudioReader();
    //When using audio jack devices, notification of a device insertion will appear here
    break;
```

External Accessory (iOS Only).

External Accessory devices are automatically detected

Additional Callbacks.

There are some additional DeviceState callback event on applicable to Audio Jack connections.

There is a DeviceMessage callback. These are device message returned from various audio jack products. Messages such as "POWERING UNIPAY" and "PLEASE SWIPE"

```
case DeviceState.DeviceMessage:
    //When iOS SDK needs to send out SDK messages, it will appear here
    if (data != null)
    {
        string message = Encoding.UTF8.GetString(data, 0, data.Length);
        appendMessageToResults(message);
    }
```

The UniPay may have ICC messages

```
case DeviceState.UniPayICC:
    appendMessageToResults("UniPay ICC Event: " + data[0].ToString("x2"));
    //When using UniPay, device notifications (card inserted, card removed) will appear here
    break;
```

Chapter 4

Using SDK on Android with Xamarin

The IDTech SDK can be use on Android platform when coding with Xamarin in Visual Studio.

Android/iOS sample code can be found here:

https://atlassian.idtechproducts.com/confluence/download/attachments/30479625/Xamarin_Demo_Source_Code.zip?api=v2

Once you have established your Xamarin project (either Universal or Android only), right-click on your project and select "Manage NuGet Packages..."

Browse for "Xamarin.IDTech.AndroidBinding". Once located, install latest version for your project. This library has a dependency for our IDTechSDK_STD and our comm layer library Xamarin.IDTech.AndroidCommLib. Those additional two libraries will automatically be installed along with the Xamarin.IDTech.AndroidBinding library.

In the MainActivity onCreate routine for Android, execute Xamarin.IDTech.AndroidBinding.IDTechBinding.Init(this) BEFORE base.OnCreate.

In the MainActivity onCreate routine for Android, set this.RequestPermissions AFTER LoadApplication.

After the RequestPermission are established, enable the USB drivers in IDTechBinding by executing Xamarin.IDTech.AndroidBinding.IDTechBinding.enableUSB();

```
protected override void OnCreate(Bundle savedInstanceState)
{
    TabLayoutResource = Resource.Layout.Tabbar;
    ToolbarResource = Resource.Layout.Toolbar;

    //IDTechSDK Binding
    Xamarin.IDTech.AndroidBinding.IDTechBinding.Init(this);

    base.OnCreate(savedInstanceState);

    Xamarin.Essentials.Platform.Init(this, savedInstanceState);
    global::Xamarin.Forms.Forms.Init(this, savedInstanceState);
    LoadApplication(new App());

    //Required Permissions for IDTechSDK
    this.RequestPermissions(new[]
    {
        Manifest.Permission.AccessCoarseLocation,
        Manifest.Permission.AccessFineLocation,
        Manifest.Permission.AccessNetworkState,
        Manifest.Permission.BluetoothPrivileged,
        Manifest.Permission.Bluetooth,
        Manifest.Permission.BluetoothAdmin,
        Manifest.Permission.ModifyAudioSettings,
        Manifest.Permission.RecordAudio
    }, 0);
    Xamarin.IDTech.AndroidBinding.IDTechBinding.enableUSB();
}
```

In your manifest of your Android Project, set the required permissions. You can set the manifest by editing Android Project Options ->Android Application->Required Permissions and select the following:

```
- AccessCoarseLocation,
```

```
- AccessFineLocation,
- AccessNetworkState,
- BluetoothPrivileged,
- Bluetooth,
- BluetoothAdmin,
- ModifyAudioSettings,
- RecordAudio
```

In the Shared (Main) project import IDTechSDK_STD from NuGet

USB device detection is automatic. The SDK callback will inform when devices are attached or detached from the system. No special considerations or coding needed

Using a Serial to USB adapter, you can attempt to connect to a serial device with the following method:

```
IDT_Device.useSerialAdapter(IDT_DEVICE_Types type, int baud);
```

The first parameter is the type of device you would like to connect to. If you would like to connect to a MiniSmartII, you would pass IDT_DEVICE_Types.IDT_DEVICE_MINISMAII

The second parameter is the baud rate. If you pass a value of 0, it will use the factory default baud rate. Otherwise, if the baud rate was changed, you specify it here (example 9600)

4.1 Communicating with Android Devices

There are four groups of devices that can be used on Android: Bluetooth Low Energy, Audio Jack, USB, and Serial

Bluetooth Low Energy.

This includes the VP3300, and NEOII devices, such as VP3310, VP3320, VP3350, and VP3600.

You connect to a BLE device using the following method:

```
IDT_Device.startBLEScan(IDT_DEVICE_Types type, string name);
```

The first parameter is the device type. This can have one of three possible values: IDT_DEVICE_Types.IDT_DEVICE_VP3300 - VP3300, VP8300 IDT_DEVICE_Types.IDT_DEVICE_NEO2 - VP3600, VP3320, VP3350 (VP6800 BLE is currently not supported) IDT_DEVICE_Types.IDT_DEVICE_NONE - scan for name when no name provided

The second parameter is the device name that you would like to connect to.

If you don't specify a name (and set device type to IDT_DEVICE_Types.IDT_DEVICE_NONE), the SDK will scan the area and return a list of all found names. You can then use one of those names to attempt to connect to that device. When the SDK finishes scanning the area (2-3 seconds), it will return a callback DeviceState.BTListReady. You can then execute IDT_Device.getBLEDDeviceList() to retrieve the list of device names found in the vicinity.

```
case DeviceState.BTListReady:
    List<string> btDevices = null;

    btDevices = IDT_Device.getBLEDDeviceList();
    displayBLEDevices(btDevices);
    break;
```

USB.

USB device detection is automatic. The SDK callback will inform when devices are attached or detached from the system. No special considerations or coding needed

Serial.

Using a Serial to USB adapter, you can attempt to connect to a serial device with the following method: :

```
IDT_Device.useSerialAdapter(IDT_DEVICE_Types type, int baud);
```

The first parameter is the type of device you would like to connect to. If you would like to connect to a MiniSmartII, you would pass `IDT_DEVICE_Types.IDT_DEVICE_MINISMARTII`

The second parameter is the baud rate. If you pass a value of 0, it will use the factory default baud rate. Otherwise, if the baud rate was changed, you specify it here (example 9600)

Audio Jack.

For Audio Jack, you first need to tell the SDK which device it need to configure the audio drivers for, and then monitor the Headphone Jack plug/unplug events, and when a plug event happens, send a command to connect to the audio jack reader.

Here are the available devices to configure the SDK for the correct model audio jack product

```
IDT_Device.startAudioJack(IDT_DEVICE_Types.IDT_DEVICE_VP3300);  
IDT_Device.startAudioJack(IDT_DEVICE_Types.IDT_DEVICE_UNIPAYI_V);  
IDT_Device.startAudioJack(IDT_DEVICE_Types.IDT_DEVICE_UNIPAY);  
IDT_Device.startAudioJack(IDT_DEVICE_Types.IDT_DEVICE_UNIMAG);  
IDT_Device.startAudioJack(IDT_DEVICE_Types.IDT_DEVICE_UNIMAG_PRO);
```

Then when a plug insertion event is detected, you power up the device:

```
case DeviceState.PlugInserted:  
    appendMessageToResults("Plug Inserted");  
    IDT_Device.connectToAudioReader();  
    //When using audio jack devices, notification of a device insertion will appear here  
    break;
```

Chapter 5

Using SDK on iOS with Xamarin

The IDTech SDK can be use on iOS platform when coding with Xamarin in Visual Studio.

Android/iOS sample code can be found here:

https://atlassian.idtechproducts.com/confluence/download/attachments/30479625/Xamarin_Demo_Source_Code.zip?api=v2

Once you have established your Xamarin project (either Universal or iOS only), right-click on your project and select "Manage NuGet Packages..."

Browse for "Xamarin.IDTech.iOSBinding". Once located, install latest version for your project. This library has a dependency for our IDTechSDK_STD and our comm layer library Xamarin.IDTech.iOSCommLib. Those additional two libraries will automatically be installed along with the Xamarin.IDTech.iOSBinding library.

In your iOS project, you will have a Info.plist file. This file must be modified accoding to what devices you are planning on using.

If you are planning on using VP3300, VP3310, VP3320, VP330, or any device that uses BLE, you must enter two key entries in your .plist. The keys are "NSBluetoothAlwaysUsageDescription" and "NSBluetoothPeripheralUsageDescription", and must contant a string, such as "Captures Payment Information".

```
<key>NSBluetoothAlwaysUsageDescription</key>
<string>Captures Payment Information</string>
<key>NSBluetoothPeripheralUsageDescription</key>
<string>Captures Payment Information</string>
```

If you are planning on using any IDTech device that interfaces with the Microphone, you must enter a key entry in your .plist. Thy key is "NSMicrophoneUsageDescription" and must contain a string, , such as "Captures Payment Information".

```
<key>NSMicrophoneUsageDescription</key>
<string>Captures Payment Information</string>
```

If you are planning on using the BTMag and/or iMag, you must add the key "UISupportedExternalAccessoryProtocols", and list the associated protocol strings for those devices in that key's array.

```
<key>UISupportedExternalAccessoryProtocols</key>
<array>
  <string>com.idtechproducts.reader</string>
  <string>com.idtechproducts.BTMag</string>
</array>
```

In your application code. add references to the IDTech libraries you imported using NuGet

```
using IDTechSDK;
using iOSBinding;
```

Then set up the rest of your code as if it is the standard Window .Net project using our IDTechSDK_STD. You define the message callback and make command calls to the [IDTechSDK](#)

5.1 Communicating with iOS Devices

There are three groups of devices that can be used on iOS: Bluetooth Low Energy, Audio Jack, and External Accessory

Bluetooth Low Energy.

This includes the VP3300, and NEOII devices, such as VP3310, VP3320, VP3350, and VP3600.

You connect to a BLE device using the following method:

```
IDT_Device.startBLEScan(IDT_DEVICE_Types type, string name);
```

The first parameter is the device type. This can have one of three possible values: IDT_DEVICE_Types.IDT_DEVICE_VP3300 - VP3300, VP8300 IDT_DEVICE_Types.IDT_DEVICE_NEO2 - VP3600, VP3320, VP3350 (VP6800 BLE is currently not supported) IDT_DEVICE_Types.IDT_DEVICE_NONE - scan for name when no name provided

The second parameter is the device name that you would like to connect to.

If you don't specify a name (and set device type to IDT_DEVICE_Types.IDT_DEVICE_NONE), the SDK will scan the area and return a list of all found names. You can then use one of those names to attempt to connect to that device. When the SDK finishes scanning the area (2-3 seconds), it will return a callback DeviceState.BTListReady. You can then execute IDT_Device.getBLEDeviceList() to retrieve the list of device names found in the vicinity.

```
case DeviceState.BTListReady:
    List<string> btDevices = null;

    btDevices = IDT_Device.getBLEDeviceList();
    displayBLEDevices(btDevices);
    break;
```

Audio Jack.

For Audio Jack, you first need to tell the SDK which device it need to configure the audio drivers for, and then monitor the Headphone Jack plug/unplug events, and when a plug event happens, send a command to connect to the audio jack reader.

Here are the available devices to configure the SDK for the correct model audio jack product

```
IDT_Device.startAudioJack(IDT_DEVICE_Types.IDT_DEVICE_VP3300);
IDT_Device.startAudioJack(IDT_DEVICE_Types.IDT_DEVICE_UNIPAYI_V);
IDT_Device.startAudioJack(IDT_DEVICE_Types.IDT_DEVICE_UNIPAY);
IDT_Device.startAudioJack(IDT_DEVICE_Types.IDT_DEVICE_UNIMAG);
IDT_Device.startAudioJack(IDT_DEVICE_Types.IDT_DEVICE_UNIMAG_PRO);
```

Then when a plug insertion event is detected, you power up the device:

```
case DeviceState.PlugInserted:
    appendMessageToResults("Plug Inserted");
    IDT_Device.connectToAudioReader();
    //When using audio jack devices, notification of a device insertion will appear here
    break;
```

External Accessory.

This includes the BTMag and iMag. For External Accessory, you just need to link the binding library to the SDK with a single call (depending on device):

```
//BTMag:
IDTechBinding.SharedController.setDeviceType(IDT_DEVICE_Types.IDT_DEVICE_BTMAg, DEVICE_INTERFACE_Types.DEVICE_INTERFACE_SERIAL);

//iMag:
IDTechBinding.SharedController.setDeviceType(IDT_DEVICE_Types.IDT_DEVICE_IMAG, DEVICE_INTERFACE_Types.DEVICE_INTERFACE_SERIAL);
```


Additional Callbacks.

There are some additional DeviceState callback event on applicable to iOS.

There is a DeviceMessage callback. These are device message returned from various audio jack products. Messages such as "POWERING UNIPAY" and "PLEASE SWIPE"

```
case DeviceState.DeviceMessage:
    //When iOS SDK needs to send out SDK messages, it will appear here
    if (data != null)
    {
        string message = Encoding.UTF8.GetString(data, 0, data.Length);
        appendMessageToResults(message);
    }
```

The UniPay may have ICC messages

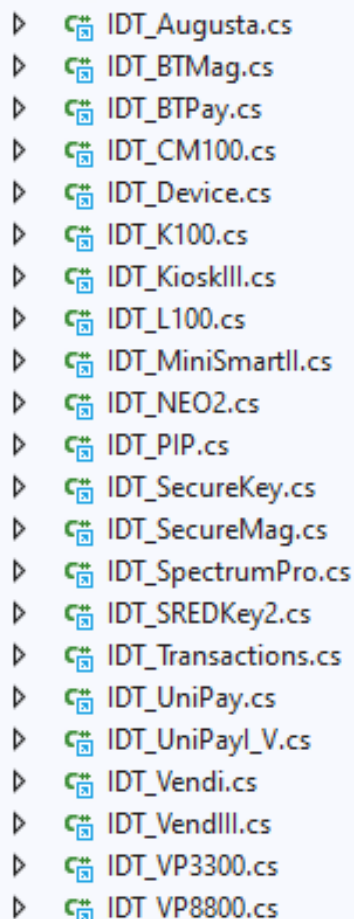
```
case DeviceState.UniPayICC:
    appendMessageToResults("UniPay ICC Event: " + data[0].ToString("x2"));
    //When using UniPay, device notifications (card inserted, card removed) will appear here
    break;
```

Chapter 6

SDK Class Organization

There are two strategies for establishing instances to the classes. There is a single-device project, and there could be a multi-device project.

A single device project is meant for an application that will only target a single IDTech device. A multi-device project would be expected to switch between different IDTech devices.



A screenshot of a file explorer window showing a list of 20 C# class files. Each file name is preceded by a green folder icon with a plus sign and a blue square icon. The files are listed as follows:

- IDT_Augusta.cs
- IDT_BTMag.cs
- IDT_BTPay.cs
- IDT_CM100.cs
- IDT_Device.cs
- IDT_K100.cs
- IDT_KioskIII.cs
- IDT_L100.cs
- IDT_MiniSmartII.cs
- IDT_NEO2.cs
- IDT_PIP.cs
- IDT_SecureKey.cs
- IDT_SecureMag.cs
- IDT_SpectrumPro.cs
- IDT_SREDKey2.cs
- IDT_Transactions.cs
- IDT_UniPay.cs
- IDT_UniPayI_V.cs
- IDT_Vendi.cs
- IDT_VendIII.cs
- IDT_VP3300.cs
- IDT_VP8800.cs

The SDK is made up of a class for each device. In addition, there is the main class file "IDT_Device". The "IDT_Device" class is the main class for executing all API calls on all devices. The individual device class files ("IDT_VP3300", "IDT_MiniSmartII") contain the API calls that are meant just for that device. The individual device class files call the API methods in IDT_Device class. Think of the individual device class files as a filter of what can be executed in the IDT_Device class.

The pros of using an individual device class for executing all commands:

1. It defines the target device automatically
2. It filters out commands only applicable to that device The cons of using an individual device class:

1. You cannot use any other devices with your application

The pros of using IDT_Device class for executing all commands:

1. You can switch between different devices and still use the same API command The cons of using IDT_Device class for executing all commands:
1. You must explicitly tell the SDK which device you want to talk to first
2. There are many commands in IDT_Device not applicable to all devices, causing potential confusion.

Examples:

Getting Firmware Version from VP3300 using it's individual device class

```
string firmwareVersion = "";
RETURN_CODE rt = IDT_VP3300.SharedController.device_getFirmwareVersion(ref firmwareVersion);
```

Getting Firmware Version from VP3300 and UniPay 1.5 using IDT_Device device class

```
string firmwareVersion = "";
IDT_Device.setDeviceType(IDT_DEVICE_Types.IDT_DEVICE_VP3300, DEVICE_INTERFACE_Types.DEVICE_INTERFACE_USB);
RETURN_CODE rt = IDT_Device.SharedController.device_getFirmwareVersion(ref firmwareVersion);
IDT_Device.setDeviceType(IDT_DEVICE_Types.IDT_DEVICE_UNIPAYI_V, DEVICE_INTERFACE_Types.DEVICE_INTERFACE_USB);
rt = IDT_Device.SharedController.device_getFirmwareVersion(ref firmwareVersion);
```

When you execute "setDeviceType", you must pass, as a minimum, the Device Type, and also the Interface.

Device Types:

```
public enum IDT_DEVICE_Types
{
    IDT_DEVICE_NONE = -1,
    IDT_DEVICE_BTTPAY = 0,
    IDT_DEVICE_BTMAP = 1,
    IDT_DEVICE_UNIPAY = 2,
    IDT_DEVICE_UNIPAYI_V = 3,
    IDT_DEVICE_UNIMAG = 4,
    IDT_DEVICE_SPECTRUM_PRO = 5,
    IDT_DEVICE_MINISMARTII = 6,
    IDT_DEVICE_AUGUSTA = 7,
    IDT_DEVICE_AUGUSTA_KB = 8,
    IDT_DEVICE_KIOSKIII = 9,
    IDT_DEVICE_CM100 = 10,
    IDT_DEVICE_VENDI = 11,
    IDT_DEVICE_L100 = 12,
    IDT_DEVICE_VENDIII = 13,
    IDT_DEVICE_VP3300 = 14,
    IDT_DEVICE_VP8800 = 15,
    IDT_DEVICE_SECUREMAG = 16,
    IDT_DEVICE_BTTPAY_MINI = 17,
    IDT_DEVICE_K100 = 18,
    IDT_DEVICE_NEO2 = 19,
    IDT_DEVICE_TMS = 20,
    IDT_DEVICE_SECUREKEY = 21,
    IDT_DEVICE_SREDKEY2 = 22,
    IDT_DEVICE_PIP = 23,
    COUNT = 24
}
```

Interfaces (only available interfaces will be USB, Serial, IP):

```
public enum DEVICE_INTERFACE_Types
{
    DEVICE_INTERFACE_UNKNOWN = 0,
    DEVICE_INTERFACE_AUDIO_JACK = 1,    //RFU
}
```

```
    DEVICE_INTERFACE_USB = 2,  
    DEVICE_INTERFACE_BLE = 3,    //RFU  
    DEVICE_INTERFACE_SERIAL = 4,  
    DEVICE_INTERFACE_BT = 5,    //RFU  
    DEVICE_INTERFACE_IP = 6  
}
```

In this document, most all code example will be using IDT_NEO2 class, but be aware that can be substituted for

The API reference lists the commands available for each device. The IDT_Device class is not listed in this API

Chapter 7

Important Security Notice

The Payment Card Industry Payment Application Data Security Standard (PCI PA-DSS) is comprised of fourteen requirements that support the Payment Card Industry Data Security Standard (PCI DSS). The PCI Security Standards Council (PCI SSC), which was founded by the major card brands in June 2005, set these requirements in order to protect cardholder payment information. The standards set by the council are enforced by the payment card companies who established the Council: American Express, Discover Financial Services, JCB International, MasterCard Worldwide, and Visa, Inc.

PCI PA-DSS is an evolution of Visas Payment Application Best Practices (PABP), which was based on the Visa Cardholder Information Security Program (CISP). In addition to Visa CISP, PCI DSS combines American Express Data Security Operating Policy (DSOP), Discover Networks Information Security and Compliance (DISC), and MasterCards Site Data Protection (SDP) into a single comprehensive set of security standards. The transition to PCI PA-DSS was announced in April 2008. In early October 2008, PCI PA-DSS Version 1.2 was released to align with the PCI DSS Version 1.2, which was released on October 1, 2008. On January 1, 2011, PCI PA-DSS Version 2.0 was released. This extends the PCI DSS Version 1.2, which was released on October 1, 2008 and is effective as of January 1, 2011.

7.1 Applicability

The PCI PA-DSS applies to any payment application that stores, processes, or transmits cardholder data as part of authorization or settlement, unless the application would fall under the merchants PCI DSS validation. It is important to note that PA-DSS validated payment applications alone do not guarantee PCI DSS compliance for the merchant. The validated payment application must be implemented in a PCI DSS compliant environment. If your application runs on Windows XP, you are required to turn off Windows XP System Restore Points.

7.2 What Does PA-DSS Mean to You?

The following table provides opening points to cover in any discussion with merchants on data storage.

	Data Element	Storage Permitted	Protection Required	PCI DSS Req. 3, 4
Cardholder Data	Primary Account Number	Yes	Yes	Yes
	Cardholder Name ¹	Yes	Yes ¹	No
	Service Code ¹	Yes	Yes ¹	No
	Expiration Date ¹	Yes	Yes ¹	No
Sensitive Authentication Data ²	Full Magnetic Stripe Data ³	No	N/A	N/A
	CAV2/CID/CVC2/CVV2	No	N/A	N/A
	PIN/PIN Block	No	N/A	N/A

¹ These data elements must be protected if stored in conjunction with the PAN. This protection should be per PCI DSS requirements for general protection of the cardholder environment. Additionally, other legislation (for example, related to consumer personal data protection, privacy, identity theft, or data security) may require specific protection of this data, or proper disclosure of a company's practices if consumer-related personal data is being collected during the course of business. PCI DSS, however, does not apply if PANs are not stored, processed, or transmitted.

² Do not store sensitive authentication data after authorization (even if encrypted).

³ Full track data from the magnetic stripe, magnetic-stripe image on the chip, or elsewhere.

7.3 Third Party Applications

The end-to-end transaction process, beginning with entry into the third party application until the response from the payment engine is returned, must meet the same level of compliance. In order to claim the third party application is end-to-end compliant, the application would need to be submitted to a QSA for a full PA-DSS audit.

The end user and/or P.O.S. developer can integrate and be compliant in the processing portion of a payment transaction. A brief review (given below) of the PA-DSS environmental variables that impact the end user merchant can help the end user merchant obtain and/or maintain PA-DSS compliance. Environmental variables that could prevent passing an audit include without limitation issues involving a secure network connection(s), end user setup location security, users, logging and assigned rights. Remove all testing configurations, samples, and data prior to going into production on your application.

7.4 PA-DSS Guidelines

The following PA-DSS Guidelines are being provided by IDTech as a convenience to its customers. Customers should not rely on these PA-DSS Guidelines, but should instead always refer to the most recent PCI DSS Program Guide published by PCI SSC.

1. Sensitive Data Storage Guidelines

Do not retain full magnetic stripe, card validation code or value (CAV2, CID, CVC2, CVV2), or PIN block data.

1.1 Do not store sensitive authentication data after authorization (even if encrypted): Sensitive authentication data includes the data as cited in the following Requirements 1.1.1 through 1.1.3. PCI Data Security Standard Requirement 3.2

Note: By prohibiting storage of sensitive authentication data after authorization, the assumption is that the transaction has completed the authorization process and the customer has received the final transaction approval. After authorization has completed, this sensitive authentication data cannot be stored.

1.1.1 After authorization, do not store the full contents of any track from the magnetic stripe (located on the back

of a card, contained in a chip, or elsewhere). This data is alternatively called full track, track, track 1, track 2, and magnetic-stripe data.

In the normal course of business, the following data elements from the magnetic stripe may need to be retained:

- The accountholders name,
- Primary account number (PAN),
- Expiration date, and
- Service code
- To minimize risk, store only those data elements needed for business.

Note: See PCI DSS and PA-DSS Glossary of Terms, Abbreviations, and Acronyms for additional information. PCI Data Security Standard Requirement 3.2.1

1.1.2 After authorization, do not store the card-validation value or code (three-digit or four-digit number printed on the front or back of a payment card) used to verify card-not-present transactions. Note: See PCI DSS and PA-DSS Glossary of Terms, Abbreviations, and Acronyms for additional information. PCI Data Security Standard Requirement 3.2.2

1.1.3 After authorization, do not store the personal identification number (PIN) or the encrypted PIN block.

Note: See PCI DSS and PA-DSS Glossary of Terms, Abbreviations, and Acronyms for additional information. PCI Data Security Standard Requirement 3.2.3

1.1.4 Securely delete any magnetic stripe data, card validation values or codes, and PINs or PIN block data stored by previous versions of the payment application, in accordance with industry-accepted standards for secure deletion, as defined, for example by the list of approved products maintained by the National Security Agency, or by other State or National standards or regulations. PCI Data Security Standard Requirement 3.2

Note: This requirement only applies if previous versions of the payment application stored sensitive authentication data.

1.1.5 Securely delete any sensitive authentication data (pre-authorization data) used for debugging or troubleshooting purposes from log files, debugging files, and other data sources received from customers, to ensure that magnetic stripe data, card validation codes or values, and PINs or PIN block data are not stored on software vendor systems. These data sources must be collected in limited amounts and only when necessary to resolve a problem, encrypted while stored, and deleted immediately after use. PCI Data Security Standard Requirement 3.2

2. Protect stored cardholder data

2.1 Software vendor must provide guidance to customers regarding purging of cardholder data after expiration of customer-defined retention period. PCI Data Security Standard Requirement 3.1

2.2 Mask PAN when displayed (the first six and last four digits are the maximum number of digits to be displayed).

Notes:

- This requirement does not apply to those employees and other parties with a legitimate business need to see full PAN;
- This requirement does not supersede stricter requirements in place for displays of cardholder data for example, for point-of-sale (POS) receipts. PCI Data Security Standard Requirement 3.3

2.3 Render PAN, at a minimum, unreadable anywhere it is stored, (including data on portable digital media, backup media, and in logs) by using any of the following approaches:

- One-way hashes based on strong cryptography with associated key management processes and procedures
- Truncation

- Index tokens and pads (pads must be securely stored)
- Strong cryptography with associated key management processes and procedures. The MINIMUM account information that must be rendered unreadable is the PAN. PCI Data Security Standard Requirement 3.4

The PAN must be rendered unreadable anywhere it is stored, even outside the payment application. Note: Strong cryptography is defined in the PCI DSS and PA-DSS Glossary of Terms, Abbreviations, and Acronyms.

2.4 If disk encryption is used (rather than file- or column-level database encryption), logical access must be managed independently of native operating system access control mechanisms (for example, by not using local user account databases). Decryption keys must not be tied to user accounts. PCI Data Security Standard Requirement 3.4.2

2.5 Payment application must protect cryptographic keys used for encryption of cardholder data against disclosure and misuse. PCI Data Security Standard Requirement 3.5

2.6 Payment application must implement key management processes and procedures for cryptographic keys used for encryption of cardholder data. PCI Data Security Standard Requirement 3.6

2.7 Securely delete any cryptographic key material or cryptogram stored by previous versions of the payment application, in accordance with industry-accepted standards for secure deletion, as defined, for example the list of approved products maintained by the National Security Agency, or by other State or National standards or regulations. These are cryptographic keys used to encrypt or verify cardholder data. PCI Data Security Standard Requirement 3.6

Note: This requirement only applies if previous versions of the payment application used cryptographic key materials or cryptograms to encrypt cardholder data.

3. Provide secure authentication features

3.1 The payment application must support and enforce unique user IDs and secure authentication for all administrative access and for all access to cardholder data. Secure authentication must be enforced to all accounts, generated or managed by the application by the completion of installation and for subsequent changes after the "out of the box" installation (defined at PCI DSS Requirements 8.1, 8.2, and 8.5.88.5.15) for all administrative access and for all access to cardholder data. PCI Data Security Standard Requirements 8.1, 8.2, and 8.5.88.5.15

Note: These password controls are not intended to apply to employees who only have access to one card number at a time to facilitate a single transaction. These controls are applicable for access by employees with administrative capabilities, for access to servers with cardholder data, and for access controlled by the payment application. This requirement applies to the payment application and all associated tools used to view or access cardholder data.

3.1.10 If a payment application session has been idle for more than 15 minutes, the application requires the user to re-authenticate. PCI Data Security Standard Requirement 8.5.15.

3.2 Software vendors must provide guidance to customers that all access to PCs, servers, and databases with payment applications must require a unique user ID and secure authentication. PCI Data Security Standard Requirements 8.1 and 8.2

3.3 Render payment application passwords unreadable during transmission and storage, using strong cryptography based on approved standards

Note: Strong cryptography is defined in PCI DSS and PA-DSS Glossary of Terms, Abbreviations, and Acronyms. PCI Data Security Standard Requirement 8.4

4. Log payment application activity

4.1 At the completion of the installation process, the out of the box default installation of the payment application must log all user access (especially users with administrative privileges), and be able to link all activities to individual users. PCI Data Security Standard Requirement 10.1

4.2 Payment application must implement an automated audit trail to track and monitor access. PCI Data Security Standard Requirements 10.2 and 10.3

5. Develop secure payment applications

5.1 Develop all payment applications in accordance with PCI DSS (for example, secure authentication and logging) and based on industry best practices and incorporate information security throughout the software development life cycle. These processes must include the following: PCI Data Security Standard Requirement 6.3

5.1.1 Live PANS are not used for testing or development. PCI Data Security Standard Requirement 6.4.4.

- Validation of all input (to prevent cross-site scripting, injection flaws, malicious file execution, etc.)
- Validation of proper error handling
- Validation of secure cryptographic storage
- Validation of secure communications
- Validation of proper role-based access control (RBAC)

5.1.2 Separate development/test, and production environments

5.1.3 Removal of test data and accounts before production systems become active development. PCI Data Security Standard Requirement 6.4.4

5.1.4 Review of payment application code prior to release to customers after any significant change, to identify any potential coding vulnerability. Removal of custom payment application accounts, user IDs, and passwords before payment applications are released to customers

Note: This requirement for code reviews applies to all payment application components (both internal and public-facing web applications), as part of the system development life cycle required by PA-DSS Requirement 5.1 and PCI DSS Requirement 6.3. Code reviews can be conducted by knowledgeable internal personnel or third parties.

5.2 Develop all web payment applications (internal and external, and including web administrative access to product) based on secure coding guidelines such as the Open Web Application Security Project Guide. Cover prevention of common coding vulnerabilities in software development processes, to include:

- Injection flaws, with particular emphasis on SQL injection, Cross-site scripting (XSS) OS Command Injection, LDAP and Xpath injection flaws, as well as other injection flaws.
- Buffer Overflow.
- Insecure cryptographic storage.
- Insecure communications.
- Improper error handling.
- All HIGH vulnerabilities as identified in the vulnerability identification process at PA-DSS Requirement 7.1.
- Cross-site scripting (XSS)
- Improper access control such as insecure direct object references, failure to restrict URL access and directory traversal.
- Cross-site request forgery (CSRF)

Note: The vulnerabilities listed in PA-DSS Requirements 5.2.1 through 5.2.9 and in PCI DSS at 6.5.1 through 6.5.9 were current in the OWASP guide when PCI DSS v1.2 / PCI DSS v2.0 (01/01/10) were published. However, if and when the OWASP guide is updated, the current version must be used for these requirements.

5.3 Software vendor must follow change control procedures for all product software configuration changes. PCI Data Security Standard Requirement 6.4. 5.The procedures must include the following:

- Documentation of impact
- Management sign-off by appropriate parties
- Testing functionality to verify the new change(s) does not adversely impact the security of the system. Remove all testing configurations, samples, and data before finalizing the product for production.

- Back-out or product de-installation procedures

5.4 The payment application must not use or require use of unnecessary and insecure services and protocols (for example, NetBIOS, file-sharing, Telnet, unencrypted FTP must be secured via SSH, S-FTP, SSL, IPsec and other technology to implement end to end security). PCI Data Security Standard Requirement 2.2.2

6. Protect wireless transmissions

6.1 For payment applications using wireless technology, the wireless technology must be implemented securely. Payment applications using wireless technology must facilitate use of industry best practices (for example, IEEE 802.11i) to implement strong encryption for authentication and transmission. Controls must be in place to protect the implemented wireless network from unknown wireless access points and clients. This includes testing the end users wireless deployment on a quarterly basis to detect unauthorized access points within the system. Change wireless vendor defaults, including but not limited to default wireless encryption keys, passwords, and SSID community strings. Maintain a detailed updated hardware list. The end to end wireless implementation must be end to end secure. The use of WEP as a security control was prohibited as of 30 June 2010. PCI Data Security Standard Requirements 1.2.3, 2.1.1, 4.1.1, 6.2, 11.1a-e and 11.4a-c.

7. Test payment applications to address vulnerabilities

7.1 Software vendors must establish a process to identify newly discovered security vulnerabilities (for example, subscribe to alert services freely available on the Internet) and to test their payment applications for vulnerabilities. Any underlying software or systems that are provided with or required by the payment application (for example, web servers, third-party libraries and programs) must be included in this process. Remove all test configurations, samples, and data after testing and before promoting the changes to production. PCI Data Security Standard Requirement 6.2

7.2 Software vendors must establish a process for timely development and deployment of security patches and upgrades, which includes delivery of updates and patches in a secure manner with a known chain-of-trust, and maintenance of the integrity of patch and update code during delivery and deployment.

8. Facilitate secure network implementation

8.1 The payment application must be able to be implemented into a secure network environment. Application must not interfere with use of devices, applications, or configurations required for PCI DSS compliance (for example, payment application cannot interfere with anti-virus protection, firewall configurations, or any other device, application, or configuration required for PCI DSS compliance). PCI Data Security Standard Requirements 1, 3, 4, 5, and 6.

9. Cardholder data must never be stored on a server connected to the Internet

9.1 The payment application must be developed such that the database server and web server are not required to be on the same server, nor is the database server required to be in the DMZ with the web server. PCI Data Security Standard Requirement 1.3.7

10. Facilitate secure remote software updates

10.1 If payment application updates are delivered securely via remote access into customers systems, software vendors must tell customers to turn on remote-access technologies only when needed for downloads from vendor

and to turn off immediately after download completes. Alternatively, if delivered via VPN or other high-speed connection, software vendors must advise customers to properly configure a firewall or a personal firewall product to secure authentication using a two factor authentication mechanism. PCI Data Security Standard Requirement 8.3

10.2 If payment application may be accessed remotely, remote access to the payment application must be authenticated using a two factor authentication mechanism. PCI Data Security Standard Requirement 8.3

10.3 Any remote access into the payment application must be done securely. If vendors, resellers/integrators, or customers can access customers payment applications remotely, the remote access must be implemented securely. PCI Data Security Standard Requirements 1, 8.3 and 12.3.9

11. Encrypt sensitive traffic over public networks

11.1 If the payment application sends, or facilitates sending, cardholder data over public networks, the payment application must support use of strong cryptography and security protocols such as SSL/TLS and Internet protocol security (IPSEC) to safeguard sensitive cardholder data during transmission over open, public networks. Examples of open, public networks that are in scope of the PCI DSS are: The Internet Wireless technologies Global System for Mobile Communications (GSM) General Packet Radio Service (GPRS) PCI Data Security Standard Requirement 4.1

11.2 The payment application must never send unencrypted PANs by end-user messaging technologies (for example, e-mail, instant messaging, and chat). PCI Data Security Standard Requirement 4.2

12. Encrypt all non-console administrative access

12.1 Instruct customers to encrypt all non-console administrative access using technologies such as SSH, VPN, or SSL/TLS for web-based management and other non-console administrative access. Telnet or remote login must never be used for administrative access. PCI Data Security Standard Requirement 2.3

13. Maintain instructional documentation and training programs for customers, resellers, and integrators

13.1 Develop, maintain, and disseminate a PA-DSS Implementation Guide(s) for customers, resellers, and integrators that accomplishes the following:

- Addresses all requirements in this document wherever the PA-DSS Implementation Guide is referenced.
- Includes a review at least annually and updates to keep the documentation current with all major and minor software changes as well as with changes to the requirements in this document.

13.2 Develop and implement training and communication programs to ensure payment application resellers and integrators know how to implement the payment application and related systems and networks according to the PA-DSS Implementation Guide and in a PCI DSS-compliant manner.

- Update the training materials on an annual basis and whenever new payment application versions are released.

7.5 More Information

IDTech Systems, Inc. highly recommends that merchants contact the card association(s) or their processing company and find out exactly what they mandate and/or recommend. Doing so may help merchants protect themselves from fines and fraud.

For more information related to security, visit:

- <http://www.pcisecuritystandards.org>
- <http://www.visa.com/cisp>
- <http://www.sans.org/resources>
- <http://www.microsoft.com/security/default.asp>
- <https://sdp.mastercardintl.com/>
- <http://www.americanexpress.com/merchantspecs>

CAPN questions: capninfocenter@aexp.com

Chapter 8

Main Transaction Commands

The methods below are provided as a reference to the main commands needed to execute an EMV transaction.

8.1 EMV Methods

Start EMV Transaction

```
IDTechSDK::IDT_NEO2::emv_startTransaction()
```

Begins an amount authorization request with the ICC. Returns authorization decision (approved, denied, or go online) in callback method.

Authenticate EMV Transaction

```
IDTechSDK::IDT_NEO2::emv_authenticateTransaction()
```

By default, auto-authenticate is ON and this step does not need to be performed. If auto-authenticate is OFF ([IDTechSDK::IDT_NEO2::emv_autoAuthenticate](#)), when the results to [IDTechSDK::IDT_NEO2::emv_startTransaction\(\)](#) come back as EMV_RESULT_CODE.EMV_RESULT_CODE_AUTHENTICATE_TRANSACTION, this method must be called to continue the EMV transaction.

Complete Online EMV Transaction

```
IDTechSDK::IDT_NEO2::emv_completeTransaction()
```

After receiving a host response, pass host tags (minimum 8A Authorization Response Code) as a parameter.

If there was a communication error with host, you must still finish the EMV transaction by passing "TRUE" for commError.

Terminal Configuration

```
IDTechSDK::IDT_NEO2::emv_retrieveTerminalData()  
IDTechSDK::IDT_NEO2::emv_setTerminalData()
```

Methods for terminal configuration. When setting the terminal data, you pass the tags in TLV format.

AID Management

```
IDTechSDK::IDT_NEO2::emv_retrieveApplicationData  
IDTechSDK::IDT_NEO2::emv_removeApplicationData()  
IDTechSDK::IDT_NEO2::emv_removeAllApplicationData()  
IDTechSDK::IDT_NEO2::emv_setApplicationData()  
IDTechSDK::IDT_NEO2::emv_retrieveAIDList()
```

Methods for AID management. When setting the AID, you pass tags in TLV format. When retrieving AID, you can receive the results as tags in TLV format. When retrieving the AID list, the list of AID Names/length can be retrieved from the 2 dimensional byte array

CAPK Management

```
IDTechSDK::IDT_NEO2::emv_retrieveCAPK()
IDTechSDK::IDT_NEO2::emv_removeCAPK()
IDTechSDK::IDT_NEO2::emv_removeAllCAPK()
IDTechSDK::IDT_NEO2::emv_setCAPK()
IDTechSDK::IDT_NEO2::emv_retrieveCAPKList()
```

Methods for Certificate Authority Public Key management. When setting the CAPK, you populate and pass the key as a sequence of ordered bytes. When specifying a CAPK to retrieve or remove, you populate the name in the byte[] parameter. When retrieving the CAPK list, the list of RID/Index can be retrieved from the ordered byte stream, 6 bytes each, bytes 1-5 RID, byte 6 index

CRL Management

```
IDTechSDK::IDT_NEO2::emv_removeCRL()
IDTechSDK::IDT_NEO2::emv_removeAllCRL()
IDTechSDK::IDT_NEO2::emv_retrieveCRLList()
IDTechSDK::IDT_NEO2::emv_setCRL:()
```

Methods for Certificate Revocation List management.

Kernel Version

```
IDTechSDK::IDT_NEO2::emv_getEMVKernelVersion()
```

Method to retrieve kernel version. Valued returned in IDTResult.data

APDU Communication

```
IDTechSDK::IDT_NEO2::device_setPassThroughMode:()
IDTechSDK::IDT_NEO2::icc_powerOnICC:()
IDTechSDK::IDT_NEO2::icc_powerOffICC:()
```

```
IDTechSDK::IDT_NEO2::icc_exchangeAPDU:()
```

Allows the direct sending of APDU packets to ICC. Pass through mode must first be enabled. Then Power On needs to complete successfully. Then APDU packet exchange can take place

8.2 MSR/CTLS

Request Swipe or Tap

```
IDTechSDK::IDT_NEO2::msr_startMSRSwipe()
IDTechSDK::IDT_NEO2::ctls_startTransaction()
```

Both methods perform identical operation. Enables MSR to receive Swipe and CTLS to receive tap.

Cancel Swipe or Tap

```
IDTechSDK::IDT_NEO2::msr_cancelMSRSwipe()
IDTechSDK::IDT_NEO2::ctls_cancelTransaction()
```

Both methods perform identical operation. Cancels the Swipe/Tap request.

8.3 All Interfaces

Request any interface (Swipe, Tap, Insert)

[IDTechSDK::IDT_NEO2::device_startTransaction\(\)](#)

This method will allow any interface to capture data.

Chapter 9

Connecting to IDTech Devices

IDTech devices connects through USB, Serial Interface (COM), or IP depending on hardware capabilities.

[IDTechSDK](#) version 2.1.x.x is a multi-device SDK. I can maintain and execute commands simultaneously on multiple connected USB, RS-232 and IP devices.

All callbacks from the SDK are received in the MessageCallback function

```
private void MessageCallBack(IDTechSDK.IDT_DEVICE_Types type, DeviceState state, byte[] data,
    IDTTransactionData cardData, EMV_Callback emvCallback, RETURN_CODE transactionResultCode,
    string ident)
```

When a device connects, a callback will fire with DeviceState.Connected. This call back will contain the string "ident". That string is a unique identifier for the device that was detected. I will be in the following form:

DEVICE_TYPE: Connection-ID

DEVICE_TYPE = Device, example VP3300 Connection= USB, COM, or IP ID =

- USB will be the device serial number,
- COM will be the COM port #
- IP will be the IP Address and Port

Examples:

- VP3300:37T4972819
- KioskIII:COM-9
- VP6800:10.12.34.98:1442

Every API command has an optional parameter "ident". By putting the device identifier in this field, this will direct that command directly to that device, regardless of what the default device is for the SDK. Without that identifier, the command will go to whatever is the last device the SDK sent default commands to.

After executing setDeviceType, any commands with providing an ident will be directed to that device.

At any time, you can get a complete listing of all the devices conneced, listed by the ident, by executing the following:

```
List<string> ulist = Profile.getAllConnections();
```

You set the SDK default device that will be used for all commands, unless it is overridden by providing the ident with the command:

```
IDT_Device.setDeviceType(ident);
```


9.1 Connect with USB:

IDTech devices use USB-HID (Human Interface Device), or USB-KB format and does not need any vendor-specific drivers. Simply by plugging into an available USB port makes it available for SDK connectivity. To enable detection of USB devices, you need to execute the SDK command to monitor for the plugging/unplugging of devices:

```
IDT_Device.startUSBMonitoring();
```

There is also a command to exit the thread monitoring for the USB activity, which should be executed at the termination of your application

```
IDT_Device.stopUSBMonitoring();
```

9.2 Connect with Serial Interface (COM)

Most IDTech devices can connect via Serial Interface. The default serial port settings for most devices are as follows:

- Speed: 115,200
- Bits: 8
- Stop Bit: 1
- Parity: None

To inform the SDK you are using the Serial Interface , you would execute the following command during program initialization to establish a connection by passing the correct COM port number and device type you would like to connect to.

```
IDT_DEVICE_Types type = IDT_DEVICE_Types.IDT_DEVICE_MINISMARTII;  
int COMPORT = 1;  
bool isConnected = false;  
isConnected = IDT_NEO2.useSerialPort(COMPORT, type);
```

If you changed the baud rate of your device other than the factory default baud rate, you must also pass that to connect to it

```
IDT_DEVICE_Types type = IDT_DEVICE_Types.IDT_DEVICE_MINISMARTII;  
int COMPORT = 1;  
int BAUD = 9600  
bool isConnected = false;  
isConnected = IDT_NEO2.useSerialPort(COMPORT, type, BAUD);
```

You can check the return value to see if it found a valid COM port. If it did find the valid port, you can then tell the SDK to use that device

```
if (isConnected) IDT_Device.setDeviceType(type, DEVICE_INTERFACE_Types.DEVICE_INTERFACE_SERIAL);
```

9.3 Connect with TCP/IP

Some IDTech devices can connect via a network connection.

To inform the SDK you are using the IP interface of a NEO2 device, you would execute the following command during program initialization to establish a connection by passing the correct address. In the following example, we are connecting to a device as IP address 10.0.1.1. We need to specify the IP address, and if the device has TLS security

```
bool hasTLS = false;  
IDT_NEO2.ip_connectToSocket("10.0.1.1", hasTLS)
```

There can be multiple IP connected devices to a single SDK instance. Each one is initially connected and registered with `ip_connectToSocket`.

To get a listing of all registered devices the SDK has access to, a listing of all the strings passed to `ip_connectToSocket` can be retrieved by the following

```
List<string> devices = IDT_NEO2.ip_getSocketList();
```

To switch between devices listed in `ip_getSocketList()`:

```
IDT_NEO2.ip_switchToSocket("10.0.1.1");
```

NOTE: the string passed to `ip_switchToSocket` must match the complete gateway string as originally defined, and reported by `ip_getSocketList`.

9.4 Automatically Accept Incoming TLS Secure Connections

When using a device that has TLS, you can instruct the SDK to automatically accept incoming connections from any device that it can establish a secure channel with. You use function `monitorNetworkForDevices(enable, Optional:port)`, where `enable = true` will start monitoring, `enable = false` will stop monitoring, and the SSL port, which will default to 1443 if no value is parameter

```
IDT_Device.monitorNetworkForDevices(enable, port);
```

- `enable TRUE` = connect to all incoming TLS connections
- `port Default = 1443`. Only provide if another port is desired

9.5 Using Alternate TLS Certificates

The SDK contains a default TLS certificate used for IP secure communication. In cases where the device has been updated with another certificate not embedded in the SDK, before the SDK uses it's default certificate, it will scan the location where `IDTechSDK.dll` is located at to see if the file `"tls_cert.cer"` exists. If that file exists, it will use that certificate instead.

If IDTech provides you with an updated TLS certificate, please be sure to include that certificate in your project at the same locations as `IDTechSDK.dll`

Chapter 10

Core Implementation: WinForms

10.1 Integrating with IDTechSDK.dll

- [Import the necessary libraries](#)
- [Add using statements to utilize library](#)
- [Implement the callback functions](#)
- [Initialize Device](#)
- [Sample Source Code Info](#)

10.2 Import the necessary libraries

Communicating with IDTech Devices requires the following library to be referenced by the project:

- IDTechSDK.dll

To obtain this library, and keep in sync with the latest updates, you need to import the library from NuGet. The NuGet package is "IDTechSDK_STD". This will also install any other dependencies needed for the particular project you are building for.

- [Obtaining Latest SDK](#)

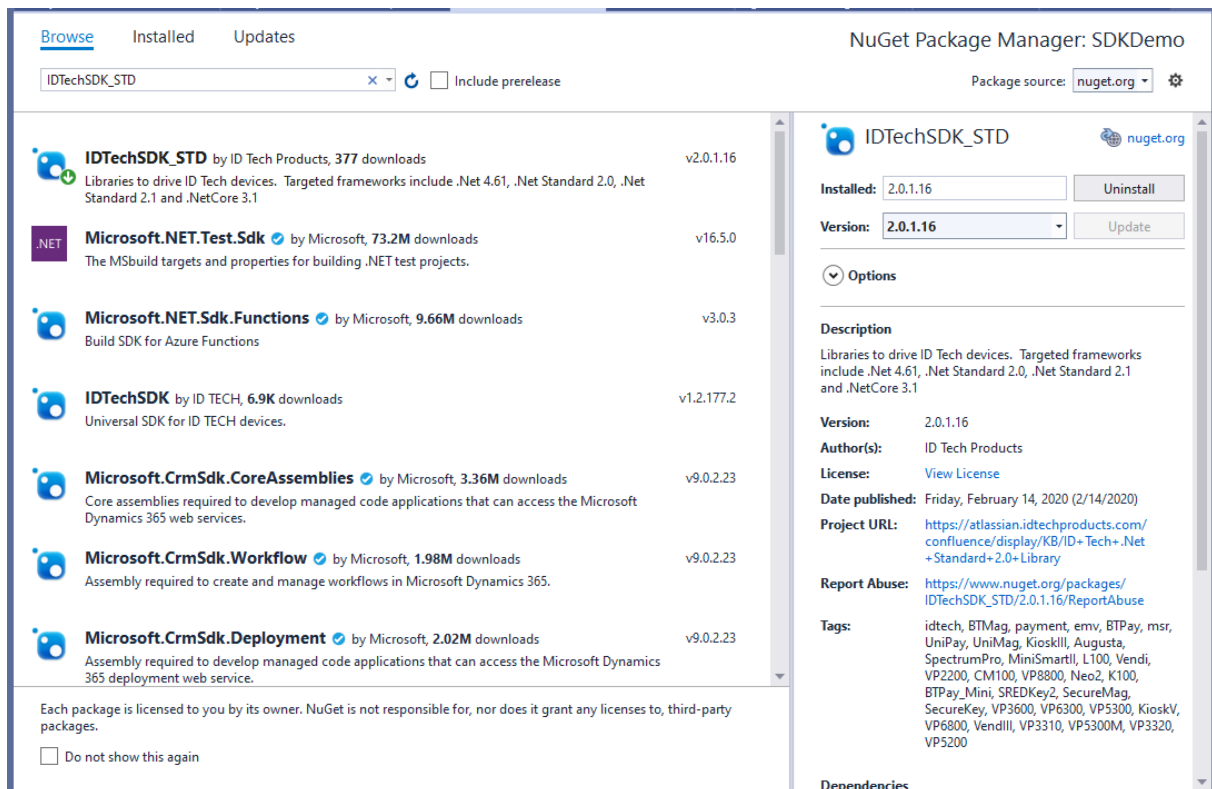
10.2.1 Obtaining Latest SDK

The IDTech SDK will be maintained as a NuGet package.

To install IDTechSDK.dll into your project, right-click on your project and select "Manage NuGet Packages..."

Browse for "IDTechSDK_STD". Once located, install a version for your project.

The [IDTechSDK](#) currently can be used in .NetFramework projects 4.6.1 or greater, .Net Standard Projects 2.1, .NetCore 3.1, or Net6.0



10.3 Add using statements to utilize library

Add a line of code to the class that will utilize IDTechSDK.dll at the start of the file:

```
using IDTechSDK;
```

10.4 Implement the callback functions

There are currently two callback functions defined, one for button press callback, and another universal callback for all other requirements. The universal callback uses DeviceState to determine which action to take. For IP connected devices, the last parameter is provided for instance where multiple IP connected devices may be sending data back to this callback.

```
private void MessageCallBack(IDTechSDK.IDT_DEVICE_Types type, DeviceState state, byte[] data,
    IDTTransactionData cardData, EMV_Callback emvCallback, RETURN_CODE transactionResultCode, string IP)
{
    switch (state)
    {
        case DeviceState.ToConnect:
            //A connection attempt is starting for IDT_DEVICE_TYPES type
            break;
        case DeviceState.DefaultDeviceTypeChange:
            //The SDK is changing the default device to IDT_DEVICE_TYPES type
            break;
        case DeviceState.Connected:
            //A connection has been made to IDT_DEVICE_TYPES type
            break;
        case DeviceState.Disconnected:
            //A disconnection has occurred with IDT_DEVICE_TYPES type
            break;
        case DeviceState.ConnectionFailed:
            //A connection attempt has failed for IDT_DEVICE_TYPES type
            break;
        case DeviceState.TransactionData:
            //Transaction data is being returned in IDTTransactionData cardData
            break;
    }
}
```

```

        case DeviceState.DataReceived:
            //Low-level data received for IDT_DEVICE_TYPES type
            break;
        case DeviceState.DataSent:
            //Low-level data sent for IDT_DEVICE_TYPES type
            break;
        case DeviceState.CommandTimeout:
            //Command timeout has occurred for IDT_DEVICE_TYPES type
            break;
        case DeviceState.ToSwipe:
            //Awaiting a swipe for IDT_DEVICE_TYPES type
            break;
        case DeviceState.MSRDecodeError:
            //Awaiting a swipe for IDT_DEVICE_TYPES type
            break;
        case DeviceState.ToTap:
            //Awaiting a contactless tap for IDT_DEVICE_TYPES type
            break;
        case DeviceState.SwipeTimeout:
            //Waiting for swipe timed out
            break;
        case DeviceState.TransactionCancelled:
            //Transaction has been cancelled
            break;
        case DeviceState.DeviceTimeout:
            //Waiting for device timeout
            SetOutputText("Callback:DeviceTimeout\n");
            break;
        case DeviceState.TransactionFailed:
            //Transaction failed to complete
            break;
        case DeviceState.EMVCallback:
            SetOutputText("EMV Callback\n");
            break;
        case DeviceState.PINpadKeypress:
            SetOutputText("PINPad Key was pressed\n");
            break;
        case DeviceState.PINTimeout:
            SetOutputText("PINPad Timeout\n");
            break;
        case DeviceState.MenuTimeout:
            SetOutputText("Menu Entry Timeout\n");
            break;
        case DeviceState.TransactionIdle:
            SetOutputText("Ready For Transaction\n");
            break;
        case DeviceState.Signature:
            SetOutputText("Signature Captured\n");
            break;
        case DeviceState.FirmwareUpdate:
            SetOutputText("firmware update\n");
            break;
        case DeviceState.FileTransfer:
            SetOutputText("File Transfer\n");
            break;
        case DeviceState.ConfigurationGroup:
            SetOutputText("Config Group\n");
            break;
        case DeviceState.InvalidInterface:
            SetOutputText("Invalid Interface\n");
            break;
        case DeviceState.FunctionKey:
            SetOutputText("Function Key Pressed\n");
            break;
        case DeviceState.PINFail:
            SetOutputText("PIN Fail\n");
            break;
        case DeviceState.SocketEstablished:
            SetOutputText("Socket connection established to: " + IP + " \r\n");
            break;
        case DeviceState.SocketFailed:
            SetOutputText("Socket connection failed to: " + IP + " \r\n");
            break;
        case DeviceState.ButtonEvent:
            SetOutputText("Screen ID: " + screenID.ToString() + "\r\n");
            SetOutputText("Button ID: " + ID.ToString() + "\r\n");
            break;
        default:
            break;
    }
}
}

```

When `lcd_addButton` is executed, a `buttonCallback` can be passed that will report all press events for that button. If `buttonCallback` is set to `NULL`, then all press events will be returned to the main callback as a `DeviceState.Button↔Event`. The `buttonCallback` has the following signature, passing back the device type, the screen, the button ID, and the device IP address:

```
private void ButtonCallback(IDT_DEVICE_Types sender, UInt16 screenID, UInt16 itemID, string IP)
{
    SetOutputText("BUTTON CALLBACK, IP : " + IP + "\n");
    SetOutputText("Button Pressed, Screen ID: " + screenID.ToString() + "\n");
    SetOutputText("Button Pressed, Button ID: " + itemID.ToString() + "\n");
}
```

For the universal callback, there is an `OPTIONAL` parameter to associate that callback with just a single IP address.

Setting button callback is established during `lcd_addButton`, in addition, there is another method to set the callback of any existing button.

10.5 Initialize Device:

A Singleton instance model is used to the class. Establish the callback, and then access the device through USB via the `SharedController` instance. Or, you can establish through Serial Port or IP if available for your device

```
public NEO2_Simple_Demo()
{
    InitializeComponent();
    IDT_NEO2.setCallbackIP(MessageCallBack);
    IDT_Device.startUSBMonitoring(); //connect via USB
    // bool isConnected = IDT_NEO2.useSerialPort(1); //use serial port 1
    // IDT_NEO2.ip_connectToSocket("10.0.1.1", false); //connect to socket 10.0.1.1, no TLS
}
```

Alternately, the `IDTechSDK.dll` library can be used to drive multiple `IDTech` Devices. If the plan is to use two or more devices, then the `IDT_Device` class should be used instead of the individual device class. You can then tell the SDK what device to use, and then use the same code/API method for both devices:

```
public MultiDevice_Simple_Demo()
{
    InitializeComponent();
    IDT_Device.setCallback(MessageCallBack);
    IDT_Device.startUSBMonitoring();
    string firmwareVersion = "";

    string SREDIdent = "39Y30259300";
    string VP3300Ident = "20T4926526"

    //Let's send command first to SREDKey2 using that device identifier

    RETURN_CODE rt = IDT_Device.SharedController.device_getFirmwareVersion(ref firmwareVersion, SREDIdent);

    //Let's now switch to the VP3300 using that device identifier

    rt = IDT_Device.SharedController.device_getFirmwareVersion(ref firmwareVersion, VP3300Ident);
}
```

The SDK is driven by the commands in the `IDT_Device` class. This class has all the functions for many different `IDTech` devices. As a result, there may be many functions that may not be applicable to your particular device. That is why we also provide the individual device class, like `IDT_SREDKey2`. When you use that individual device class, that acts as a filter of all the available commands that can be executed in the `IDT_Device` class. If you attempt to use an API method in `IDT_Device` class not intended for your particular device, either the SDK or the device FW will return an applicable error code as a response.

10.6 Sample Source Code Info

A full featured SDK Demo application with source code is provided by IDTech to provide a code example to aid in understanding code operation.

Please refer to the source code for guidance.

SDK Reference Page: <https://atlassian.idtechproducts.com/confluence/display/KB/Universal+Library+for+Visual+Studio+-+Home?desktop=true¯oName=attachments>

.Net Source Code: <https://atlassian.idtechproducts.com/confluence/download/attachments/304796/NET%20SDK%20Demo%20Source%20Code.zip?api=v2>

Chapter 11

L100 Pass-Through Mode

On some products, the L100 Pinpad interfaces directly with the unit. Normally, the L100 would be controlled by the NEOII device as needed, for example, collecting a PIN during an EMV transaction. But there may be instances where sending direct commands to the L100 is required by the integrator. For this, we have a L100 Pass-Through Mode.

Enabling L100 Pass-Through Mode

Pass a value to TRUE to [IDTechSDK::IDT_NEO2::device_enablePassThrough\(\)](#)

You then send IDT_L100.SharedController commands according to the commands available in the IDT_L100 class. For example, to reboot device, you send:

```
IDT_L100.SharedController.rebootDevice();
```

Disabling L100 Pass-Through Mode

Pass a value to FALSE to [IDTechSDK::IDT_NEO2::device_enablePassThrough\(\)](#)

You now resume sending all commands to the IDT_NEO2.SharedController

Chapter 12

Virtual Device Operation

The purpose of the SDK is to execute API commands and process the responses. While an API command such as `getFirmware` may seem straight forward in execution, the SDK needs to decide on the proper protocol depending on the device (IDG, NGA, ITP) to properly wrap the command, then it must prepare the data for the proper interface. A serial interface is unaltered command data, but if the interface is USB-HID, it can be two different type of HID report packets. If it is USB-KB, then it must execute 8-byte `GetFeature/SetFeature` packets.

It may be desired to leverage the SDK to communicate with devices not directly attached to the SDK. Examples of this would be for a company to connect to the device using their own SDK, or if the device is attached to an iOS or Android device through Xamarin.

This can be accomplished by configuring the SDK for Virtual Device Operation. You accomplish this by implementing a callback that will receive all the data that SDK wants to send to the device, and then any data that is received from the device you send back to the SDK. We are basically putting a hook in the SDK and diverting the data that would have normally gone to its COM or USB ports.

Step 1: Define the Device Write Callback

Implement a callback that will receive all the data the SDK would like to transmit to the device:

```
private void virtualWriteDelegate(byte[] data)
{
}
```

Step 2: Initialize the SDK

Set the SDK message callback (standard SDK implementation):

```
IDT_Device.SetCallbackIP(MessageCallBack);
```

Request the SDK to open a virtual device, specifying the virtual device type, the connection type, and the virtual write callback

```
comm = IDT_Device.openVirtualDevice(IDT_DEVICE_Types.IDT_DEVICE_KIOSKIII, DEVICE_INTERFACE_Types.DEVICE_INTERFACE_USB, virtualWriteDelegate);
```

if "comm" does not come back null, the virtual device is initialized. This "comm" instance of `IDTechComm` will be needed to send device responses back into the SDK.

Step 3: Execute Commands

Execute a standard SDK command

```
RETURN_CODE rt = IDT_Device.SharedController.device_getFirmwareVersion(ref fw);
```

The SDK will construct the command and deliver it to the `virtualWriteDelegate`

```
private void virtualWriteDelegate(byte[] data)
{
    //outgoing command received here. Send to connected device
}
```

Take that commands and direct it to a connected device (Xamarin->Android, Xamarin-iOS or other destination)

Step 4: Monitor Device Communication

You must continually monitor the device for any data it is sending out. This can be data in response to a command, a notification, or even transaction data that may have been started on a different thread. Any data that is returned from the device must be directed back into the SDK:

```
if (comm != null)
{
    comm.virtualRead(data);
}
```

Step 5: Device Type and Connection Settings

The SDK supports the following device types :

```
public enum IDT_DEVICE_Types
{
    IDT_DEVICE_BTPAY,
    IDT_DEVICE_BTMAg,
    IDT_DEVICE_UNIPAY,
    IDT_DEVICE_UNIPAYI_V,
    IDT_DEVICE_UNIMAG,
    IDT_DEVICE_SPECTRUM_PRO,
    IDT_DEVICE_MINISMARTII,
    IDT_DEVICE_AUGUSTA,
    IDT_DEVICE_AUGUSTA_KB,
    IDT_DEVICE_KIOSKIII,
    IDT_DEVICE_CM100,
    IDT_DEVICE_VENDI,
    IDT_DEVICE_L100,
    IDT_DEVICE_VENDIII,
    IDT_DEVICE_VP3300,
    IDT_DEVICE_VP8800,
    IDT_DEVICE_SECUREMAG,
    IDT_DEVICE_BTPAY_MINI,
    IDT_DEVICE_K100,
    IDT_DEVICE_NEO2,
    IDT_DEVICE_TMS,
    IDT_DEVICE_SECUREKEY,
    IDT_DEVICE_SREDKEY2,
    IDT_DEVICE_PIP
}
```

The SDK supports the following interface types for Virtual Devices

```
public enum DEVICE_INTERFACE_Types
{
    DEVICE_INTERFACE_USB,
    DEVICE_INTERFACE_SERIAL
}
```

Please note, the Device Type should accurately reflect the type of device being communicated with, but the Interface Type indicates how data will be provided and expected in return.

If DEVICE_INTERFACE_SERIAL is selected, all commands are complete command packets in their proper protocol wrapping, no additional modification are made. The response is expected to be the same.

If DEVICE_INTERFACE_USB is selected, the commands are further processed into USB Reports. These are 64 or 65 byte report format (when USB-HID), or 8 or 9 byte report format when USB-KB. Data is provided in the raw format the device will understand. It is expected the data will be sent through a communication bridge that is just acting as pass through mode.

Example: if you were to execute getFirmware to a virtual device that was defined as serial, a Kiosk IV, which is an IDG Device, will produce the command wrapped in IDG protocol with no additional data:

```
data = 0x5669564f74656368320029000000dea0
```

```
data = 0x5669564f746563683200290000124b696f736b2049562056312e32302e3130390aff
```

```
//pass this data to comm.virtualRead(data);
```

[illegible]

```
data =  
0x015669564f746563683200290000124b696f736b2049562056312e32302e3130390aff000000000000000000000000000000000000000000000000
```

```
//pass this data to comm.virtualRead(data);
```

Step 6: Closing Virtual Device

The SDK can only communicate with a single Virtual Device. The Virtual Device should be closed on exit. The Virtual Device should be closed if another Virtual Device is desire to be used. To close the currently open Virtual Device, execute:

```
IDT_Device.closeVirtualDevice();
```

Chapter 13

LCD Foreign Language Mapping Table

ID	Message ID	English	French	Spanish	Chinese
0	MSG_NULL	-	-	-	-
1	MSG_AMOUNT	AMOUNT	MONTANT	CANTIDAD	金
2	MSG_AMOUNT_↔ OK	AMOUNT OK?	MONTANT OK	MONTO CORRE↔ CTO?	确定金
3	MSG_APPROVED	APPROVED	APPROUVE	APROVADO	通
4	MSG_CALL_YO↔ UR_BANK	CALL YOUR BANK	APPE VOTRE B↔ ANQE	LLAME A SU BA↔ NCO	系您的行
5	MSG_CANCEL_↔ OR_ENTER	CANCEL OR EN↔ TER	ANNULE OU EN↔ TRER	CANCEL O ENT↔ RAR	取消或确定
6	MSG_CARD_ER↔ ROR	CARD ERROR	ERREUR CARTE	ERROR DE TAR↔ JETA	卡
7	MSG_DECLINED	DECLINED	REFUSE	DECLINADO	卡被拒
8	MSG_ENTER_A↔ MOUNT	ENTER AMOUNT	ENTRER MONT↔ ANT	INGRESE MONTO	入金
9	MSG_ENTER_PIN	ENTER PIN:	ENTRER PIN:	ENTRAR NPI:	入密
10	MSG_INCORRE↔ CT_PIN	INCORRECT PIN	NIP INCORRECT	NPI INCORRECTO	密
11	MSG_ICC_MSR1	SWIPE OR INSE↔ RT	PASSER OU INS↔ ERT	MOVER O INSERT	刷卡或插卡
12	MSG_ICC_MSR2	CARD	CARTE	TARJETA	卡
13	MSG_INSERT_↔ CARD	INSERT CARD	INSERT CARTE	INSERTAR TAR↔ JETA	插卡
14	MSG_USE_CHI↔ P_READER	USE CHIP READ↔ ER UTI	LECTEUR CHIP	USO CHIP LECT↔ OR	使用芯片卡
15	MSG_NOT_ACC↔ EPTED	NOT ACCEPTED	PAS ACCEPTE	DENEGADO	法接受
16	MSG_PIN_OK	GET PIN OK			密正确
17	MSG_PLEASE_↔ WAIT	PLEASE WAIT...	ATTENDRE...	POR FAVOR ES↔ PERE	等候中
18	MSG_PROCESS↔ ING_ERROR	PROCESSING E↔ RROR	ERREUR DE TR↔ AITE	ERROR PROCE↔ SANDO	理
19	MSG_USE_MAG↔ STRIPE	USE MAGSTRIPE	USAGE MAGST↔ RIPE	USO DE MAGST↔ RIPE	使用磁卡
20	MSG_TRY_AGAIN	TRY AGAIN	REESSAYER	VUELV INTENTA↔ RLO	重
21	MSG_ONLINE	GO ONLINE	GO LIGNE	GO LINEA	在

ID	Message ID	English	French	Spanish	Chinese
22	MSG_TRANSACTION_ERROR↔	TRANSACTION ERR	ERREUR DE TRAN↔	ERROR DE TRA↔NSAC	交易
23	MSG_TERMINATE	TERMINATE	RESILIER	TERMINAR	止
24	MSG_ADVICE	ADVICE	CONSEILS	CONSEJOS	建
25	MSG_TIMEOUT	TIME OUT	TIMEOUT	TIEMPO DE ESP↔ERA	超
26	MSG_PROCESSING↔	PROCESSING...	PROCESSUS...	PROCESANDO...	理中。。。
27	MSG_PIN_TRY_EX↔	PIN TRY LIMIT EX	PIN TRY DEPAS↔SE	TRY PIN SUPER↔ADA	密次多
28	MSG_ISSUER_AUTH_FAIL↔	ISSUER AUTH FAIL	EMETTEUR FAIL	EMISOR FALLA	与卡机构
29	MSG_CONTINUE_PROCESS↔	CONTINUE PROCESS	CONTINUER LA	CONTINUAR PR↔OCES	理
30	MSG_GET_PIN_ERROR↔	GET PIN ERROR	GET PIN ERROR	OBTENER PIN E↔RR	密
31	MSG_GET_PIN_FAIL↔	GET PIN FAIL	GET PIN FAIL	OBTENER PIN F↔ALL	取密
32	MSG_NOKEY_GET_PIN↔	NO KEY GET PIN	NO KEY GET PIN	NO CLAVE GET PIN	法入密
33	MSG_CANCELLED↔	CANCELLED	ANNULE	CANCELADO	取消
34	MSG_LAST_PIN_TRY↔	LAST PIN TRY	-	-	最后一次入密

Chapter 14

Error Code Reference

```
public static string getErrorString(RETURN_CODE code)
{
    switch ((int)code)
    {
        case 0: return "no error, beginning task";
        case 1: return "no response from reader";
        case 2: return "invalid response data";
        case 3: return "time out for task or CMD";
        case 4: return "wrong parameter";
        case 5: return "SDK is doing MSR or ICC task";
        case 6: return "SDK is doing PINPad task";
        case 7: return "SDK is doing CTLS task";
        case 8: return "Firmware return timeout value";
        case 9: return "SDK is doing Other task";
        case 10: return "err response or data";
        case 11: return "no reader attached";
        case 12: return "mono audio is enabled";
        case 13: return "did connection";
        case 14: return "audio volume is too low";
        case 15: return "task or CMD be canceled";
        case 16: return "UF wrong string format";
        case 17: return "UF file not found";
        case 18: return "UF wrong file format";
        case 19: return "Attempt to contact online host failed";
        case 20: return "Attempt to perform RKI failed";
        case 21: return "Missing DLL. Cannot access device.";
        case 22: return "Firmware Block Transfer Successful";
        case 23: return "SDK is doing Firmware Update task";
        case 24: return "Applying the firmware update downloaded to memory";
        case 25: return "No Data Available";
        case 26: return "SDK Busy doing file transfer task";
        case 27: return "Applying the file transfer";
        case 28: return "File Transfer Successful";
        case 29: return "Not enough space available on drive";
        case 30: return "Entering Bootload Mode";
        case 31: return "Starting Firmware Update";
        case 32: return "PCI File Mismatch";
        case 33: return "Incorrect FW File Block Size";
        case 34:
            return "Device is busy finalizing transaction.";
        case 35:
            return "The SDK Busy doing RKI update";
        case 36:
            return "Bad MSR Swipe";
        case 37:
            return "Financial card not allowed";
        case 38:
            return "SDK Busy waiting for input event";
        case 39:
            return "Unsupported Command";
        case 40:
            return "Erasing SPI VP8800";
        case 41:
            return "SDK is doing EMV task";
        case 42:
            return "SDK is doing Camera task";
        case 43:
            return "SDK is doing ViVO Config task";
        case 44:
```

```

        return "SDK is starting VIVO Config task";
    case 45:
        return "SDK is finishing ViVO Config task";
    case 46:
        return "ViVO Config task failed";
    case 47:
        return "ViVO Config Message";
    case 48:
        return "Failed to change ViVO Config mode";
    case 49:
        return "VivoConfig updated was cancelled";
    case 50:
        return "ViVO Config Verify Success";
    case 51:
        return "ViVO Vonfig Verify failure";
    case 0x100: return "Log is full";
    case 0x300: return "Key Type(TDES) of Session Key is not same as the related Master Key.";
    case 0x400: return "Related Key was not loaded.";
    case 0x410: return "Non-SRED Device need Load Manufacture Key and Firmware Key.";
    case 0x500: return "Key Same.";
    case 0x501: return "Key is all zero";
    case 0x502: return "TR-31 format error";
    case 0x700: return "No BDK of Pairing MSR Key.";
    case 0x701: return "Have BDK of Pairing MSR Key, Not Pairing with MSR (No PAN Encryption
Key)";
    case 0x702: return "PAN is Error Key.";
    case 0x703: return "Pairing Failed";
    case 0x704: return "MSR Pairing Key Other Error";
    case 0x705: return "No Internal MSR PAN (or Internal MSR PAN is erased timeout)";

    case 0X0800: return "Invalid Pinpad Data Received";
    case 0X0801: return "Key Pad Cancel";
    case 0X0802: return "External Command Cancel";
    case 0X0803: return "Invalid Input Parameters";
    case 0X0804: return "PAN Error";
    case 0X0805: return "PIN DUKPT Key is absent";
    case 0X0806: return "PIN DUKPT Key is exhausted";
    case 0X0807: return "Display Error Message";
    case 0X080C: return "Not Allowed";

    case 0X0C01: return "Incorrect Frame Tag";
    case 0X0C02: return "Incorrect Frame Type";
    case 0X0C03: return "Unknown Frame Type";
    case 0X0C04: return "Unknown Command";
    case 0X0C05: return "Unknown Sub-Command";
    case 0X0C06: return "CRC Error";
    case 0X0C07: return "Failed";
    case 0X0C08: return "Timeout";
    case 0X0C0A: return "Incorrect Parameter";
    case 0X0C0B: return "Command Not Supported";
    case 0X0C0C: return "Sub-Command Not Supported";
    case 0X0C0D: return "Parameter Not Supported / Status Abort Command";
    case 0X0C0F: return "Sub-Command Not Allowed";
    case 0X0D01: return "Incorrect Header Tag";
    case 0X0D02: return "Unknown Command";
    case 0X0D03: return "Unknown Sub-Command";
    case 0X0D04: return "CRC Error in Frame";
    case 0X0D05: return "Incorrect Parameter";
    case 0X0D06: return "Parameter Not Supported";
    case 0X0D07: return "Mal-formatted Data";
    case 0X0D08: return "Timeout";
    case 0X0D0A: return "Failed / NACK";
    case 0X0D0B: return "Command not Allowed (Passthrough Mode May Be On)";
    case 0X0D0C: return "Sub-Command not Allowed";
    case 0X0D0D: return "Buffer Overflow (Data Length too large for reader buffer)";
    case 0X0D0E: return "User Interface Event";
    case 0X0D0F: return "Decryption Error";
    case 0X0D11: return "Communication type not supported, VT-1, burst, etc.";
    case 0X0D12: return "Secure interface is not functional or is in an intermediate state.";
    case 0X0D13: return "Data field is not mod 8";
    case 0X0D14: return "Pad 0x80 not found where expected";
    case 0X0D15: return "Specified key type is invalid";
    case 0X0D16: return "Busy - device, resource or system is busy";
    case 0X0D17: return "Hash code problem";
    case 0X0D18: return "Could not store the key into the SAM(InstallKey)";
    case 0X0D19: return "Frame is too large";
    case 0X0D1A: return "Unit powered up in authentication state but POS must resend the
InitSecureComm command";
    case 0X0D1B: return "The EEPROM may not be initialized because SecCommInterface does not
make sense";
    case 0X0D1C: return "Problem encoding APDU";
    case 0X0D20: return "Unsupported Index(ILM) SAM Transceiver error - problem communicating
with the SAM(Key Mgr)";
    case 0X0D21: return "Unexpected Sequence Counter in multiple frames for single bitmap(ILM)

```

```

Length error in data returned from the SAM(Key Mgr)";
case 0X0D22: return "Improper bit map(ILM)";
case 0X0D23: return "Request Online Authorization (Contactless Only)";
case 0X0D24: return "ViVOCard3 raw data read successful";
case 0X0D25: return "Message index not available(ILM) ViVOcomm activate transaction card
type(ViVOcomm)";
case 0X0D26: return "Version Information Mismatch(ILM)";
case 0X0D27: return "Not sending commands in correct index message index(ILM)";
case 0X0D28: return "Time out or next expected message not received(ILM)";
case 0X0D29: return "ILM languages not available for viewing(ILM)";
case 0X0D2A: return "Other language not supported(ILM)";
case 0X0D2B: return "Device Not Ready";
case 0X0D2C: return "The 8341 operation was exited by the Cancel Transaction command";
case 0X0D2D: return "The 8341 operation was exited by the user pressing the cancel key";
case 0X0D30: return "Request Online Authorization (Contact Only)";
case 0X0D31: return "Request Online PIN (for Cardholder Verification Method -CVM)";
case 0X0D32: return "Request Signature (for Cardholder Verification Method - CVM)";
case 0X0D33: return "Advice Required";
case 0X0D34: return "Reversal Required";
case 0X0D35: return "Advice and Reversal Required";
case 0X0D36: return "No Advice or Reversal Required (Declined)";
case 0X0D39: return "Reserved";
case 0X0D3A: return "Fallback from Contact (ICC) to Magstripe";
case 0X0D3B: return "Fallback from Contactless to Contact";
case 0X0D3C: return "Fallback from Contactless to Magstripe";
case 0X0D3D: return "Fallback from Contactless to Other Interface";
case 0X0D41: return "Unknown Error from SAM";
case 0X0D42: return "Invalid data detected by SAM";
case 0X0D43: return "Incomplete data detected by SAM";
case 0X0D44: return "Reserved";
case 0X0D45: return "Invalid key hash algorithm";
case 0X0D46: return "Invalid key encryption algorithm";
case 0X0D47: return "Invalid modulus length";
case 0X0D48: return "Invalid exponent";
case 0X0D49: return "Key already exists";
case 0X0D4A: return "No space for new RID";
case 0X0D4B: return "Key not found";
case 0X0D4C: return "Crypto not responding";
case 0X0D4D: return "Crypto communication error";
case 0X0D4E: return "Module-specific error for Key Manager";
case 0X0D4F: return "All key slots are full (maximum number of keys has been installed)";
case 0X0D50: return "Key data is bad or corrupt";
case 0X0D51: return "Keys out of sync between contactless and contact interfaces (use this
command on one interface at a time)";
case 0X0D55: return "Illegal duplicate data set";
case 0X0D56: return "Indication to fail the transaction to Pass Through Mode";
case 0X0D57: return "No payment transaction occurred. The response is loyalty only.";
case 0X0D59: return "Start Transaction successful. Perform Authentication..";
case 0X0D60: return "Data does not exist";
case 0X0D61: return "Data Full";
case 0X0D62: return "Write Flash Error";
case 0X0D63: return "OK and Have Next Command";
case 0X0D90: return "Account DUKPT Key doesn't exist";
case 0X0D91: return "Account DUKPT Key KSN exhausted";
case 0X0DA3: return "CTLS/MSR was initiated, but card was inserted into Contact interface";
case 0X0DA1: return "No PAN available to encode PIN Block with";
case 0X0DA0: return "Invalid Screen";
case 0X0D00: return "This Key had been loaded.";
case 0X0E00: return "Base Time was loaded.";
case 0X0F00: return "Encryption Or Decryption Failed.";
case 0x1000: return "Battery Low Warning (It is High Priority Response while Battery is
Low.)";
case 0x1800: return "Send \"Cancel Command\" after send \"Get Encrypted PIN\" &\"Get Numeric \"&
\"Get Amount\"";
case 0x1900: return "Press \"Cancel\" key after send \"Get Encrypted PIN\" &\"Get Numeric \"&
\"Get Amount\"";
case 0x30FF: return "Security Chip is not connect";
case 0x3000: return "Security Chip is deactivation & Device is In Removal Legally State.";
//case 0x3005: return "Removal detection not active";
case 0x3010: return "Master Chip was Activated.";
//case 0x30FF: return "Slave Chip is not connect.";
case 0x3101: return "Security Chip is activation & Device is In Removal Legally State.";
case 0x5500: return "No Admin DUKPT Key.";
case 0x5501: return "Admin DUKPT Key STOP.";
case 0x5502: return "Admin DUKPT Key KSN is Error.";
case 0x5503: return "Get Authentication Code1 Failed.";
case 0x5504: return "Validate Authentication Code Error.";
case 0x5505: return "Encrypt or Decrypt data failed.";
case 0x5506: return "Not Support the New Key Type.";
case 0x5507: return "New Key Index is Error.";
case 0x5508: return "Step Error.";
case 0x5509: return "KSN Error";
case 0x550A: return "MAC Error.";
case 0x550B: return "Key Usage Error.";
case 0x550C: return "Mode Of Use Error.";
case 0x550F: return "Other Error.";
case 0x5530: return "Key Number Error";

```



```

        case 0x5531: return "Key Block length error";
        case 0x5532: return "Key Length Error";
        case 0x5533: return "HMAC checking error";
        case 0x6000: return "Save or Config Failed / Or Read Config Error.";
        case 0x6200: return "No Serial Number.";
        case 0x6900: return "Invalid Command - Protocol is right, but task ID is invalid.";
        case 0x6A00: return "Unsupported Command - Protocol and task ID are right, but command is
invalid.";
        case 0x6A01: return "Unsupported Command - Protocol and task ID are right, but command is
invalid - In this State";
        case 0x6A02: return "Unsupported Command - Protocol and task ID are right, but command is
invalid - for disable";
        case 0x6B00: return "Unknown parameter in command - Protocol task ID and command are right,
but parameter is invalid.";
        case 0x6C00: return "Unknown parameter in command - Protocol task ID and command are right,
but length is out of the requirement.";
        case 0x6D00: return "Beeper Control Error.";
        case 0x6D01: return "LED Control Error.";
        case 0x6D10: return "ICC Reading Function Disabled (Do Not Support ICC related reading
characteristics commands).";
        case 0x6D11: return "MSR Reading Function Disabled (Do Not Support MSR related reading
characteristics commands).";
        case 0x7200: return "Device is suspend (MKSX suspend or press password suspend).";
        case 0x7300: return "PIN DUKPT is STOP (21 bit 1).";
        case 0x7400: return "Device is Busy.";
        //case 0x7500: return "Device is in diagnose mode.";
        case 0x7600: return "Device is in Transparent Transmission mode";
        case 0xE100: return "Can not enter sleep mode";
        case 0xE200: return "File has existed";
        case 0xE300: return "File has not existed";
        case 0xE313: return "IO line low -- Card error after session start";
        case 0xE400: return "Open File Error";
        case 0xE500: return "SmartCard Error";
        case 0xE600: return "Get MSR Card data is error";
        case 0xE700: return "Command time out";
        case 0xE800: return "File read or write is error";
        case 0xE900: return "Active 1850 error!";
        case 0xEA00: return "Load bootloader error";
        case 0xEF00: return "Protocol Error- STX or ETX or check error.";
        case 0xEB00: return "Picture is not exist";
        case 0x2C02: return "No Microprocessor ICC seated";
        case 0x2C06: return "no card seated to request ATR";
        case 0x2D01: return "Card Not Supported,";
        case 0x2D03: return "Card Not Supported, wants CRC";
        case 0x690D: return "Command not supported on reader without ICC support";
        case 0x8100: return "Timeout";
        case 0x8200: return "invalid TS character received - Wrong operation step";
        case 0x8300: return "Decode MSR Error";
        case 0x8400: return "TriMagII no Response";
        case 0x8500: return "No Swipe MSR Card";
        case 0x8510: return "No Financial Card";
        case 0x8600: return "Unsupported F, D, or combination of F and D";
        case 0x8700: return "protocol not supported EMV TD1 out of range";
        case 0x8800: return "power not at proper level";
        case 0x8900: return "ATR length too long";
        case 0x8B01: return "EMV invalid TA1 byte value";
        case 0x8B02: return "EMV TB1 required";
        case 0x8B03: return "EMV Unsupported TB1 only 00 allowed";
        case 0x8B04: return "EMV Card Error, invalid BWI or CWI";
        case 0x8B06: return "EMV TB2 not allowed in ATR";
        case 0x8B07: return "EMV TC2 out of range";
        case 0x8B08: return "EMV TC2 out of range";
        case 0x8B09: return "per EMV96 TA3 must be > 0xF";
        case 0x8B10: return "ICC error on power-up";
        case 0x8B11: return "EMV T=1 then TB3 required";
        case 0x8B12: return "Card Error, invalid BWI or CWI";
        case 0x8B13: return "Card Error, invalid BWI or CWI";
        case 0x8B17: return "EMV TC1/TB3 conflict*";
        case 0x8B20: return "EMV TD2 out of range must be T=1";
        case 0x8C00: return "TCK error";
        case 0xA304: return "connector has no voltage setting";
        case 0xA305: return "ICC error on power-up invalid (SBLK(IFSD) exchange";
        case 0xE301: return "ICC error after session start";
        case 0xFF00: return "Request to go online";
        case 0xFF01: return "EMV: Accept the offline transaction";
        case 0xFF02: return "EMV: Decline the offline transaction";
        case 0xFF03: return "EMV: Accept the online transaction";
        case 0xFF04: return "EMV: Decline the online transaction";
        case 0xFF05: return "EMV: Application may fallback to magstripe technology";
        case 0xFF06: return "EMV: ICC detected tah the conditions of use are not satisfied";
        case 0xFF07: return "EMV: ICC didn't accept transaction";
        case 0xFF08: return "EMV: Transaction was cancelled";
        case 0xFF09: return "EMV: Application was not selected by kernel or ICC format error or ICC
missing data error";
        case 0xFF0A: return "EMV: Transaction is terminated";
        case 0xFF0B: return "EMV: Other EMV Error";
        case 0xFFFF: return "NO RESPONSE";

```

```

case 0xF002: return "ICC communication timeout";
case 0xF003: return "ICC communication Error";
case 0xF005: return "ICC Encrypted C - APDU Data Structure Length Error Or Format Error.";
case 0xF00F: return "ICC Card Seated and Highest Priority, disable MSR work request";
case 0xF20F: return "No Financial Card";
case 0xF210: return "In Encrypt Result state, TLV total Length is greater than Max Length";
case 0xF211: return "ICC L2 is not in idle state.";
case 0xF212: return "Transaction Type Error.";
case 0xF213: return "Major Config Error for Set Terminal Data.";

case 0x1001: return "INVALID ARG";
case 0x1002: return "FILE_OPEN_FAILED";
case 0x1003: return "FILE OPERATION_FAILED";
case 0x2001: return "MEMORY_NOT_ENOUGH";
case 0x3002: return "SMARTCARD_FAIL";
case 0x3003: return "SMARTCARD_INIT_FAILED";
case 0x3004: return "FALLBACK_SITUATION";
case 0x3005: return "SMARTCARD_ABSENT";
case 0x3006: return "SMARTCARD_TIMEOUT";
case 0x5001: return "EMV_PARSING_TAGS_FAILED";
case 0x5002: return "EMV_DUPLICATE_CARD_DATA_ELEMENT";
case 0x5003: return "EMV_DATA_FORMAT_INCORRECT";
case 0x5004: return "EMV_NO_TERM_APP";
case 0x5005: return "EMV_NO_MATCHING_APP";
case 0x5006: return "EMV_MISSING_MANDATORY_OBJECT";
case 0x5007: return "EMV_APP_SELECTION_RETRY";
case 0x5008: return "EMV_GET_AMOUNT_ERROR";
case 0x5009: return "EMV_CARD_REJECTED";
case 0x5010: return "EMV_AIP_NOT_RECEIVED";
case 0x5011: return "EMV_AFL_NOT_RECEIVED";
case 0x5012: return "EMV_AFL_LEN_OUT_OF_RANGE";
case 0x5013: return "EMV_SFI_OUT_OF_RANGE";
case 0x5014: return "EMV_AFL_INCORRECT";
case 0x5015: return "EMV_EXP_DATE_INCORRECT";
case 0x5016: return "EMV_EFF_DATE_INCORRECT";
case 0x5017: return "EMV_ISS_COD_TBL_OUT_OF_RANGE";
case 0x5018: return "EMV_CRYPTOGAM_TYPE_INCORRECT";
case 0x5019: return "EMV_PSE_NOT_SUPPORTED_BY_CARD";
case 0x5020: return "EMV_USER_SELECTED_LANGUAGE";
case 0x5021: return "EMV_SERVICE_NOT_ALLOWED";
case 0x5022: return "EMV_NO_TAG_FOUND";
case 0x5023: return "EMV_CARD_BLOCKED";
case 0x5024: return "EMV_LEN_INCORRECT";
case 0x5025: return "CARD_COM_ERROR";
case 0x5026: return "EMV_TSC_NOT_INCREASED";
case 0x5027: return "EMV_HASH_INCORRECT";
case 0x5028: return "EMV_NO_ARC";
case 0x5029: return "EMV_INVALID_ARC";
case 0x5030: return "If EMV transaction, terminated because no online comm. Otherwise
Firmware error Timeout Workstate 1";
case 0x5031: return "If EMV transaction, terminated because trans type incorrect. Otherwise
Firmware error Timeout Workstate 2";
case 0x5032: return "If EMV transaction, terminated because no app support. Otherwise
Firmware Data Error";
case 0x5034: return "If EMV transaction, terminated because Language not selected.
Otherwise Firmware error Application Version Error";
case 0x5035: return "If EMV transaction, terminated because Terminal Data Missing.
Otherwise Firmware Flash Error";
case 0x5036: return "If EMV transaction, terminated because Blocked AID encountered.
Otherwise Firmware Check value error";
case 0x5037: return "RETURN_CODE_FW_DEVICE_NAME_ERROR";
case 0x5038: return "RETURN_CODE_FW_ENCRYPTION_MODE_ERROR";
case 0x5039: return "RETURN_CODE_FW_FIRMWARE_ADDRESS_ERROR";
case 0x6001: return "CVM_TYPE_UNKNOWN";
case 0x6002: return "CVM_AIP_NOT_SUPPORTED";
case 0x6003: return "CVM_TAG_8E_MISSING";
case 0x6004: return "CVM_TAG_8E_FORMAT_ERROR";
case 0x6005: return "CVM_CODE_IS_NOT_SUPPORTED";
case 0x6006: return "CVM_COND_CODE_IS_NOT_SUPPORTED";
case 0x6007: return "NO_MORE_CVM";
case 0x6008: return "PIN_BYPASSED_BEFORE";
case 0x7001: return "PK_BUFFER_SIZE_TOO_BIG";
case 0x7002: return "PK_FILE_WRITE_ERROR";
case 0x7003: return "PK_HASH_ERROR";
//TTK SELF-CHECK
case 0x7500: return "FW Self-Test Failed(De - active Device)";
case 0x7501: return "TTK Self -Test - No MTK Key or No Key";
case 0x7502: return "TTK Self -Test - MTK Key Stop or Key Stop";
case 0x7503: return "TTK Self -Test - No EMV L2 Configuration Data";
case 0x7504: return "TTK Self -Test - EMV L2 Configuration Check Value Failed (De - active
Device)";
case 0x7505: return "TTK Self -Test - Future Key Check Value Failed(De - active Device)";
case 0x7506: return "TTK Self -Test - MTK Key Update Error";
case 0x7510: return "TTK Self-Test - MTK Key manage error";
case 0x7511: return "TTK Self-Test - No EMV L2 Configuration Data - Terminal Data";
case 0x7512: return "TTK Self-Test - No EMV L2 Configuration Data - Application Data";

```

```

case 0x7513: return "TTK Self-Test - No EMV L2 Configuration Data - CA Public Key";
case 0x7514: return "TTK Self-Test - No EMV L2 Configuration Data - CRL";
case 0x7515: return "TTK Self-Test - Error EMV L2 Configuration Data";
case 0x7516: return "TTK Self-Test - Future Key Check Value Failed(De - active Device)";

case 0x7FF0: return "Server returned a HTTP error condition on Authorize";
case 0x7FF1: return "Server returned a HTTP error condition on Step 1";
case 0x7FF2: return "Server returned a HTTP error condition on Step 2";
case 0x7FF3: return "Server returned a HTTP error condition on Step 3";
case 0x8001: return "Authorization: Cannot initialize RKI; no customer/ key information
found";

case 0x8101: return "Step 1: No key injection established";
case 0x8102: return "Step 1: Failed to encrypt challenge";
case 0x8103: return "Step 1: challenge length is incorrect";
case 0x8104: return "Step 1: Incorrect challenge data";
case 0x8105: return "Step 1: Response length incorrect";
case 0x8106: return "Step 1: Firmware responded NAK for Step 1";
case 0x8107: return "Step 1: Admin key not found for Step 1";
case 0x8201: return "Step 2: Customer key id could not be found in the DB";
case 0x8202: return "Step 2: Key Slot does not exist";
case 0x8203: return "Step 2: Could not get the future KSI from the server";
case 0x8204: return "Step 2: Could not get TR31 data block";
case 0x8205: return "Step 2: TR31 block length is incorrect";
case 0x8206: return "Step 2: Incorrect challenge data";
case 0x8207: return "Step 2: Firmware responded NAK for Step 2";
case 0x8301: return "Step 3: No key injection record found";
case 0x8302: return "Step 3: Remote Key Injection failed(NAK)";
case 0x8303: return "Step 3: Incorrect response form";
case 0x8304: return "Step 3: Firmware responded NAK for Step 3";

case 0x8002: return "GET_ONLINE_PIN";
case 0xD000: return "Data not exist";
case 0xD001: return "Data access error";
case 0xD100: return "RID not exist";
case 0xD101: return "RID existed";
case 0xD102: return "Index not exist";
case 0xD200: return "Maximum exceeded";
case 0xD201: return "Hash error";
case 0xD205: return "System Busy";
case 0xE001: return "Unable to go online";
case 0xE002: return "Technical Issue";
case 0xE003: return "Declined";
case 0xE004: return "Issuer Referral transaction";
case 0xF001: return "Decline the online transaction";
case 0xF002: return "Request to go online";
case 0xF003: return "Transaction is terminated";
case 0xF005: return "Application was not selected by kernel or ICC format error or ICC
missing data error";
case 0xF007: return "ICC didn't accept transaction";
case 0xF00A: return "Application may fallback to magstripe technology";
case 0xF00C: return "Transaction was cancelled";
case 0xF00D: return "Timeout";
case 0xF00F: return "Other EMV Error";
case 0xF010: return "Accept the offline transaction";
case 0xF011: return "Decline the offline transaction";
case 0xF021: return "ICC detected that the conditions of use are not satisfied";
case 0xF022: return "No app were found on card matching terminal configuration";
case 0xF023: return "Terminal file does not exist";
case 0xF024: return "CAPK file does not exist";
case 0xF025: return "CRL Entry does not exist";
case 0xFFE: return "Return code when blocking is disabled";
case 0xFFFF: return "Return code when command is not applicable on the selected device";
case 0xBBE0: return "CM100 Success";
case 0xBBE1: return "CM100 Parameter Error";
case 0xBBE2: return "CM100 Low Output Buffer";
case 0xBBE3: return "CM100 Card Not Found";
case 0xBBE4: return "CM100 Collision Card Exists";
case 0xBBE5: return "CM100 Too Many Cards Exist";
case 0xBBE6: return "CM100 Saved Data Does Not Exist";
case 0xBBE8: return "CM100 No Data Available";
case 0xBBE9: return "CM100 Invalid CID Returned";
case 0xBBEA: return "CM100 Invalid Card Exists";
case 0xBBEC: return "CM100 Command Unsupported";
case 0xBBED: return "CM100 Error In Command Process";
case 0xBBEE: return "CM100 Invalid Command";

case 0X9031: return "Unknown command";
case 0X9032: return "Wrong parameter (such as the length of the command is incorrect)";

case 0X9038: return "Wait (the command couldn't be finished in BWT)";
case 0X9039: return "Busy (a previously command has not been finished)";
case 0X903A: return "Number of retries over limit";

case 0X9040: return "Invalid Manufacturing system data";
case 0X9041: return "Not authenticated";
case 0X9042: return "Invalid Master DUKPT Key";
case 0X9043: return "Invalid MAC Key";

```

```

case 0X9044: return "Reserved for future use";
case 0X9045: return "Reserved for future use";
case 0X9046: return "Invalid DATA DUKPT Key";
case 0X9047: return "Invalid PIN Pairing DUKPT Key";
case 0X9048: return "Invalid DATA Pairing DUKPT Key";
case 0X9049: return "No nonce generated";
case 0X9949: return "No GUID available. Perform getVersion first.";
case 0X9950: return "MAC Calculation unsuccessful. Check BDK value.";

case 0X904A: return "Not ready";
case 0X904B: return "not MAC Data";

case 0X9050: return "Invalid Certificate";
case 0X9051: return "Duplicate key detected";
case 0X9052: return "AT checks failed";
case 0X9053: return "TR34 checks failed";
case 0X9054: return "TR31 checks failed";
case 0X9055: return "MAC checks failed";
case 0X9056: return "Firmware download failed";
case 0X9057: return "LCL - KEK exists";
case 0X9060: return "Log is full";
case 0X9061: return "Removal sensor unengaged";
case 0X9062: return "Any hardware problems";

case 0X9070: return "ICC communication timeout";
case 0X9071: return "ICC data error (such check sum error)";
case 0X9072: return "Smart Card not powered up";

case 0xF200: return "No AID or No Application Data";
case 0xF201: return "No Terminal Data";
case 0xF202: return "Wrong TLV format";
case 0xF203: return "AID list is full, maxim is 16";
case 0xF204: return "No any CA Key";
case 0xF205: return "No CA Key RID";
case 0xF206: return "No CA Key Index";
case 0xF207: return "CA Key list is full, maxim is 96";
case 0xF208: return "Wrong CA Key hash";
case 0xF209: return "Wrong Transaction Command Format";
case 0xF20A: return "Unexpected Command";
case 0xF20B: return "No CRL";
case 0xF20C: return "CRL list is full, maxim is 30";
case 0xF20D: return "No amount, other amount and transaction type in Transaction Command";
case 0xF20E: return "Wrong CA Hash and Encryption algorithm";
case 0X1C01: return "RKI could not retrieve device serial number";
case 0X1C02: return "RKI could not retrieve device firmware version";
case 0X1C03: return "RKI failed on Step 1: Authenticate a RKI with Server";
case 0X1C04: return "RKI failed on Step 2: Initiate RKI with Server";
case 0X1C05: return "RKI failed on Step 3: Request Key Pair";

case 0X0C57: return "Apple Pay VAS captured";
case 0x2400: return "Cannot retrieve device serial number in RKI update";
case 0x2401:
    return "Cannot retrieve device certificates in RKI update";
case 0x2402:
    return "Starting RKI Update Process";
case 0x2403:
    return "Cannot retrieve key device group in RKI update";
case 0x2404:
    return "PEDI Command failed in RKI Update";
case 0x2405:
    return "Invalid RSA Signature in RKI Update";
case 0x2406:
    return "PEDK Command failed in RKI Update";
case 0x2407:
    return "PEDK Command failed setting keys in RKI Update";
case 0x2408:
    return "PEDV Command failed in RKI Update";
case 0x2409:
    return "PEDV Command failed verifying keys in RKI Update";
case 0x240A:
    return "Completed RKI Update Process";
case 0x240B:
    return "Completed RKI Update Process. More keys available.";
case 0x240C:
    return "Error ending secure task.";
case 0x240D:
    return "Unknown Failure.";
case 0x240E:
    return "Failed on device key request, symmetric RKI.";
case 0x240F:
    return "Failed to get keys from the server, symmetric RKI.";
case 0x2410:
    return "Failed on device to set key, symmetric RKI.";
case 0x2411:
    return "Failed to validate key with server, symmetric RKI.";
case 0x2412:
    return "Error starting secure task.";

```

```

    }
    return "";
}

public static string getEMVAppErrorState(CEMV_APP_ERROR_STATE code)
{
    switch ((int)code)
    {
        case 0x00: return "CEMV_APP_ERROR_STATE_NONE";
        case 0x01: return "CEMV_APP_ERROR_STATE_SERIAL_COMM";
        case 0x02: return "CEMV_APP_ERROR_STATE_VALIDATE_COMMAND";
        case 0x03: return "CEMV_APP_ERROR_STATE_TRANSACTION";
        case 0x04: return "CEMV_APP_ERROR_STATE_CANDIDATE_SELECTION";
        case 0x05: return "CEMV_APP_ERROR_STATE_DISPLAY_APP_MENU";
        case 0x06: return "CEMV_APP_ERROR_STATE_LOAD_CONFIG_GROUP";
        case 0x07: return "CEMV_APP_ERROR_STATE_CARD_REMOVAL";
        case 0xff: return "CEMV_APP_ERROR_STATE_UNKNOWN";
    }
    return "";
}

public static string getEMVAppErrorFn(CEMV_APP_ERROR_FN code)
{
    switch ((int)code)
    {
        case 0x00: return "CEMV_APP_ERROR_FN_NONE";
        case 0x01: return "CEMV_APP_ERROR_FN_CEMV_INITIALIZE";
        case 0x02: return "CEMV_APP_ERROR_FN_CEMV_INITIATE_TRANSACTION";
        case 0x03: return "CEMV_APP_ERROR_FN_CEMV_BUILD_CANDIDATE_LIST";
        case 0x04: return "CEMV_APP_ERROR_FN_CEMV_FINAL_APPLICATION_SELECTION";
        case 0x05: return "CEMV_APP_ERROR_FN_CEMV_INITIATE_APPLICATION";
        case 0x06: return "CEMV_APP_ERROR_FN_CEMV_READ_APPLICATION_DATA";
        case 0x07: return "CEMV_APP_ERROR_FN_CEMV_OFFLINE_DATA_AUTHENTICATION";
        case 0x08: return "CEMV_APP_ERROR_FN_CEMV_PROCESS_RESTRICTIONS";
        case 0x09: return "CEMV_APP_ERROR_FN_CEMV_CARDHOLDER_VERIFICATION";
        case 0x0a: return "CEMV_APP_ERROR_FN_CEMV_CARDHOLDER_VERIFICATION_CONTINUE";
        case 0x0b: return "CEMV_APP_ERROR_FN_CEMV_TERMINAL_RISK_MANAGEMENT";
        case 0x0c: return "CEMV_APP_ERROR_FN_CEMV_TERMINAL_ACTION_ANALYSIS";
        case 0x0d: return "CEMV_APP_ERROR_FN_CEMV_CARD_ACTION_ANALYSIS";
        case 0x0e: return "CEMV_APP_ERROR_FN_CEMV_COMPLETION";
        case 0x0f: return "CEMV_APP_ERROR_FN_CEMV_TRANSACTION_MODE";
        case 0x10: return "CEMV_APP_ERROR_FN_CEMV_GET_APPLICATION_NAME";
        case 0x11: return "CEMV_APP_ERROR_FN_CEMV_SET_DATA_ELEMENTS";
        case 0x12: return "CEMV_APP_ERROR_FN_CEMV_GET_DATA_ELEMENTS";
        case 0x13: return "CEMV_APP_ERROR_FN_CEMV_UPDATE_EMV_KERNEL_CONFIG_FILE";
        case 0x20: return "CEMV_APP_ERROR_FN_UI_DISPLAY_CLEAR";
        case 0x21: return "CEMV_APP_ERROR_FN_UI_DISPLAY_TEXT";
        case 0x22: return "CEMV_APP_ERROR_FN_UI_DISPLAY_BUTTON";
        case 0x23: return "CEMV_APP_ERROR_FN_UI_LIST_CREATE";
        case 0x24: return "CEMV_APP_ERROR_FN_UI_LIST_ADD_ITEM";
        case 0x25: return "CEMV_APP_ERROR_FN_UI_LIST_READ_BUTTON";
        case 0x26: return "CEMV_APP_ERROR_FN_UI_LIST_GET_SELECTED_ITEM";
        case 0x30: return "CEMV_APP_ERROR_FN_PCI_START_CUSTOM_DISPLAY_MODE";
    }
    return "";
}

public static string getExtendedStatusCode(EXTENDED_STATUS_CODES code)
{
    switch ((int)code)
    {
        case 0x000A0000: return "Error creating AppMgr Client";
        case 0x000A0001: return "Invalid client type for this operation";
        case 0x000A0002: return "Service is not available";
        case 0x000A0003: return "Received bad response from Daemon";
        case 0x000A0004: return "Command was canceled";
        case 0x000A0005: return "Command was Deferred";
        case 0x000A0006: return "Module failed to load";
        case 0x000A0007: return "Module missing Init interface";
        case 0x000A0008: return "Module missing Exit interface";
        case 0x000A0009: return "Module missing Install interface";
        case 0x000A000A: return "Module missing Process interface";
        case 0x000A000B: return "Failed to install module";
        case 0x000A000C: return "Protocol violation";
        case 0x000A000D: return "Bad number of AIDs for the module";
        case 0x000A000E: return "No EPS module for the CL card";
        case 0x000A000F: return "No AID module for the application selected";
        case 0x000A0010: return "No MIFARE module for the CL card";
        case 0x000A0011: return "Failed to authenticate MIFARE sector";
        case 0x000A0012: return "Failed to read MIFARE block";
        case 0x000A0013: return "Fallback from contactless interface to magstripe";
        case 0x000A0014: return "Fallback from contactless interface to contact";
        case 0x000A0015: return "Fallback from contactless interface to other";
        case 0x000A0016: return "Fallback from contactless interface to Mifare";
        case 0x000A0018: return "Fallback from AID Module to Mifare to process specified MIDs";
    }
}

```

```

    case 0x000A0019: return "ERR_APPMGR_CL_DESFIRE_RAW_DATA_SUCCESS";
    case 0x000A001A: return "ERR_APPMGR_CL_VIVOCOMM_RAW_DATA_SUCCESS";
    case 0x000A001B: return "ERR_APPMGR_CL_VIVOCOMM_RAW_WRITE_SUCCESS";
    case 0x000A001C: return "Bad number of AIDs for the module";
    case 0x000A001D: return "No module for the CT Application Selection";
    case 0x000A001E: return "No AID Module to handle selected AID";
    case 0x000A001F: return "Fallback from contact interface to magstripe";
    case 0x000A0020: return "No module registered for MSR cards";
    case 0x000A0021: return "Module platform and system platform mismatch";
    case 0x000A0022: return "ERR_APPMGR_DECLINE";
    case 0x000A0023: return "ERR_APPMGR_NO_ADVICE_OR_REVERSAL_REQUIRED";
    case 0x000A0024: return "ERR_APPMGR_ADVICE_REQUIRED";
    case 0x000A0025: return "ERR_APPMGR_REVERSAL_REQUIRED";
    case 0x000A0026: return "ERR_APPMGR_ADVICE_AND_REVERSAL_REQUIRED";
    case 0x000A0027: return "ERR_APPMGR_APP_SELECTION_RETRY";
    case 0x000A0028: return "ERR_APPMGR_REQUEST_ONLINE_AUTH";
    case 0x000A0029: return "ERR_APPMGR_CL_REQUEST_ONLINE_AUTH";
    case 0x000A002A: return "ERR_APPMGR_REQUEST_ONLINE_PIN";
    case 0x000A002B: return "ERR_APPMGR_REQUEST_SIGNATURE";
    case 0x60000: return "An EMV Level 1 Error Occurred while accessing the Card";
    case 0x60001: return "There is no card inserted in the reader";
    case 0x60002: return "EMVD Response was incorrect";
    case 0x60003: return "EMV Kernel Returned an unknown Error (or an error that has not been
mapped to a CEMV Error yet)";
    case 0x60004: return "Card rejected due to application level error";
    case 0x60005: return "Card Failed to respond in given time";
    case 0x60006: return "The card is blocked";
    case 0x60007: return "Service not allowed by Card";
    case 0x60008: return "Smartcard transaction failed but application can fallback to
MagStripe";
    case 0x60009: return "Mal-formatted TLV Data";
    case 0x0006000A: return "Length of one or more Tags requested via the CEMV_GetDataElements";
;
    case 0x0006000B: return "Data read from card is not formatted correctly";
    case 0x0006000C: return "Data read from card is missing one or more mandatory data items";
    case 0x0006000D: return "Duplicate Data Element(s) read from Card";
    case 0x0006000E: return "CEMV_STATUS_PSE_NOT_SUPPORTED_BY_CARD";
    case 0x0006000F: return "There are no applications on the card";
    case 0x60010: return "No matching application between terminal and card";
    case 0x60011: return "Application list returned in the selection of application has a
wrong data format";
    case 0x60012: return "Application Name was not Found";
    case 0x60013: return "Issuer code table index received from card is not EMV compliant";
    case 0x60014: return "Card didn't return AIP parameter, which is mandatory in Initiate
Application Processing step";
    case 0x60015: return "Card didn't return AFL parameter, which is mandatory in Initiate
Application Processing step";
    case 0x60016: return "AFL parameter length is wrong";
    case 0x60017: return "AFL parameter is wrong";
    case 0x60018: return "Expiry date format read from the card is incorrect";
    case 0x60019: return "Effective date format read from the card is incorrect";
    case 0x0006001A: return "SFI (Short File Identifier) returned by card has a wrong value";
    case 0x0006001B: return "Short file identifier of the record which is read is not correct
Note: This seems very similar to the CEMV_STATUS_SFI_NOT_CORRECT error code and it seems that it is spurious.
It is being included for now since the EMV Kernel Specs define them as two separate error codes.";
    case 0x0006001C: return "SFI Out of Range";
    case 0x0006001D: return "CA Public Key not found for Data Authentication";
    case 0x0006001E: return "CA Public Key failed hash check. Key data is not valid.";
    case 0x0006001F: return "CA Public Key Hash Algorithm Indicator has an invalid value.";
    case 0x60020: return "CA Public Key Algorithm Indicator has an invalid value.";
    case 0x60021: return "CA Public Key Modulus length is invalid.";
    case 0x60022: return "CA Public Key Exponent has an invalid value.";
    case 0x60023: return "CA Public Key already exists.";
    case 0x60024: return "There is no more space to store the CA Public Key.";
    case 0x60025: return "CA Public Key data is bad or corrupt.";
    case 0x60026: return "A required terminal parameter is missing";
    case 0x60027: return "Length of CVM List is not correct";
    case 0x60028: return "Cryptogram types requested by terminal and returned by card in
EMV_CardActionAnalysis and EMV_Completion functions are inconsistent";
    case 0x60029: return "One or More Tags were not found";
    case 0x0006002A: return " Amount parameter format is wrong";
    case 0x0006002B: return "CEMV_STATUS_USER_SELECTED_LANGUAGE";
    case 0x0006002C: return "CEMV_STATUS_REQUEST_ONLINE_PIN";
    case 0x0006002D: return "CEMV_STATUS_REQUEST_SIGNATURE";
    case 0x0006002E: return "Application Selection Retry (for example if GPO returns SW=6985)";
    case 0x0006002F: return "CEMV_STATUS_TSC_NOT_INCREASED";
    case 0x60030: return "Authorization Response Code is missing";
    case 0x60031: return "Authorization Response Code is invalid";
    case 0x60032: return "EMV Kernel returned No Online Comm Error";
    case 0x60033: return "CEMV_STATUS_CVM_TYPE_UNKNOWN";
    case 0x60034: return "CEMV_STATUS_CVM_AIP_NOT_SUPPORTED";
    case 0x60035: return "CEMV_STATUS_CVM_LIST_MISSING";
    case 0x60036: return "CEMV_STATUS_CVM_LIST_FORMAT_ERROR";
    case 0x60037: return "CEMV_STATUS_CVM_CODE_NOT_SUPPORTED";
    case 0x60038: return "CEMV_STATUS_CVM_COND_CODE_NOT_SUPPORTED";
    case 0x60039: return "CEMV_STATUS_CVM_NO_MORE_CVM";
    case 0x0006003A: return "CEMV_STATUS_CVM_PIN_BYPASSED_BEFOFRE";

```



```

case 0x70000: return "Error creating CFG Client";
case 0x70001: return "CFG Service is not available";
case 0x70002: return "Received bad response from Daemon";
case 0x1: return "General failure for the procedure";
case 0x2: return "Out of memory";
case 0x3: return "Bad parameter (s) supplied";
case 0x4: return "Procedure is not supported";
case 0x5: return "Item was not found";
case 0x6: return "Item already exists";
case 0x7: return "Out of bounds error";
case 0x8: return "Not authorized";
case 0x9: return "Q error";
case 0x0000000A: return "Timeout";
case 0x0000000B: return "Device not found";
case 0x0000000C: return "File open error";
case 0x0000000D: return "File operation error";
case 0x0000000E: return "Operation canceled";
case 0x0000000F: return "Resource or device is busy";
case 0x10: return "Specified directory not found";
case 0x11: return "Specified file is too big";
case 0x12: return "Not enough memory to complete command";
case 0x50000: return "ICCD Response was Incorrect";
case 0x50001: return "The slot is already in use for a complex command such as polling or
Card Removal";
case 0x50002: return "The driver resources are dedicated for processing a command on
another slot.";
case 0x50003: return "Error - Card was not present";
case 0x50004: return "The card is not ready for application level transactions since a TTL
session has not been established with the card yet";
case 0x50005: return "Level 1 error occurred. Card was deactivated by the ICC Service or
the smart card driver";
case 0x50006: return "Level 1 error occurred. Card was not deactivated by the ICC Service
or the smart card driver. Decision to deactivate card left up to the client application";
case 0x50007: return "The card failed to respond to the APDU command";
case 0x50008: return "Card is present but ATR data not yet available. Wait until Card is
Ready before calling this function.";
case 0x50009: return "ATR was returned but the Mode is not known. This condition will occur
if a non-standard ATR is returned by the card.";
case 0x0005000A: return "Command not allowed (Command used out of sequence)";
case 0x90000: return "MSRD Response was Incorrect";
case 0x90001: return "A card was swiped but an error occurred";
case 0x90002: return "Command not allowed (Command used out of sequence)";
case 0x10000: return "Error creating PCI Client";
case 0x10001: return "PCI Service is not available";
case 0x10002: return "Received bad response from Daemon";
case 0x10003: return "No key was found for PIN request";
case 0x10004: return "Key Generation (DUKPT) has reached its end of life. Cease unit
operation.";
case 0x10005: return "PIN entry cancelled";
case 0x10006: return "Key is incorrectly formatted";
case 0x10007: return "Random number generator device failure";
case 0x10008: return "Illegal char passed in place of a number";
case 0x10009: return "Illegal currency character given in command string";
case 0x0001000A: return "Command was cancelled via signal";
case 0x0001000B: return "Intrusion was detected on the keypad";
case 0x0001000C: return "Attempt to read keypad while in non-secure display mode";
case 0x0001000D: return "No data to return";
case 0x40000: return "Error creating PICC Client";
case 0x40001: return "PICC Service is not available";
case 0x40002: return "Received bad response from Daemon";
case 0x40003: return "Command was cancelled via signal";
case 0x40004: return "ERR_PICC_POLL_TIMEOUT";
case 0x40005: return "ERR_PICC_CARD_NOT_FOUND";
case 0x40006: return "ERR_PICC_COLLISION";
case 0x40007: return "ERR_PICC_TYPE_NOT_SUPPORTED";
case 0x40008: return "ERR_PICC_CRC_ERR";
case 0x40009: return "ERR_PICC_PARITY_ERR";
case 0x0004000A: return "ERR_PICC_FRAMING_ERR";
case 0x0004000B: return "ERR_PICC_BIT_COUNT_ERR";
case 0x0004000C: return "ERR_PICC_TRANSMISSION_ERR";
case 0x0004000D: return "ERR_PICC_BYTE_COUNT_ERR";
case 0x0004000E: return "ERR_PICC_AUTHENTICATION_FAILED";
case 0x0004000F: return "ERR_PICC_ACCESS_TIMEOUT";
case 0x40010: return "ERR_PICC_READ_ERR";
case 0x40011: return "ERR_PICC_WRITE_ERR";
case 0x40012: return "ERR_PICC_NOT_AUTHORIZED";
case 0x40013: return "ERR_PICC_RETURN_CODE_ERR";
case 0x40014: return "ERR_PICC_VALUE_ERR";
case 0x40015: return "ERR_PICC_KEY_ERR";
case 0x40016: return "ERR_PICC_FIFO_OVERFLOW";
case 0x40017: return "ERR_PICC_TX_BUFFER_OVERFLOW";
case 0x40018: return "ERR_PICC_RX_BUFFER_OVERFLOW";
case 0x40019: return "ERR_PICC_DATA_CODING_ERR";
case 0x0004001A: return "ERR_PICC_GEN_PROTOCOL_ERR";
case 0x0004001B: return "ERR_PICC_EXCEEDED_RETRIES";
case 0x0004001C: return "ERR_PICC_BLOCK_PROTOCOL_ERR";
case 0x0004001D: return "ERR_PICC_UNK_COMMAND";

```

```

    case 0x0004001E: return "ERR_PICC_COLLISION_INTERPRETED";
    case 0x0004001F: return "ERR_PICC_TIMEOUT_WITH_ERR";
    case 0x40020: return "ERR_PICC_ALREADY_POLLING";
    case 0x40021: return "ERR_PICC_CARD_NOT_REMOVED";
    case 0x00040022: return "ERR_PICC_CODE_END";
    case 0x000B0000: return "SoftSAM Response was Incorrect";
    case 0x000B0001: return "ERR_SOFTSAM_CA_KEY_NOT_FOUND";
    case 0x000B0002: return "ERR_SOFTSAM_BAD_MODULUS_LENGTH";
    case 0x000B0003: return "ERR_SOFTSAM_BAD_EXPONENT_LENGTH";
    case 0x000B0004: return "ERR_SOFTSAM_BAD_CERTIFICATE_LENGTH";
    case 0x000B0005: return "ERR_SOFTSAM_MALFORMATED_CERTIFICATE";
    case 0x000B0006: return "ERR_SOFTSAM_RECOVERY_FN_FAILED";
    case 0x000B0007: return "ERR_SOFTSAM_BUFFER_OVF";
    case 0x000B0008: return "ERR_SOFTSAM_ALG_NOT_SUPPORTED";
    case 0x000B0009: return "ERR_SOFTSAM_HASH_ALG_NOT_SUPPORTED";
    case 0x000B000A: return "ERR_SOFTSAM_CRYPT_ALG_NOT_SUPPORTED";
    case 0x000B000B: return "ERR_SOFTSAM_HASH_FN_FAILED";
    case 0x000B000C: return "ERR_SOFTSAM_RID_KEY_SLOTS_FULL";
    case 0x000B000D: return "ERR_SOFTSAM_KEY_ALREADY_EXISTS";
    case 0x000B000E: return "ERR_SOFTSAM_NO_SPACE_FOR_NEW_RID";
    case 0x000B000F: return "ERR_SOFTSAM_NO_FREE_KEY_SLOTS";
    case 0x000B0010: return "ERR_SOFTSAM_BAD_PKT_DATA_LEN";
    case 0x000B0011: return "ERR_SOFTSAM_BAD_PACKET_NUMBERING";
    case 0x000B0012: return "ERR_SOFTSAM_INVALID_CRYPTO_ALGORITHM";
    case 0x000B0013: return "ERR_SOFTSAM_INVALID_CRYPTO_CMD";
    case 0x000B0014: return "ERR_SOFTSAM_INVALID_CRYPTO_KEYTYPE";
    case 0x000B0015: return "ERR_SOFTSAM_INVALID_CRYPTO_DATA_LENGTH";
    case 0x000B0016: return "ERR_SOFTSAM_RESP_EMV_FAILURE";
    case 0x000B0017: return "ERR_SOFTSAM_RESP_EMV_BAD_PKT_DATA_LEN";
    case 0x000B0018: return "ERR_SOFTSAM_RESP_EMV_CA_KEY_NOT_FOUND";
    case 0x000B0019: return "ERR_SOFTSAM_RESP_EMV_BAD_CERTIFICATE_LENGTH";
    case 0x000B001A: return "ERR_SOFTSAM_RESP_EMV_RECOVERY_FN_FAILED";
    case 0x000B001B: return "ERR_SOFTSAM_RESP_EMV_MALFORMATED_CERTIFICATE";
    case 0x000B001C: return "ERR_SOFTSAM_RESP_EMV_HASH_ALG_NOT_SUPPORTED";
    case 0x000B001D: return "ERR_SOFTSAM_RESP_EMV_HASH_FN_FAILED";
    case 0x000B001E: return "ERR_SOFTSAM_RESP_EMV_CRYPT_ALG_NOT_SUPPORTED";
    case 0x000B001F: return "ERR_SOFTSAM_RESP_EMV_BAD_EXPONENT_LENGTH";
    case 0x000B0020: return "ERR_SOFTSAM_UNKNOWN_ERROR";
    case 0x20000: return "TLV Tag provided was badly formatted";
    case 0x20001: return "TLV Length was badly formatted";
    case 0x20002: return "TLV tag was found under an incorrect constructed tag.";
    case 0x20003: return "TLV was badly formatted. This error is usually thrown when some
bytes are left in the end that did not get processed. Check if SW word is also being passed in.";
    case 0x20004: return "Non-conforming tag callback has returned illegal value for the tag
lengths";
    case 0x30000: return "Error creating UI Client";
    case 0x30001: return "UI Service is not available";
    case 0x30002: return "Received bad response from Daemon";
    case 0x30003: return "Buffer isn't large enough to hold signature";
    case 0x30004: return "Font not supported (either font size or the font)";
    case 0x30005: return "No touchscreen buttons currently defined";
    case 0x30006: return "Command cancelled by signal";
    case 0x30007: return "Text areas overlap on screen - not allowed";
    case 0x30008: return "Text areas overlap on screen - not allowed";
    case 0x30009: return "Line item areas overlap on screen - not allowed";
    case 0x0003000A: return "Checkbox areas overlap on screen - not allowed";
    case 0x0003000B: return "No touchscreen checkboxes currently defined";
    case 0x0003000C: return "Signature capture area overlaps";
    case 0x0003000D: return "New screen item overlaps existing input field";
    case 0x0003000E: return "Request for read with zero timeout and no data";
    case 0x0003000F: return "Exceeded limit for allowed items on the screen";
    case 0x30010: return "Text areas overlap on screen - not allowed";
    case 0x30011: return "No touch sensitive areas are defined";
    case 0x30012: return "Command requires custom mode";
    case 0x30013: return "Slideshow is currently in progress";
    case 0x30014: return "Touchscreen hardware not present";
    case 0x30015: return "No graphic display hardware present";
    case 0x30016: return "Rectangle areas overlap on screen display";
    case 0x00030017: return "Paragraph areas overlap on screen - not allowed";
    case 0x00030018: return "A word in a paragraph is too wide for the rectangle";
    case 0x00030019: return "Paragraph areas violate the minimum dimensions";
    case 0x0003001A: return "Get PAN is cancelled";
    case 0x00080000: return "Given buffer won't contain a record";
    case 0x00080001: return "File size is not a multiple of record size";
    case 0x00080002: return "No records in file";
    case 0x00080003: return "No records matched the search criteria";
    case 0x00080004: return "Length of input record does not match record size";
    case 0x00080005: return "Failed to read expected record length";
    case 0x00080006: return "Record already exists";
    case 0x00080007: return "Failed to write expected number of bytes";

    }
    return "";
}

public static string getRFState(RF_STATE code)
{

```



```

switch ((int)code)
{
    case 0x00: return "RF State Code not available";
    case 0x01: return "Error occurred during PPSE command";
    case 0x02: return "Error occurred during SELECT command";
    case 0x03: return "Error occurred during GET PROCESSING OPTIONS command";
    case 0x04: return "Error occurred during READ RECORD command";
    case 0x05: return "Error occurred during GEN AC command";
    case 0x06: return "Error occurred during CCC command";
    case 0x07: return "Error occurred during IA command";
    case 0x08: return "Error occurred during SDA processing";
    case 0x09: return "Error occurred during DDA processing";
    case 0x0A: return "Error occurred during CDA processing";
    case 0x0B: return "Error occurred during TAA processing";
    case 0x0C: return "Error occurred during GET DATA command";
    case 0x21: return "Error occurred during CARD READ COMPLETE processing";
    case 0x22: return "Error occurred during PROCESSING RESTRICTIONS processing";
    case 0x23: return "Error occurred during ODA processing";
    case 0x24: return "Error occurred during CARDHOLDER VERIFICATION processing";
    case 0x25: return "Error occurred during ONLINE PROCESSING processing";
    case 0x26: return "Error occurred during COMPLETION processing";
}
return "";
}

public static string getTransError(TRANS_ERROR_CODE code)
{
    switch ((int)code)
    {
        case 0x01: return "Out of Sequence Command";
        case 0x02: return "Go to Contact Interface";
        case 0x03: return "Transaction Amount is Zero";
        case 0x04: return "Go to Other Interface";
        case 0x05: return "Go to Nearby Contact Interface";
        case 0x06: return "Go to Magstripe Interface";
        case 0x20: return "Card returned Error Status";
        case 0x21: return "Collision Error";
        case 0x22: return "Amount Over Maximum Limit";
        case 0x23: return "Request Online Authorization";
        case 0x25: return "Card Blocked";
        case 0x26: return "Card Expired";
        case 0x27: return "Unsupported Card";
        case 0x30: return "Card did not respond";
        case 0x40: return "Unknown Data Element";
        case 0x41: return "Required Data Element(s) Missing";
        case 0x42: return "Card Generated AAC";
        case 0x43: return "Card Generated ARQC";
        case 0x44: return "SDA/DDA Failed (Not Supported by Card)";
        case 0x50: return "SDA/DDA/CDDA Failed (CA Public Key)";
        case 0x51: return "SDA/DDA/CDDA Failed (Issuer Public Key)";
        case 0x52: return "SDA Failed (SSAD)";
        case 0x53: return "DDA/CDDA Failed (ICC Public Key)";
        case 0x54: return "DDA/CDDA Failed (Dynamic Signature Verification)";
        case 0x55: return "Processing Restrictions Failed";
        case 0x56: return "Terminal Risk Management (TRM) Failed";
        case 0x57: return "Cardholder Verification Failed";
        case 0x58: return "Terminal Action Analysis (TAA) Failed";
        case 0x61: return "SD Memory Error";
        case 0x70: return "Contact EMV Error";
        case 0x80: return "No Merchants";
        case 0x81: return "TLV Parse Error";
        case 0x82: return "Merchant Data Error";
        case 0x83: return "System Memory Error";
        case 0x84: return "Application Skip Error";
        case 0x85: return "Application Version Error";
    }
    return "";
}

```

Chapter 15

Enumeration Reference

Common

```

public enum EVENT_TRANSACTION_DATA_Types
{
    EVENT_TRANSACTION_DATA_UNKNOWN, EVENT_TRANSACTION_DATA_CARD_DATA, EVENT_TRANSACTION_DATA_EMV_DATA,
    EVENT_TRANSACTION_DATA_MSR_CANCEL_KEY, EVENT_TRANSACTION_DATA_MSR_BACKSPACE_KEY,
    EVENT_TRANSACTION_DATA_MSR_ENTER_KEY, EVENT_TRANSACTION_DATA_MSR_DATA_ERROR, EVENT_TRANSACTION_PIN_DATA
}

public enum CAPTURE_ENCODE_TYPE
{
    CAPTURE_ENCODE_TYPE_ISOABA, CAPTURE_ENCODE_TYPE_AAMVA, CAPTURE_ENCODE_TYPE_Other,
    CAPTURE_ENCODE_TYPE_Raw, CAPTURE_ENCODE_TYPE_JisI_II
}

public enum CAPTURE_ENCRYPT_TYPE
{
    CAPTURE_ENCRYPT_TYPE_TDES, CAPTURE_ENCRYPT_TYPE_AES, CAPTURE_ENCRYPT_TYPE_NONE
}

public enum EMV_ENCRYPTION_MODE
{
    EMV_ENCRYPTION_MODE_TDES = 0, EMV_ENCRYPTION_MODE_AES = 1
}

public enum EMV_ENCRYPTION_MODE
{
    EMV_ENCRYPTION_MODE_TDES = 0, EMV_ENCRYPTION_MODE_AES = 1
}

public enum EMV_LCD_DISPLAY_MODE
{
    EMV_LCD_DISPLAY_MODE_CANCEL = 0, EMV_LCD_DISPLAY_MODE_MENU = 1, EMV_LCD_DISPLAY_MODE_PROMPT = 2,
    EMV_LCD_DISPLAY_MODE_MESSAGE = 3, EMV_LCD_DISPLAY_MODE_LANGUAGE_SELECT = 8, EMV_LCD_DISPLAY_MODE_CLEAR_SCREEN = 16
}

public enum EMV_RESULT_CODE
{
    EMV_RESULT_CODE_APPROVED_OFFLINE = 0,
    EMV_RESULT_CODE_DECLINED_OFFLINE = 1,
    EMV_RESULT_CODE_APPROVED = 2,
    EMV_RESULT_CODE_DECLINED = 3,
    EMV_RESULT_CODE_GO_ONLINE = 4,
    EMV_RESULT_CODE_CALL_YOUR_BANK = 5,
    EMV_RESULT_CODE_NOT_ACCEPTED = 6,
    EMV_RESULT_CODE_FALLBACK_TO_MSR = 7,
    EMV_RESULT_CODE_TIMEOUT = 8,
    EMV_RESULT_CODE_GO_ONLINE_CTLS = 9,
    EMV_RESULT_CODE_AUTHENTICATE_TRANSACTION = 0x0010,
    EMV_RESULT_CODE_SWIPE_NON_ICC = 17,
    EMV_RESULT_CODE_CTLS_TWO_CARDS = 0x7A,
    EMV_RESULT_CODE_CTLS_TERMINATE = 0x7E,
    EMV_RESULT_CODE_CTLS_TERMINATE_TRY_ANOTHER = 0x7D,
    EMV_RESULT_CODE_UNABLE_TO_REACH_HOST
}

```


Chapter 16

EMV Callback

During an EMV transaction, without a built-in LCD display on the NEO2, LCD Display messages will be returned as an EMV Callback.

In the MessageCallback for the SDK, there is a DeviceState EMVCallback. When this DeviceState is received, a `IDTechSDK::EMV_Callback` class object will be returned. `IDTechSDK::EMV_Callback::callbackType` will specify the type of callback: `EMV_CALLBACK_TYPE_LCD`, `EMV_CALLBACK_TYPE_PINPAD`, `EMV_CALLBACK_MSR`. The NEO2 will not utilize the `EMV_CALLBACK_TYPE_PINPAD`.

If NEO2 terminal settings specify MSR is part of the configuration, for cases of fallback, the type of callback will be `EMV_CALLBACK_MSR`. When this is received, MSR data must be collected and returned to `IDTechSDK::IDT_NEO2::emv_callbackResponseMSR()` to complete the transaction.

For LCD display messages, the callback type will be `EMV_CALLBACK_TYPE_LCD`. To evaluate what kind of LCD message, you get `EMV_LCD_DISPLAY_MODE` from `IDTechSDK::EMV_Callback::lcd_displayMode`: 1- `LCD_DISPLAY_MODE_MENU`: Menu selection, response required with selected menu index #, or 0 to cancel 2- `LCD_DISPLAY_MODE_PROMPT`: Message Prompt, response required 'E' for Enter/Accept, or 'C' for cancel 3- `LCD_DISPLAY_MODE_MESSAGE`: Display Message, no response required 8 – `LCD_DISPLAY_MODE_LANGUAGE_SELECT`: Language selection, response required with selected language index # 16 - `LCD_DISPLAY_MODE_CLEAR_SCREEN`: Request to clear LCD screen of information

If the mode is `LCD_DISPLAY_MODE_MESSAGE` or `LCD_DISPLAY_MODE_CLEAR_SCREEN`, these do not pause the EMV transaction. These two modes are for displaying a message (no response required), or for clearing the screen.

If the mode is `LCD_DISPLAY_MODE_MENU`, `LCD_DISPLAY_MODE_PROMPT`, or `LCD_DISPLAY_MODE_LANGUAGE_SELECT`, the provided message must be displayed, and then the EMV transaction pauses until a response is sent to `IDTechSDK::IDT_NEO2::emv_callbackResponseLCD()`.

The message to display is `byte[] lcd_messages`. This contains either Message String, or a Message ID according to LCD Foreign Language Mapping Table ([Foreign Language Mapping Table](#)).

Chapter 17

EMV Tag Reference

Tag	Description
42	Issuer Identification Number (IIN)
4F	Application Identifier (ADF Name)
50	Application Label
52	Command to perform
56	Track 1 Data
57	Track 2 Equivalent Data
5A	Application Primary Account Number (PAN)
5D	Deleted (see 9D)
5F20	Cardholder Name
5F24	Application Expiration Date
5F28	Issuer Country Code
5F2A	Transaction Currency Code (Default: 08 40)
5F2D	Language Preference
5F30	Service Code
5F34	Application Primary Account Number (PAN) Sequence Number (PSN)
5F36	Transaction Currency Exponent
5F3C	Transaction Reference Currency Code
5F3D	Transaction Reference Currency Exponent
5F50	Issuer URL
5F53	International Bank Account Number (IBAN)
5F54	Bank Identifier Code (BIC)
5F55	Issuer Country Code (alpha2 format)
5F56	Issuer Country Code (alpha3 format)
5F57	Account Type Selection
6F	File Control Information (FCI) Template
61	Application Template
62	File Control Parameters (FCP) Template
70	READ RECORD Response Message Template
71	Issuer Script Template 1
72	Issuer Script Template 2
73	Directory Discretionary Template
77	Response Message Template Format 2
80	Response Message Template Format 1
81	Amount, Authorised (Binary)
82	Application Interchange Profile (AIP)

Tag	Description
83	Command Template
84	Dedicated File (DF) Name
86	Issuer Script Command
87	Application Priority Indicator
88	Short File Identifier (SFI)
89	Authorisation Code
8A	Authorization Response Code
8A	Authorisation Response Code (ARC)
8C	Card Risk Management Data Object List 1 (CDOL1)
8D	Card Risk Management Data Object List 2 (CDOL2)
8E	Cardholder Verification Method (CVM) List
8F	Certification Authority Public Key Index (PKI)
90	Issuer Public Key Certificate
91	Issuer Authentication Data
92	Issuer Public Key Remainder
93	Signed Application Data
94	Application File Locator (AFL)
95	Terminal Verification Results (TVR)
97	Transaction Certificate Data Object List (TDOL)
98	Transaction Certificate (TC) Hash Value
99	Transaction Personal Identification Number (PIN) Data
99	Transaction Personal Identification Number (PIN) Data
98	Transaction Certificate (TC) Hash Value
9A	Transaction Date (YYMMDD)
9A	Transaction Date
9B	Transaction Status Information
9B	Transaction Status Information
9C	Transaction Type
9C	Transaction Type
9D	Directory Definition File (DDF) Name
9F01	Acquirer Identifier
9F02	Amount, Authorized (Numeric)
9F03	Amount, Other (Numeric)
9F04	Amount, Other (Binary)
9F05	Application Discretionary Data
9F06	Application Identifier (AID) – terminal
9F07	Application Usage Control (AUC)
9F08	Application Version Number
9F09	Application Version Number (Default: 00 02)
9F0B	Cardholder Name Extended
9F0D	Issuer Action Code - Default
9F0E	Issuer Action Code - Denial
9F0F	Issuer Action Code - Online
9F10	Issuer Application Data (IAD)
9F11	Issuer Code Table Index
9F12	Application Preferred Name
9F13	Last Online Application Transaction Counter (ATC) Register
9F14	Lower Consecutive Offline Limit
9F15	Merchant Category Code

Tag	Description
9F16	Merchant Identifier
9F17	Personal Identification Number (PIN) Try Counter
9F18	Issuer Script Identifier
9F19	Deleted (see 9F49)
9F1A	Terminal Country Code
9F1B	Terminal Floor Limit
9F1C	Terminal Identification
9F1D	Terminal Risk Management Data
9F1E	Interface Device (IFD) Serial Number
9F1F	Track 1 Discretionary Data
9F20	Track 2 Discretionary Data
9F21	Transaction Time (HHMMSS)
9F22	Certification Authority Public Key Index
9F23	Upper Consecutive Offline Limit
9F26	Application Cryptogram (AC)
9F27	Cryptogram Information Data (CID)
9F29	Extended Selection
9F2A	Kernel Identifier
9F2D	Integrated Circuit Card (ICC) PIN Encipherment Public Key Certificate
9F2E	Integrated Circuit Card (ICC) PIN Encipherment Public Key Exponent
9F2F	Integrated Circuit Card (ICC) PIN Encipherment Public Key Remainder
9F32	Issuer Public Key Exponent
9F33	Terminal Capabilities (see below)
9F34	Cardholder Verification Method (CVM) Results
9F35	Terminal Type (see below)
9F36	Application Transaction Counter (ATC)
9F37	Unpredictable Number
9F38	Processing Options Data Object List (PDOL)
9F39	POS Entry Mode (Default: 07)
9F3A	Amount, Reference Currency
9F3B	Application Reference Currency
9F3C	Transaction Reference Currency Code
9F3D	Transaction Reference Currency Exponent
9F40	Additional Terminal Capabilities (see below)
9F41	Transaction Sequence Counter
9F42	Application Currency Code
9F43	Application Reference Currency Exponent
9F44	Application Currency Exponent
9F45	Data Authentication Code
9F46	Integrated Circuit Card (ICC) Public Key Certificate
9F47	Integrated Circuit Card (ICC) Public Key Exponent
9F48	Integrated Circuit Card (ICC) Public Key Remainder
9F49	Dynamic Data Authentication Data Object List (DDOL)
9F4A	Static Data Authentication Tag List (SDA)
9F4B	Signed Dynamic Application Data (SDAD)
9F4C	ICC Dynamic Number
9F4D	Log Entry
9F4E	Merchant Name and Location
9F4E	Merchant Name and Location

Tag	Description
9F4F	Log Format
9F50	Offline Accumulator Balance
9F51	Application Currency Code
9F52	Application Default Action (ADA)
9F53	Transaction Category Code
9F54	DS ODS Card
9F55	Geographic Indicator
9F56	Issuer Authentication Indicator
9F57	Issuer Country Code
9F58	Consecutive Transaction Counter Limit (CTCL)
9F59	Consecutive Transaction Counter Upper Limit (CTCUL)
9F5A	Application Program Identifier (Program ID)
9F5B	Issuer Script Results
9F5C	Magstripe Data Object List (MDOL)
9F5D	Available Offline Spending Amount (AOSA)
9F5D	Application Capabilities Information (ACI)
9F5E	Consecutive Transaction International Upper Limit (CTIUL)
9F5E	DS ID
9F5F	DS Slot Availability
9F60	CVC3 (Track1)
9F61	CVC3 (Track2)
9F62	PCVC3 (Track1)
9F64	NATC (Track1)
9F65	PCVC3 (Track2)
9F66	PUNATC (Track2)
9F67	NATC (Track2)
9F68	Card Additional Processes
9F69	UDOL
9F6A	Unpredictable Number (Numeric)
9F6B	Track 2 Data
9F6C	Card Transaction Qualifiers (CTQ)
9F6D	Mag-stripe Application Version Number (Reader)
9F6E	Third Party Data
9F6E	Terminal Transaction Capabilities
9F6F	DS Slot Management Control
9F70	Protected Data Envelope 1
9F71	Protected Data Envelope 2
9F72	Protected Data Envelope 3
9F73	Protected Data Envelope 4
9F74	Protected Data Envelope 5
9F75	Unprotected Data Envelope 1
9F76	Unprotected Data Envelope 2
9F77	Unprotected Data Envelope 3
9F78	Unprotected Data Envelope 4
9F79	Unprotected Data Envelope 5
9F7A	VLP Terminal Support Indicator
9F7B	VLP Terminal Transaction Limit
9F7C	Customer Exclusive Data (CED)
9F7D	DS Summary 1

Tag	Description
9F7F	DS Unpredictable Number
A5	File Control Information (FCI) Proprietary Template
BF0C	File Control Information (FCI) Issuer Discretionary Data
BF50	Visa Fleet - CDO
BF60	Integrated Data Storage Record Update Template
C3	Card issuer action code -decline
C4	Card issuer action code -default
C5	Card issuer action code online
C6	PIN Try Limit
C7	CDOL 1 Related Data Length
C8	Card risk management country code
C9	Card risk management currency code
CA	Lower cumulative offline transaction amount
CB	Upper cumulative offline transaction amount
CD	Card Issuer Action Code (PayPass) – Default
CE	Card Issuer Action Code (PayPass) – Online
CF	Card Issuer Action Code (PayPass) – Decline
D1	Currency conversion table
D2	Integrated Data Storage Directory (IDSD)
D3	Additional check table
D5	Application Control
D6	Default ARPC response code
D7	Application Control (PayPass)
D8	AIP (PayPass)
D9	AFL (PayPass)
DA	Static CVC3-TRACK1
DB	Static CVC3-TRACK2
DC	IVCVC3-TRACK1
DD	IVCVC3-TRACK2
DF01	ApplePay VAS Protocol
DF02	ApplePay VAS Failure Report
DF10	Terminal Languages Supported
DF10	Multi Language (Default: "enfr")
DF11	Enable Transaction Logging
DF13	Terminal Action Code - Default
DF14	Terminal Action Code - Denial
DF15	Terminal Action Code - Online
DF17	Threshold Value for Biased Random Selection
DF18	Target Percentage to be Used for Random Selection
DF19	Maximum Target Percentage to be used for Biased Random Selection
DF1F	Last 4 digits of Primary Account Number (PAN)
DF21	Issuer Script Results
DF22	Force Online (1-Enable, 0-Disable)
DF25	Default DDOL (1-Enable, 0-Disable)
DF26	Revocation List Support (Default: Enable - 1)
DF27	Exception File Support (Default: Disable - 0)
DF28	Default TDOL
DF29	Terminal Capabilities - CVM Required
DF2A	Threshold Value for Biased Random Selection(Interac)

Tag	Description
DF2B	Maximum Target Percentage for Biased Random Selection (Interac)
DF2C	Target Percentage for Random Selection(Interac)
DF30	Track Data Source
DF31	DD Card Track 1
DF32	DD Card Track 2
DF33	Interac Receipt Required
DF34	TTK Customer - Firmware Version
DF40	Message to be displayed by EMV Kernel on "PIN Try Limit Exceeded" condition
DF41	Message to be displayed by EMV Kernel on "Last PIN Try" condition
DF42	Message to be displayed by EMV Kernel on "Please Try Again" condition
DF43	Message to be displayed by EMV Kernel on "Call Your Bank" condition
DF45	GMEDS Secret Keys
DF46	GMAD MIDs
DF47	ISIS Read Cmd Data
DF48	ISIS Write Data
DF49	ISIS Transaction Data
DF4A	TTK Customer - Current KSN of Data encryption Key
DF4B	TTK Customer - MSR all track data
DF4C	TTK Customer - Masked PAN
DF4D	TTK Customer - Additional POS Info
DF4E	Polling Options
DF4F	TTK Customer - Fallback Reason
DF50	Special Flow
DF51	Amex Terminal Capability
DF52	Transaction CVM
DF55	RID
DF56	Activate Trans for DESFireViVOCComm Flows
DF57	Reader Primary Language
DF57	2nd usage: Remaining Candidates
DF58	Reader Secondary Language
DF5A	TLVExclusion List
DF5B	Terminal Entry Capability
DF5C	RF Deactivate Period
DF5D	D-PAS Issuer Script Response status
DF5E	Transaction Timing Information
DF5F	Encrypted PAN for remote PIN Pad
DF60	Product ID
DF61	Processor ID
DF61	CVMRequiredLimit_JCBScheme
DF62	Main Firmware Build ID
DF63	CB Enhanced DDA Indicator (same block as DF03)
DF64	CB Wave 2 CVM Requirements (same block as DF04)
DF65	Build ID Num (Cxx)
DF65	CB Display Offline Funds Indicator (same block as DF05)
DF65	Serial heartbeat Required
DF66	SVN Number
DF66	CB Terminal Type (same block as 9F35)
DF66	Display Unsupported Card
DF68	Enable/Disable STOP command processing
DF69	ConfigureProprietaryTags

Tag	Description
DF6A	Enable/Disable Comm Error Recovery
DF6C	Cubic FTP Phase 2 Mode Options
DF6D	Cubic Mode 3 Match AID
DF6E	Cubic Fixed Fare Amounts
DF6F	Cubic Timestamp Data
DF70	Loyalty Program ID
DF70	Generic Name String
DF71	Value Added Tax 1
DF71	Generic Numeric
DF72	Value Added Tax 2
DF72	Generic Specification String
DF73	Merchant Category Code
DF73	Generic Implementation String
DF74	Discover Optional Features
DF75	Communications Error Message Delay
DF76	TVR from GenAC
DF77	ViVOpay MSR Custom Data Output Tag
DF78	MC Timing Performance Enable
DF79	Card Disable Mask
DF7A	Card Disable Interval
DF7B	Serial Port (UART) Inter-character Timeout Period
DF7C	Auto Switch Feature
DF7D	Track Formatting Feature
DF7F	Improved Collision Detection & Media Removal Feature
DF891B	Poll Mode
DF891C	Interac Retry Limit
DFDE04	MSR Encryption Option
DFEE0C	PPSE Terminate Flags
DFEE12	KID
DFEE15	Application Selection Indicator
DFEE16	DUKPT Key or MKSK Select for Online PIN Encrypted
DFEE17	ICC Terminal Entry Mode
DFEE18	MSR Terminal Entry Mode
DFEE19	Online DOL
DFEE1A	Output data element
DFEE1B	Authorization Request data elements
DFEE1E	Contact Terminal Configuration (see below)
DFEE1F	Issuer script device limit, Range: 0~255 (Default: 128)
DFEE20	ICC Power on detect waiting time. (Unit: Sec) (Default: 60S)
DFEE21	ICC L1 waiting time. (Unit: Sec)(Default: 10 S)
DFEE22	Driver (Menu, Get PIN, Get MSR) Timeout. (Unit: Sec) (see below)
DFEE23	MSR Track Data
DFEE24	Force Acceptance (Default: 00)
DFEE25	ICC Response Code
DFEE26	Encryption StatusInformation
DFEE27	MSR Control
DFEF1A	TLV available
DFEF1A	Encrypted Sensitive Tags
DFEF1A	Auto Authenticate
DFEF20	MAC option in reponse data

Tag	Description
DFEF21	BIN
DFEF22	AID
DFEF23	HMAC
DFEF24	HMAC KSN
DFEF25	Output Data Format Select
DFEF26	MSR fallback
DFEF27	Online capability
DFEF28	Disable Encrypt ON
DFEF2C	Terminal AID List
DFEF2E	Terminal Transaction Log
DFEF2F	CUP configuration
DFEF30	White List
DFEF31	Black List
DFEF32	Auto-Switch
DFEF34	Antenna Detection Switch
DFEF35	Communications Watchdog Period
DFEF36	Media Control & Status Tracking
DFEF37	Interface Select
DFEF38	Timeout for Next Command
DFEF39	Network Indicate
DFEF3A	Reader Behavior Mode
DFEF3B	Autopoll Transaction Separation Interval
DFEF40	Ascii-code encryption Tag57 TLV
DFEF41	MAC Verification Data for SRED
DFEF42	MAC VerificationKSN for SRED
DFEF43	Local TZ/DST information.
DFEF44	CombinationOptions
DFEF45	Removal Timeout
DFEF46	ACT Pass Response DOL
DFEF47	CDA Hash Input
DFEF48	Indicate - retrieve transaction result again due to Output RAM is Not enough.
DFEF49	Outcome Parameter Set
DFE↔ F4A	User Interface Request Data
DFEF4B	MSR Equivalent Data Option
DFEF4C	MSR Equivalent Data Track Lengths
DFEF4D	MSR Equivalent Data
DFEF4E	ACT MSD Response DOL
DFEF4F	ACT Decline Response DOL
DFEF50	Terminal Interchange Profile (JCB)
DFEF51	Bypass EMV Completion Output
DFEF52	Re-FallBack times
DFEF53	Dynamic Reader Limits
DFEF54	SmartTap AID Index
DFEF55	Kernel Specific Features
DFEF56	Retry Limit
DFEF57	PPSE Terminate Flags
DFEF59	Terminal Data Setting - Default Amount
DFEF5A	Terminal Data Setting - Tags to Return
DFE↔ F5B	Mask for Tag5A

Tag	Description
DFEF5C	Mask for Tag56
DFEF5D	Mask for Tag57
DFEF5E	Mask for Tag9F6B
DFEF5F	Mask for TagFFEE13
DFEF60	Mask for TagFFEE14
DFEF61	Error Code
DFEF62	Allow MSR Swipe data from ICC Card
DFEF63	Tags To Read Yet
DFEF64	Referral Timeout
DFEF6E	USB-KB Output Data Postfix
DFEF6F	Inter-character Delay for USB-KB Interface
DFEF70	PISCES dual interface interference prevention mechanism fine-tune parameters.
DFEF71	Waiting ICC insert time
DFEF72	Pre-poll card mechanism control in ACT cmd & config setting
DFEF73	Transaction Message Type
DFEF74	Reference amplitude value
DFEF75	Reference delta value
DFEF76	Transaction Interface Type to activate
DFEF77	Timeout for waiting next command
DFEF78	EMV contact L2 display messages option
DFEF79	PIN block format (when TDES)
DFEF7A	Enable Apple Pay Check
DFEF7B	Apple Pay Status
DFEF7C	Track Bit Encoding
DFEF7D	Re-power on times
DFEF7E	Fallback response code list
FF69	ViVOpay Proprietary Tag List
FF70	Serial Finite State Machine Version
FF71	Transaction Finite State Machine Version
FF72	System Information Suite
FF73	Serial Protocol Version
FF74	Serial Protocol Suite
FF75	L1 Paypass Version
FF76	L1 LCR Version
FF77	L2 Card App Version
FF78	L2 Card App Suite
FF79	GMEDs Data
FF79	User Experience Version
FF7A	User Experience Suite
FF7B	ViVOtech Proprietary Suite
FF7C	VIUDS Scheme IDs Supported
FF7D	VIUDS Scheme ID Selection Criteria
FFE0	Registered Application Provider Identifier (RID)
FFE1	Partial Selection Allowed
FFE2	Application Flow
FFE3	Selection Features - GR 1.2.10
FFE4	Group Number / Fallback Group
FFE5	Max AID Length
FFE6	AID Disabled

Tag	Description
FFE7	Interface Support
FFE8	Exclude from Processing
FFE9	Kernel ID Transaction Type Group List
FFEA	Default Kernel ID
FFEE01	ViVOpay TLV Group Tag
FFEE02	ViVOpay Pre-PPSE Special Flow Group Tag
FFEE03	ViVOpay Post-PPSE Special Flow Group Tag
FFEE04	M/Chip3 Intermediate Message Data
FFEE05	M/Chip3 Intermediate Message Marker
FFEE06	ApplePay VAS Container
FFEE07	Encrypted Sensitive Tags
FFEE08	Masked Tags
FFEE0A	BIN Range
FFEE0B	AID Range
FFEE0C	White List
FFEE10	ViVOpay MChip Group Tag
FFEE11	ViVOpay Discover Group Tag
FFEE12	KID
FFEE12	Cash Reader Risk Record
FFEE13	Track 1 Data
FFEE13	Cashback Reader Risk Record
FFEE14	Track 2 Data
FFEE14	DRL Record 1
FFEE15	DRL Record 2
FFEE16	DRL Record 3
FFEE17	DRL Record 4
FFEE18	Tags To Write Yet Before GenAC
FFEE19	Tags To Write Yet After GenAC
FFEE1A	Terminal App DET Data
FFEE1C	Unpredictable Number Range
FFEE1D	Sensitive Data Mask
FFE↔ E1E	Group 0 Initialize Flag
FFE↔ E1F	Error Code Table
FFEE20	Restart Deactivation Time
FFF0	Specific Features Switch
FFF1	Terminal Contactless Transaction Limit
FFF2	Terminal IFD
FFF3	Application Capability
FFF4	Visa Reader Risk Flags
FFF6	Torn Transaction Log Clean Interval (minutes)
FFF7	Burst Mode
FFF8	UI Scheme
FFF9	LCD Font Size
FFFA	LCD Delay Time
FFFB	Language Option for LCD
FFFC	Force MagStripe

9F33 Terminal Capabilities

Byte 1

b8	b7	b6	b5	b4	b3	b2	b1	Meaning
1	x	x	x	x	x	x	x	Manual key entry
x	1	x	x	x	x	x	x	Magnetic stripe
x	x	1	x	x	x	x	x	IC with contacts
x	x	x	0	x	x	x	x	RFU
x	x	x	x	0	x	x	x	RFU
x	x	x	x	x	0	x	x	RFU
x	x	x	x	x	x	0	x	RFU
x	x	x	x	x	x	x	0	RFU

b8	b7	b6	b5	b4	b3	b2	b1	Meaning
1	x	x	x	x	x	x	x	Plaintext PIN for IC verification
x	1	x	x	x	x	X	x	Enciphered PIN for online verification
x	x	1	x	x	x	X	x	Signature(paper)
x	x	x	1	x	x	X	x	Enciphered PIN for offline verification
x	x	x	x	1	x	X	x	No CVM Required
x	x	x	x	x	0	x	x	RFU
x	x	x	x	x	x	0	x	RFU
x	x	x	x	x	x	X	0	RFU

b8	b7	b6	b5	b4	b3	b2	b1	Meaning
1	x	x	x	x	x	x	x	SDA
X	1	x	x	x	x	x	x	DDA
X	x	1	x	x	x	x	x	Card capture
x	x	x	0	x	x	x	x	RFU
x	x	x	x	1	x	x	X	CDA
x	x	x	x	x	0	x	x	RFU
x	x	x	x	x	x	0	x	RFU
x	x	x	x	X	X	x	0	RFU

9F40 Additional Terminal Capabilities

b1	b2	b3	b4	b5	b6	b7	b8	Meaning
1	x	x	x	x	x	x	x	Cash
x	1	x	x	x	x	x	x	Goods
x	x	1	x	x	x	x	x	Services
x	x	x	1	x	x	x	x	Cashback
x	x	x	x	1	x	x	x	Inquiry
x	x	x	x	x	1	x	x	Transfer
x	x	x	x	x	x	1	x	Payment
x	x	x	x	x	x	x	1	Administrative

b8	b7	b6	b5	b4	b3	b2	b1	Meaning
1	x	x	x	x	x	x	x	Cash Deposit
x	0	x	x	x	x	x	x	RFU
x	x	0	x	x	x	x	x	RFU
x	x	x	0	x	x	x	x	RFU
x	x	x	x	0	x	x	x	RFU
x	x	x	x	x	0	x	x	RFU
x	x	x	x	x	x	0	x	RFU
x	x	x	x	x	x	x	0	RFU

b8	b7	b6	b5	b4	b3	b2	b1	Meaning
1	x	x	x	x	x	x	x	Numeric keys
x	1	x	x	x	x	x	x	Alphabetic and special characters keys
x	x	1	x	x	x	x	x	Command keys
x	x	x	1	x	x	x	x	Function Keys
x	x	x	x	0	x	x	x	RFU
x	x	x	x	x	0	x	x	RFU
x	x	x	x	x	x	0	x	RFU
x	x	x	x	x	x	x	0	RFU

b8	b7	b6	b5	b4	b3	b2	b1	Meaning
1	x	x	x	x	x	x	x	Print, attendant
x	1	x	x	x	x	x	x	Print, cardholder
x	x	1	x	x	x	x	x	Display, attendant
x	x	x	1	x	x	x	x	Display, cardholder
x	x	x	x	0	x	x	x	RFU
x	x	x	x	x	0	x	x	RFU
x	x	x	x	x	x	1	x	Code table 10
x	x	x	x	x	x	x	1	Code table 9

b8	b7	b6	b5	b4	b3	b2	b1	Meaning
1	x	x	x	x	x	x	x	Code table 8
x	1	x	x	x	x	x	x	Code table 7
x	x	1	x	x	x	x	x	Code table 6
x	x	x	1	x	x	x	x	Code table 5
x	x	x	x	1	x	x	x	Code table 4
x	x	x	x	x	1	x	x	Code table 3

x	x	x	x	x	x	1	x	Code table 2
x	x	x	x	x	x	x	1	Code table 1

9F35 Terminal Type

Environment	Financial Institution	Merchant	Cardholder
Attended			
Online only	11	21	
Offline with online capability	12	22	
Offline only	13	23	
Unattended			
Online only	14	24	34
Offline with online capability	15	25	35
Offline only	16	26	36

DFEE1E Contact Terminal Configuration (Default: F0 DC 3C F0 C2 9E 94 00)

Byte 1								
b8	b7	b6	b5	b4	b3	b2	b1	Meaning
1	x	x	x	x	x	x	x	Key Pad support
x	1	x	x	x	x	x	x	LCD support
x	x	1	x	x	x	x	x	PIN Pad support
x	x	x	1	x	x	x	x	Print Support
x	x	x	x	0	x	x	x	RFU
x	x	x	x	x	0	x	x	RFU
x	x	x	x	x	x	0	x	RFU
x	x	x	x	x	x	X	0	RFU
Byte 2								
b8	b7	b6	b5	b4	b3	b2	b1	Meaning
1	x	x	x	x	x	x	x	PSE support
x	1	x	x	x	x	x	x	Cardholder confirmation
x	x	1	x	x	x	x	x	Preferred display order
x	x	x	1	x	x	x	x	Multi language
x	x	x	x	1	x	x	x	EMV language selection method
x	x	x	x	x	1	x	x	Default DDOL
x	x	x	x	x	x	0	x	RFU
x	x	x	x	x	x	x	0	RFU
Byte 3								
b8	b7	b6	b5	b4	b3	b2	b1	Meaning
0	x	x	x	x	x	x	x	RFU
(Revocation of Issuer Public Key Certificate (DF26))								
x	1	x	x	x	x	x	x	Manual action when CA PK loading fails
x	x	1	x	x	x	x	x	CA PK verified with check sum
x	x	x	1	x	x	x	x	Bypass PIN Entry
x	x	x	x	1	x	x	x	Subsequent bypass PIN Entry
x	x	x	x	1	x	x	x	Get data for pin try counter
x	x	x	x	x	x	0	x	RFU
x	x	x	x	x	x	x	0	RFU
Byte 4								
b8	b7	b6	b5	b4	b3	b2	b1	Meaning
1	x	x	x	x	x	x	x	Amount before CVM processing
x	1	x	x	x	x	x	x	Floor limit checking
x	x	1	x	x	x	x	x	Random transaction selection
x	x	x	1	x	x	x	x	Velocity checking
x	x	x	x	0	x	x	x	RFU
(Transaction Log (DF11))								
x	x	x	x	x	0	x	x	RFU
(Exception File (DF27))								
x	x	x	x	x	x	0	x	RFU
x	x	x	x	x	x	x	0	RFU
Byte 5								
b8	b7	b6	b5	b4	b3	b2	b1	Meaning
1	x	x	x	x	x	x	X	Terminal action code support
x	1	x	x	x	x	x	x	Terminal action code can be change
x	x	1	x	x	x	x	x	Terminal action code can be deleted or disable
x	x	x	1	x	x	x	x	Default Action code processing before 1st GAC
x	x	x	x	1	x	x	x	Default Action code processing after 1st GAC
x	x	x	x	x	1	x	x	TAC/IAC default process when unable to go online (Skipped)
x	x	x	x	x	x	1	x	TAC/IAC default process when unable to go online (Normal)
x	x	x	x	x	x	x	0	RFU

Byte 6

b8	b7	b6	b5	b4	b3	b2	b1	Meaning
1	x	x	x	x	x	x	x	Forced Online support
x	1	x	x	x	x	x	x	Forced acceptance support
x	x	1	x	x	x	x	x	Advices support
x	x	x	1	x	x	x	x	Issuer referrals support
X	x	x	x	1	x	x	x	Batch data capture
x	x	x	x	x	1	x	x	Online data capture
X	x	x	x	x	x	1	x	Default TDOL
X	x	x	x	x	x	x	0	RFU

Byte 7

b8	b7	b6	b5	b4	b3	b2	b1	Meaning
1	x	x	x	x	x	x	x	amount and pin entered on the same keypad
x	1	x	x	x	x	x	x	ICC/Magstripe reader combined
x	x	1	x	x	x	x	x	Magstripe read first
x	x	x	1	x	x	x	x	Support account type selection
x	x	x	x	1	x	x	x	On fly script processing
x	x	x	x	x	1	x	x	Internal date management
x	x	x	x	x	x	1	x	Reversal Mode
(1)Unable go online								
(2) ARC Error								
0: (3) Online Approved but reader not approved.								
1: (3) Online Approved but card response AAC.								
x	x	x	x	x	x	x	0	RFU

Byte 8

b8	b7	b6	b5	b4	b3	b2	b1	Meaning
x	x	x	x	x	x	x	x	RFU

DFEE22 Driver (Menu, Get PIN, Get MSR) Timeout. (Unit: Sec)

Byte1: Timeout for Menu. (Default: 30 S)
 Byte2: Timeout for Get PIN. (Default: 60 S)
 Byte3: Timeout for Get MSR. (Default: 60 S)

Chapter 18

Namespace Index

18.1 Packages

Here are the packages with brief descriptions (if available):

IDTechSDK	80
-------------------------------------	----

Chapter 19

Class Index

19.1 Class List

Here are the classes, structs, unions and interfaces with brief descriptions:

IDTechSDK.IDT_Augusta	81
IDTechSDK.IDT_BTMag	141
IDTechSDK.IDT_BTPay	174
IDTechSDK.IDT_CM100	226
IDTechSDK.IDT_K100	234
IDTechSDK.IDT_KioskIII	247
IDTechSDK.IDT_L100	287
IDTechSDK.IDT_L80	318
IDTechSDK.IDT_MiniSmartII	339
IDTechSDK.IDT_NEO2	382
IDTechSDK.IDT_PIP	537
IDTechSDK.IDT_SecureKey	563
IDTechSDK.IDT_SecureMag	586
IDTechSDK.IDT_SpectrumPro	612
IDTechSDK.IDT_SREDKey2	659
IDTechSDK.IDT_UniPay	695
IDTechSDK.IDT_UniPayI_V	714
IDTechSDK.IDT_Vendi	757
IDTechSDK.IDT_VendIII	786
IDTechSDK.IDT_VP3300	814
IDTechSDK.IDT_VP8800	876

Chapter 20

Namespace Documentation

20.1 IDTechSDK Namespace Reference

Classes

- class [IDT_Augusta](#)
- class [IDT_BTMag](#)
- class [IDT_BTPay](#)
- class [IDT_CM100](#)
- class [IDT_K100](#)
- class [IDT_KioskIII](#)
- class [IDT_L100](#)
- class [IDT_L80](#)
- class [IDT_MiniSmartII](#)
- class [IDT_NEO2](#)
- class [IDT_PIP](#)
- class [IDT_SecureKey](#)
- class [IDT_SecureMag](#)
- class [IDT_SpectrumPro](#)
- class [IDT_SREDKey2](#)
- class [IDT_UniPay](#)
- class [IDT_UniPayI_V](#)
- class [IDT_Vendi](#)
- class [IDT_VendIII](#)
- class [IDT_VP3300](#)
- class [IDT_VP8800](#)

Chapter 21

Class Documentation

21.1 IDTechSDK.IDT_Augusta Class Reference

Public Member Functions

- RETURN_CODE [device_StartRKI](#) (int type, string ident="")
- RETURN_CODE [emv_startTransaction](#) (double amount, double amtOther, int exponent, int type, int timeout, byte[] tags, bool forceOnline, string ident="")
- RETURN_CODE [emv_activateTransaction](#) (int timeout, byte[] tags, bool forceOnline, bool isFastEMV=false, string ident="")
- RETURN_CODE [device_startTransaction](#) (double amount, double amtOther, int exponent, int type, int timeout, byte[] tags, bool forceOnline, bool isFastEMV=false, string ident="")
- RETURN_CODE [device_activateTransaction](#) (int timeout, byte[] tags, bool forceOnline, bool isFastEMV=false, string ident="")
- RETURN_CODE [emv_authenticateTransaction](#) (byte[] updatedTLV, string ident="")
- RETURN_CODE [emv_completeTransaction](#) (bool commError, byte[] authCode, byte[] iad, byte[] tlvScripts, byte[] tlv, string ident="")
- RETURN_CODE [emv_callbackResponseLCD](#) (EMV_LCD_DISPLAY_MODE type, byte selection, string ident="")
- RETURN_CODE [emv_callbackResponsePIN](#) (EMV_PIN_MODE type, byte[] KSN, byte[] PIN, string ident="")
- RETURN_CODE [emv_callbackResponseMSR](#) (byte[] MSR, string ident="")
- RETURN_CODE [emv_cancelTransaction](#) (string ident="")
- RETURN_CODE [emv_getEMVKernelVersion](#) (ref string response, string ident="")
- RETURN_CODE [emv_getEMVKernelCheckValue](#) (ref string response, string ident="")
- RETURN_CODE [emv_getEMVConfigurationCheckValue](#) (ref string response, string ident="")
- RETURN_CODE [emv_retrieveTransactionResult](#) (byte[] tags, ref IDTTransactionData tlv, string ident="")
- RETURN_CODE [emv_removeApplicationData](#) (byte[] AID, string ident="")
- RETURN_CODE [emv_removeAllApplicationData](#) (string ident="")
- RETURN_CODE [emv_removeCAPK](#) (byte[] capk, string ident="")
- RETURN_CODE [emv_removeAllCAPK](#) (string ident="")
- RETURN_CODE [emv_removeCRL](#) (byte[] crlList, string ident="")
- RETURN_CODE [emv_removeAllCRL](#) (string ident="")
- RETURN_CODE [emv_removeTerminalData](#) (string ident="")
- RETURN_CODE [emv_retrieveAIDList](#) (ref byte[][] response, string ident="")
- RETURN_CODE [device_setQuickChipMode](#) (bool turnON, string ident="")
- RETURN_CODE [device_getQuickChipMode](#) (ref bool isOn, string ident="")
- RETURN_CODE [emv_retrieveApplicationData](#) (byte[] AID, ref byte[] tlv, string ident="")
- RETURN_CODE [emv_retrieveCAPK](#) (byte[] capk, ref byte[] key, string ident="")
- RETURN_CODE [emv_retrieveCAPKList](#) (ref byte[] keys, string ident="")
- RETURN_CODE [emv_retrieveCRLList](#) (ref byte[] list, string ident="")

- RETURN_CODE [emv_retrieveTerminalData](#) (ref byte[] tlv, string ident="")
- RETURN_CODE [emv_setApplicationData](#) (byte[] name, byte[] tlv, string ident="")
- RETURN_CODE [emv_setCAPK](#) (byte[] key, string ident="")
- RETURN_CODE [emv_setCRL](#) (byte[] list, string ident="")
- RETURN_CODE [emv_trySetTerminalData](#) (byte[] tlv, ref byte[] rejectedTLV, ref byte[] convertedTLV, bool overwrite=false, string ident="")
- RETURN_CODE [emv_setTerminalData](#) (byte[] tlv, string ident="")
- RETURN_CODE [emv_addTerminalData](#) (byte[] tlv, string ident="")
- RETURN_CODE [emv_setTerminalMajorConfiguration](#) (int configuration, string ident="")
- RETURN_CODE [config_getModelNumber](#) (ref string response, string ident="")
- RETURN_CODE [config_getSerialNumber](#) (ref string response, string ident="")
- bool [config_setCmdTimeOutDuration](#) (int newTimeOut, string ident="")
- RETURN_CODE [config_setLEDController](#) (bool firmwareControlMSRLED, bool firmwareControlICCL↵ED=false, string ident="")
- RETURN_CODE [config_getLEDController](#) (ref bool firmwareControlMSRLED, ref bool firmwareControlICC↵LED, string ident="")
- RETURN_CODE [config_setBeeperController](#) (bool firmwareControlBeeper, string ident="")
- RETURN_CODE [config_getBeeperController](#) (ref bool firmwareControlBeeper, string ident="")
- RETURN_CODE [device_controlLED_ICC](#) (int controlMode, int interval, string ident="")
- RETURN_CODE [device_controlBeep](#) (int index, int frequency, int duration, string ident="")
- RETURN_CODE [device_controlLED](#) (byte indexLED, byte control, int intervalOn=500, int intervalOff=500, string ident="")
- RETURN_CODE [config_setEncryptionControl](#) (bool msr, bool icc, string ident="")
- RETURN_CODE [config_getEncryptionControl](#) (ref bool msr, ref bool icc, string ident="")
- RETURN_CODE [device_getKeyStatus](#) (ref Boolean newFormat, ref byte[] status, string ident="")
- RETURN_CODE [device_setDateTime](#) (string ident="")
- RETURN_CODE [device_startRKI](#) (string ident="")
- RETURN_CODE [device_getDateTime](#) (ref byte[] dateTime, string ident="")
- RETURN_CODE [device_getDRS](#) (ref byte[] codeDRS, string ident="")
- RETURN_CODE [device_verifyBackdoorKey](#) (string ident="")
- RETURN_CODE [device_selfCheck](#) (string ident="")
- RETURN_CODE [device_getFirmwareVersion](#) (ref string response, string ident="")
- string [device_getResponseCodeString](#) (RETURN_CODE eCode)
- RETURN_CODE [device_rebootDevice](#) (string ident="")
- RETURN_CODE [device_sendDataCommand](#) (string cmd, bool calcLRC, ref byte[] response, string ident="")
- RETURN_CODE [device_getTransArmorID](#) (ref string TID, string ident="")
- RETURN_CODE [device_setTransArmorID](#) (string TID, string ident="")
- RETURN_CODE [device_getKeyTypeForICCDUKPT](#) (ref byte type, string ident="")
- RETURN_CODE [device_getKeyFormatForICCDUKPT](#) (ref byte format, string ident="")
- RETURN_CODE [device_sendPAE](#) (string command, ref string response, int timeout, string ident="")
- RETURN_CODE [device_setTransArmorEncryption](#) (byte[] cert, string ident="")
- RETURN_CODE [device_sendDataCommand_ext](#) (string cmd, bool calcLRC, ref byte[] response, int timeout, bool noResponse, string ident="")
- RETURN_CODE [device_controlLED_MSR](#) (byte control, int intervalOn=500, int intervalOff=500, string ident="")
- RETURN_CODE [icc_enable](#) (bool withNotification, string ident="")
- RETURN_CODE [icc_disable](#) (string ident="")
- RETURN_CODE [icc_getFunctionStatus](#) (ref bool enabled, ref bool withNotification, string ident="")
- RETURN_CODE [icc_exchangeAPDU](#) (string c_APDU, ref byte[] response, string ident="")
- RETURN_CODE [icc_exchangeEncryptedAPDU](#) (string c_APDU, ref byte[] response, string ident="")
- RETURN_CODE [icc_getAPDU_KSN](#) (ref byte[] ksn, string ident="")
- RETURN_CODE [icc_getICCReaderStatus](#) (ref byte status, string ident="")
- RETURN_CODE [icc_getKeyFormatForICCDUKPT](#) (ref byte format, string ident="")
- RETURN_CODE [icc_getKeyTypeForICCDUKPT](#) (ref byte type, string ident="")
- RETURN_CODE [icc_powerOffICC](#) (string ident="")

- RETURN_CODE [icc_powerOnICC](#) (ref byte[] ATR, string ident="")
- RETURN_CODE [icc_setKeyFormatForICCDUKPT](#) (byte encryption, string ident="")
- RETURN_CODE [icc_setKeyTypeForICCDUKPT](#) (byte encryption, string ident="")
- RETURN_CODE [msr_captureMode](#) (bool isBufferMode, bool withNotification=false, string ident="")
- RETURN_CODE [msr_disable](#) (string ident="")
- RETURN_CODE [msr_getFunctionStatus](#) (ref bool enabled, ref bool isBufferMode, ref bool withNotification, string ident="")
- RETURN_CODE [msr_switchUSBInterfaceMode](#) (bool blsUSBKeyboardMode)
- RETURN_CODE [msr_switchUSBInterfaceMode](#) (bool blsUSBKeyboardMode, ref string ident)
- RETURN_CODE [msr_getClearPANID](#) (ref byte value, string ident="")
- RETURN_CODE [device_disableAugustaLED](#) (bool disable, string ident="")
- RETURN_CODE [msr_getExpirationMask](#) (ref byte value, string ident="")
- RETURN_CODE [msr_getSwipeEncryption](#) (ref byte encryption, string ident="")
- RETURN_CODE [msr_getSwipeForcedEncryptionOption](#) (ref byte option, string ident="")
- RETURN_CODE [msr_getSwipeMaskOption](#) (ref byte option, string ident="")
- RETURN_CODE [msr_RetrieveWhiteList](#) (ref byte[] value, string ident="")
- RETURN_CODE [icc_getSetting](#) (byte setting, ref byte value, string ident="")
- RETURN_CODE [icc_getSettings](#) (byte setting, ref byte[] value, string ident="")
- RETURN_CODE [icc_setSetting](#) (byte setting, byte value, string ident="")
- RETURN_CODE [icc_setSetting](#) (byte setting, byte[] value, string ident="")
- RETURN_CODE [msr_getSetting](#) (byte setting, ref byte value, string ident="")
- RETURN_CODE [msr_getSettings](#) (byte setting, ref byte[] value, string ident="")
- RETURN_CODE [msr_setSetting](#) (byte setting, byte value, string ident="")
- RETURN_CODE [msr_setSettings](#) (byte setting, byte[] value, string ident="")
- RETURN_CODE [msr_setClearPANID](#) (byte val, string ident="")
- RETURN_CODE [msr_setExpirationMask](#) (bool mask, string ident="")
- RETURN_CODE [msr_setSwipeEncryption](#) (byte encryption, string ident="")
- RETURN_CODE [msr_setSwipeForcedEncryptionOption](#) (bool track1, bool track2, bool track3, bool track3card0, string ident="")
- RETURN_CODE [msr_setSwipeMaskOption](#) (bool track1, bool track2, bool track3, string ident="")
- RETURN_CODE [msr_setWhiteList](#) (byte[] value, byte[] MAC, string ident="")
- RETURN_CODE [msr_setWhiteListFromBDK](#) (byte[] val, byte[] BDK, string ident="")
- RETURN_CODE [msr_startMSRSwipe](#) (int timeout, string ident="")
- RETURN_CODE [msr_cancelMSRSwipe](#) (string ident="")
- bool [device_isSRED](#) (string ident="")
- bool [createFastEMVData](#) (ref IDTTransactionData cData, string ident="")
- RETURN_CODE [device_getRKIStatus](#) (bool isProd, bool isMultiKey, ref string status, ref Dictionary< string, RKI_KEY_TYPE > keys, string ident="")
- RETURN_CODE [device_getAnyRKIStatus](#) (bool isMultiKey, ref string status, ref Dictionary< string, RKI_KEY_TYPE > keys, string ident="")
- RETURN_CODE [device_updateFirmwareFromZip](#) (byte[] zipfile, string ident="", bool isForeground=false)
- RETURN_CODE [device_sendConfigurationFromZip](#) (byte[] zip, string filename, VC_OPERATION_TYPE type, bool matchFW, string ident="", bool isForeground=false)
- RETURN_CODE [device_sendConfiguration](#) (string filename, VC_OPERATION_TYPE type, bool matchFW, string ident="", bool isForeground=false)
- RETURN_CODE [device_readConfigurationToMemory](#) (string memo, Configs.InstallRules rules, List< Configs.DeviceCommand > cmds, string ident="", bool isForeground=false)
- RETURN_CODE [device_getConfigurationFromMemory](#) (ref string json, string ident="")
- RETURN_CODE [device_RemoteKeyInjection](#) (RKI_KEY_TYPE type, string keyName, string ident="", bool performOnForeground=false)

Static Public Member Functions

- static void **initSC** ()
- static int **getCommandTimeout** (string ident="")
- static void **setCommandTimeout** (int milliseconds, string ident="")
- static void **setCallback** (Callback my_Callback)
- static void **setCallback** (IntPtr my_Callback, SynchronizationContext context)
- static void **emv_autoAuthenticate** (bool authenticate, string ident="")
- static void **emv_autoAuthenticateTags** (bool authenticate, byte[] tags, string ident="")
- static void **emv_allowFallback** (bool allow, string ident="")
- static void **lcd_retrieveMessage** (DisplayMessages.DISPLAY_MESSAGE_LANGUAGE lang, DisplayMessages.DISPLAY_MESSAGE_IDENTIFIER id, ref string line1, ref string line2)
- static RETURN_CODE **device_updateDeviceFirmware** (byte[] firmwareData, string ident="")
- static String **SDK_Version** ()

Properties

- static **IDT_Augusta SharedController** [get]

21.1.1 Detailed Description

Class for Augusta/AugustaSRED MSR/ICC reader

21.1.2 Member Function Documentation

21.1.2.1 RETURN_CODE IDTechSDK.IDT_Augusta.config_getBeeperController (ref bool *firmwareControlBeeper*, string *ident* = " ")

Get the Beeper Controller Status Set the Beeper controlled Status by software or firmware

Parameters

<i>firmwareControlBeeper</i>	true means firmware control the beeper, false means software control beeper.
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString

21.1.2.2 RETURN_CODE IDTechSDK.IDT_Augusta.config_getEncryptionControl (ref bool *msr*, ref bool *icc*, string *ident* = " ")

Get Encryption Control

Get Encryption Control to switch status between MSR and ICC/EMV function. Following Encryption status supported:

- MSR ON, ICC/EMV ON,
- MSR ON, ICC/EMV OFF,
- MSR OFF, ICC/EMV OFF,

Parameters

<i>msr</i>	<ul style="list-style-type: none"> • true: enabled MSR with Encryption, • false: disabled MSR with Encryption,
<i>icc</i>	<ul style="list-style-type: none"> • true: enabled ICC with Encryption, • false: disabled ICC with Encryption,
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString

21.1.2.3 RETURN_CODE IDTechSDK.IDT_Augusta.config_getLEDController (ref bool *firmwareControlMSRLED*, ref bool *firmwareControlICCLED*, string *ident* = " ")

Get the LED Controller Status Get the MSR / ICC LED controlled status by software or firmware

Parameters

<i>firmwareControlMSRLED</i>	<ul style="list-style-type: none"> • true: firmware control the MSR LED • false: software control the MSR LED
<i>firmwareControlICCLED</i>	<ul style="list-style-type: none"> • true: firmware control the ICC LED • false: software control the ICC LED
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString

21.1.2.4 RETURN_CODE IDTechSDK.IDT_Augusta.config_getModelNumber (ref string *response*, string *ident* = " ")

Polls device for Model Number

Parameters

<i>response</i>	Returns Model Number
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString

21.1.2.5 RETURN_CODE IDTechSDK.IDT_Augusta.config_getSerialNumber (ref string *response*, string *ident* = " ")

Polls device for Serial Number

Parameters

<i>response</i>	Returns Serial Number
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString

21.1.2.6 RETURN_CODE IDTechSDK.IDT_Augusta.config_setBeeperController (bool *firmwareControlBeeper*, string *ident* = " ")

Set the Beeper Controller Set the Beeper controlled by software or firmware

Parameters

<i>firmwareControlBeeper</i>	true means firmware control the beeper, false means software control beeper.
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString

21.1.2.7 bool IDTechSDK.IDT_Augusta.config_setCmdTimeOutDuration (int *newTimeOut*, string *ident* = " ")

Command Acknowledgement Timeout

Sets the amount of seconds to wait for an {ACK} to a command before a timeout. Responses should normally be received under one second. Default is 3 seconds

Parameters

<i>newTimeOut</i>	Timeout value. Valid range 1 - 60 seconds
-------------------	---

Returns

Success flag. Determines if value was set and in range.

21.1.2.8 RETURN_CODE IDTechSDK.IDT_Augusta.config_setEncryptionControl (bool *msr*, bool *icc*, string *ident* = " ")

Set Encryption Control

Set Encryption Control to switch status between MSR and ICC/EMV function. Following Encryption status supported:

- MSR ON, ICC/EMV ON,
- MSR ON, ICC/EMV OFF,
- MSR OFF, ICC/EMV OFF,

Parameters

<i>msr</i>	<ul style="list-style-type: none"> • true: enable MSR with Encryption, • false: disable MSR with Encryption,
<i>icc</i>	<ul style="list-style-type: none"> • true: enable ICC with Encryption, • false: disable ICC with Encryption,
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString

21.1.2.9 RETURN_CODE IDTechSDK.IDT_Augusta.config_setLEDController (bool *firmwareControlMSRLED*, bool *firmwareControlICCLEd* = false, string *ident* = " ")

Set the LED Controller Set the MSR / ICC LED controlled by software or firmware NOTE: The ICC LED always controlled by software.

Parameters

<i>firmwareControlMSRLED</i>	<ul style="list-style-type: none"> • true: firmware control the MSR LED • false: software control the MSR LED
<i>firmwareControlICCLEd</i>	<ul style="list-style-type: none"> • true: firmware control the ICC LED • false: software control the ICC LED
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString

21.1.2.10 bool IDTechSDK.IDT_Augusta.createFastEMVData (ref IDTTransactionData *cData*, string *ident* = " ")

Create Fast EMV Data

At the completion of a Fast EMV Transaction, after the final card decision is returned and the IDTTransactionData object is provided, sending that cData object to this method will populate the .fastEMV element with string data that represents the Fast EMV data that would be returned from and IDTech FastEMV over KB protocol

Parameters

<i>cData</i>	The IDTTransactionData object populated with card data.
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

21.1.2.11 **RETURN_CODE** IDTechSDK.IDT_Augusta.device_activateTransaction (int *timeout*, byte[] *tags*, bool *forceOnline*, bool *isFastEMV* = false, string *ident* = " ")

Start EMV Transaction Request

Authorizes the EMV transaction for an ICC card

The tags will be returned in the callback routine.

Parameters

<i>timeout</i>	Timeout value in seconds.
<i>tags</i>	The tags to be included in the request. Passed as a TLV data. Example, tag 9F02 with amount 0x000000000100 would be 0x9F0206000000000100 If tags 9F02 (amount), 9F03 (other amount), or 9C (transaction type) are included, they will take priority over these values
<i>forceOnline</i>	TRUE = do not allow offline approval, FALSE = allow ICC to approve offline if terminal capable Note: To request tags to be included in default response, use tag DFEE1A, and specify tag list. Example four tags 9F02, 9F36, 95, 9F37 to be included in response = DFEE1A079F029F369f9F37
<i>isFastEMV</i>	If TRUE, it will populate the IDTTransactionData.fastEMV with ASCII data similar to IDTech FastEMV KB output, after performing an auto-authenticate and auto-complete with ResultCode = Could Not Contact Host
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString

21.1.2.12 **RETURN_CODE** IDTechSDK.IDT_Augusta.device_controlBeep (int *index*, int *frequency*, int *duration*, string *ident* = " ")

Control Beep

Controls the Beeper

Parameters

<i>index</i>	For Augusta, must be set to 1 (only one beeper)
<i>frequency</i>	Frequency, range 1000-20000 (suggest minimum 3000)
<i>duration</i>	Duration, in milliseconds (range 1 - 65525)
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString

21.1.2.13 `RETURN_CODE IDTechSDK.IDT_Augusta.device_controlLED (byte indexLED, byte control, int intervalOn = 500, int intervalOff = 500, string ident = " ")`

Control MSR LED

Controls the LED for the MSR

Parameters

<i>indexLED</i>	For Augusta, must be set to 1 (MSR LED)
<i>control</i>	LED Status: <ul style="list-style-type: none"> • 00: OFF • 01: RED Solid • 02: RED Blink • 11: GREEN Solid • 12: GREEN Blink • 21: BLUE Solid • 22: BLUE Blink
<i>intervalOn</i>	Blink interval ON, in ms (Range 200 - 2000)
<i>intervalOff</i>	Blink interval OFF, in ms (Range 200 - 2000)
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with `device_getResponseCodeString`

21.1.2.14 `RETURN_CODE IDTechSDK.IDT_Augusta.device_controlLED_ICC (int controlMode, int interval, string ident = " ")`

Control ICC LED

Controls the LED for the ICC card slot

Parameters

<i>controlMode</i>	0 = off, 1 = solid, 2 = blink
<i>interval</i>	Blink interval, in ms (500 = 500 ms)
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with `device_getResponseCodeString`

21.1.2.15 `RETURN_CODE IDTechSDK.IDT_Augusta.device_controlLED_MSR (byte control, int intervalOn = 500, int intervalOff = 500, string ident = " ")`

Control the MSR LED

Controls the MSR / ICC LED This API not recommended to control ICC LED

Parameters

<i>control</i>	<ul style="list-style-type: none"> • 0x00 = off, • 0x01 = RED Solid, • 0x02 = RED Blink, • 0x11 = GREEN Solid, • 0x12 = GREEN Blink, • 0x21 = BLUE Solid, • 0x22 = BLUE Blink,
<i>intervalOn</i>	Blink interval on time last, in ms (500 = 500 ms, valid from 200 to 2000)
<i>intervalOff</i>	Blink interval off time last, in ms (500 = 500 ms, valid from 200 to 2000)
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString

21.1.2.16 RETURN_CODE IDTechSDK.IDT_Augusta.device_disableAugustaLED (bool *disable*, string *ident* = " ")

Disable Augusta LED

Prevents the Augusta from receiving commands that can manipulate/change the LED status

Parameters

<i>disable</i>	TRUE = don't allow LED commands to execute, FALSE = allow LED commands to execute
----------------	---

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString

21.1.2.17 RETURN_CODE IDTechSDK.IDT_Augusta.device_getAnyRKIStatus (bool *isMultiKey*, ref string *status*, ref Dictionary< string, RKI_KEY_TYPE > *keys*, string *ident* = " ")

Get RKI Status Polls the RKI servers to validate device RKI eligibility. Note: if device type is known in advance (production or demo device), it is more efficient to use device_getRKIStatus instead

Parameters

<i>isMultiKey</i>	True = MultiKey, False = SingleKey
<i>status</i>	RKI status returned as a string
<i>keys</i>	Returns all available keys
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with `errorCode.getErrorString()`. When no data is available, return code = RETURN_CODE_NO_DATA_AVAILABLE

21.1.2.18 RETURN_CODE IDTechSDK.IDT_Augusta.device_getConfigurationFromMemory (ref string json, string ident = " ")

Read Device Configuration From Memory Retrieves the ViVOconfig configuration data from memory after a successful device_readConfigurationToMemory

Parameters

<i>json</i>	Returns the .json file data that can be saved to disk
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with `errorCode.getErrorString()`. When no data is available, return code = RETURN_CODE_NO_DATA_AVAILABLE

21.1.2.19 RETURN_CODE IDTechSDK.IDT_Augusta.device_getDateTime (ref byte[] dateTime, string ident = " ")

Get Date Time

Gets current system date and time of the device.

Parameters

<i>dateTime</i>	<p>The date time returned as follows:</p> <ul style="list-style-type: none"> • byte 0: Year 00-99 • byte 1: Month 01-12 • byte 2: Date 01-31 • byte 3: Hour 00-23 • byte 4: Minute 00-59 • byte 5: Second 00-59
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with `device_getResponseCodeString`

21.1.2.20 RETURN_CODE IDTechSDK.IDT_Augusta.device_getDRS (ref byte[] codeDRS, string ident = " ")

Get DRS Status

Gets the status of DRS(Destructive Reset).

Parameters

<i>codeDRS</i>	the data format is [DRS SourceBlk Number] [SourceBlk1] ... [SourceBlkN] [DRS SourceBlk Number] is 2 bytes, format is NumL NumH. It is Number of [SourceBlkX] [SourceBlkX] is n bytes, Format is [SourceID] [SourceLen] [SourceData] [SourceID] is 1 byte [SourceLen] is 1 byte, it is length of [SourceData]
----------------	--

[SourceID] [SourceLen] [SourceData] 00 1 01 – Application Error 01 1 01 – Application Error 02 1 0x01 – EMV L2 Configuration Check Value Error 0x02 – Future Key Check Value Error 10 1 01 – Battery Error 11 1 Bit 0 – Tamper Switch 1 (0-No, 1-Error) Bit 1– Tamper Switch 2 (0-No, 1-Error) Bit 2– Tamper Switch 3 (0-No, 1-Error) Bit 3– Tamper Switch 4 (0-No, 1-Error) Bit 4– Tamper Switch 5 (0-No, 1-Error) Bit 5– Tamper Switch 6 (0-No, 1-Error)

12 1 01 –TemperatureHigh or Low 13 1 01 –Voltage High or Low 1F 4 Reg31~24bits, Reg23~16bits, Reg15~8bits, Reg7~0bits

Returns

RETURN_CODE: Values can be parsed with `errorCode.getErrorString()`

21.1.2.21 RETURN_CODE IDTechSDK.IDT_Augusta.device_getFirmwareVersion (ref string *response*, string *ident* = " ")

Polls device for Firmware Version

Parameters

<i>response</i>	Response returned of Firmware Version
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with `device_getResponseCodeString`

21.1.2.22 RETURN_CODE IDTechSDK.IDT_Augusta.device_getKeyFormatForICCDUKPT (ref byte *format*, string *ident* = " ")

Get DUKPT Key Format

Parameters

<i>format</i>	Format: 0 = TDES, 1 = AES
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with `device_getResponseCodeString`

21.1.2.23 RETURN_CODE IDTechSDK.IDT_Augusta.device_getKeyStatus (ref Boolean *newFormat*, ref byte[] *status*, string *ident* = " ")

Get Key Status

Gets the status of loaded keys

Parameters

<i>status</i>	newFormat true: new format of key status false: reserved format for support previous device
<i>status</i>	when the newFormat is false, data format as follows. byte 0: PIN DUKPT Key, Does not support, always 0 byte 1: PIN Master Key, Does not support, always 0 byte 2: PIN Session Key, Does not support, always 0 byte 3: Account/MSR DUKPT Key, 1 Exists, 0 None, 0xFF STOP byte 4: Account/ICC DUKPT Key, 1 Exists, 0 None, 0xFF STOP byte 5: Admin DUKPT Key, 1 Exists, 0 None, 0xFF STOP

when the newFormat is true, data format as follows. [Block Length] [KeyStatusBlock1] [KeyStatusBlock2] ... [KeyStatusBlockN] Where: [Block Length] is 2 bytes, format is Len_L Len_H, is KeyStatusBlock Number [KeyStatusBlockX] is 4 bytes, format is [Key Index and Key Name] [key slot] [key status]: [Key Index and Key Name] is 1 byte. Please refer to following table 0x14 LCL-KEK to Encrypt Other Keys 0x02 Data encryption Key to Encrypt ICC/MSR 0x05 MAC DUKPT Key for Host-Device – MAC Verification 0x05 MTK DUKPT Key for TTK Self-Test 0x0C RKI-KEK for Remote Key Injection [key slot] is 2 bytes. Range is 0 – 9999 the MTK DUKPT Key slot is 16, the others are all 0 [key status] is 1 byte. 0 – Not Exist 1 – Exist 0xFF – (Stop. Only Valid for DUKPT Key)

Returns

RETURN_CODE: Values can be parsed with `errorCode.getErrorString()`

21.1.2.24 RETURN_CODE IDTechSDK.IDT_Augusta.device_getKeyTypeForICCDUKPT (ref byte type, string ident = " ")

Get DUKPT Key Type

Parameters

<i>type</i>	Type: 0 = Data, 1 = PIN
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with `device_getResponseCodeString`

21.1.2.25 RETURN_CODE IDTechSDK.IDT_Augusta.device_getQuickChipMode (ref bool isOn, string ident = " ")

Get QuickChip Mode

Returns the QuickChip Mode Setting.

Parameters

<i>isOn</i>	TRUE = Quickchip Mode ON, FALSE = QuickChip Mode OFF
-------------	--

Returns

RETURN_CODE: Values can be parsed with `device_getResponseCodeString`

21.1.2.26 string IDTechSDK.IDT_Augusta.device_getResponseCodeString (RETURN_CODE eCode)

Get the description of response result.

Parameters

<i>eCode</i>	the response result.
--------------	----------------------

Return values

<i>the</i>	string for description of response result
------------	---

- case 0: "no error, beginning task";
- case 1: "no response from reader";
- case 2: "invalid response data";
- case 3: "time out for task or CMD";
- case 4: "wrong parameter";
- case 5: "SDK is doing MSR or ICC task";
- case 6: "SDK is doing PINPad task";
- case 7: "SDK is doing CTLS task";
- case 8: "SDK is doing EMV task";
- case 9: "SDK is doing Other task";
- case 10: "err response or data";
- case 11: "no reader attached";
- case 12: "mono audio is enabled";
- case 13: "did connection";
- case 14: "audio volume is too low";
- case 15: "task or CMD be canceled";
- case 16: "UF wrong string format";
- case 17: "UF file not found";
- case 18: "UF wrong file format";
- case 19: "Attempt to contact online host failed";
- case 20: "Attempt to perform RKI failed";
- case 0x300: "Key Type(TDES) of Session Key is not same as the related Master Key.";
- case 0x400: "Related Key was not loaded.";
- case 0x500: "Key Same.";
- case 0x501: "Key is all zero";
- case 0x502: "TR-31 format error";
- case 0x702: "PAN is Error Key.";
- case 0x705: "No Internal MSR PAN (or Internal MSR PAN is erased timeout)";
- case 0X0C01: "Incorrect Frame Tag";

- case 0X0C02: "Incorrect Frame Type";
- case 0X0C03: "Unknown Frame Type";
- case 0X0C04: "Unknown Command";
- case 0X0C05: "Unknown Sub-Command";
- case 0X0C06: "CRC Error";
- case 0X0C07: "Failed";
- case 0X0C08: "Timeout";
- case 0X0C0A: "Incorrect Parameter";
- case 0X0C0B: "Command Not Supported";
- case 0X0C0C: "Sub-Command Not Supported";
- case 0X0C0D: "Parameter Not Supported / Status Abort Command";
- case 0X0C0F: "Sub-Command Not Allowed";
- case 0X0D01: "Incorrect Header Tag";
- case 0X0D02: "Unknown Command";
- case 0X0D03: "Unknown Sub-Command";
- case 0X0D04: "CRC Error in Frame";
- case 0X0D05: "Incorrect Parameter";
- case 0X0D06: "Parameter Not Supported";
- case 0X0D07: "Mal-formatted Data";
- case 0X0D08: "Timeout";
- case 0X0D0A: "Failed / NACK";
- case 0X0D0B: "Command not Allowed";
- case 0X0D0C: "Sub-Command not Allowed";
- case 0X0D0D: "Buffer Overflow (Data Length too large for reader buffer)";
- case 0X0D0E: "User Interface Event";
- case 0X0D11: "Communication type not supported, VT-1, burst, etc.";
- case 0X0D12: "Secure interface is not functional or is in an intermediate state.";
- case 0X0D13: "Data field is not mod 8";
- case 0X0D14: "Pad 0x80 not found where expected";
- case 0X0D15: "Specified key type is invalid";
- case 0X0D1: "Could not retrieve key from the SAM(InitSecureComm)";
- case 0X0D17: "Hash code problem";
- case 0X0D18: "Could not store the key into the SAM(InstallKey)";
- case 0X0D19: "Frame is too large";
- case 0X0D1A: "Unit powered up in authentication state but POS must resend the InitSecureComm command";

- case 0X0D1B: "The EEPROM may not be initialized because SecCommInterface does not make sense";
- case 0X0D1C: "Problem encoding APDU";
- case 0X0D20: "Unsupported Index(ILM) SAM Transceiver error – problem communicating with the SAM(Key Mgr)";
- case 0X0D2: "Unexpected Sequence Counter in multiple frames for single bitmap(ILM) Length error in data returned from the SAM(Key Mgr)";
- case 0X0D22: "Improper bit map(ILM)";
- case 0X0D23: "Request Online Authorization";
- case 0X0D24: "ViVOCard3 raw data read successful";
- case 0X0D25: "Message index not available(ILM) ViVOcomm activate transaction card type(ViVOcomm)";
- case 0X0D26: "Version Information Mismatch(ILM)";
- case 0X0D27: "Not sending commands in correct index message index(ILM)";
- case 0X0D28: "Time out or next expected message not received(ILM)";
- case 0X0D29: "ILM languages not available for viewing(ILM)";
- case 0X0D2A: "Other language not supported(ILM)";
- case 0X0D41: "Unknown Error from SAM";
- case 0X0D42: "Invalid data detected by SAM";
- case 0X0D43: "Incomplete data detected by SAM";
- case 0X0D44: "Reserved";
- case 0X0D45: "Invalid key hash algorithm";
- case 0X0D46: "Invalid key encryption algorithm";
- case 0X0D47: "Invalid modulus length";
- case 0X0D48: "Invalid exponent";
- case 0X0D49: "Key already exists";
- case 0X0D4A: "No space for new RID";
- case 0X0D4B: "Key not found";
- case 0X0D4C: "Crypto not responding";
- case 0X0D4D: "Crypto communication error";
- case 0X0D4E: "Module-specific error for Key Manager";
- case 0X0D4F: "All key slots are full (maximum number of keys has been installed)";
- case 0X0D50: "Auto-Switch OK";
- case 0X0D51: "Auto-Switch failed";
- case 0X0D90: "Account DUKPT Key not exist";
- case 0X0D91: "Account DUKPT Key KSN exhausted";
- case 0x0D00: "This Key had been loaded.";
- case 0x0E00: "Base Time was loaded.";
- case 0x0F00: "Encryption Or Decryption Failed.";

- case 0x1000: "Battery Low Warning (It is High Priority Response while Battery is Low.);";
- case 0x1800: "Send "Cancel Command" after send "Get Encrypted PIN" &"Get Numeric "& "Get Amount""";
- case 0x1900: "Press "Cancel" key after send "Get Encrypted PIN" &"Get Numeric "& "Get Amount""";
- case 0x30FF: "Security Chip is not connect";
- case 0x3000: "Security Chip is deactivation & Device is In Removal Legally State.";
- case 0x3101: "Security Chip is activation & Device is In Removal Legally State.";
- case 0x5500: "No Admin DUKPT Key.";
- case 0x5501: "Admin DUKPT Key STOP.";
- case 0x5502: "Admin DUKPT Key KSN is Error.";
- case 0x5503: "Get Authentication Code1 Failed.";
- case 0x5504: "Validate Authentication Code Error.";
- case 0x5505: "Encrypt or Decrypt data failed.";
- case 0x5506: "Not Support the New Key Type.";
- case 0x5507: "New Key Index is Error.";
- case 0x5508: "Step Error.";
- case 0x5509: "KSN Error";
- case 0x550A: "MAC Error.";
- case 0x550B: "Key Usage Error.";
- case 0x550C: "Mode Of Use Error.";
- case 0x550F: "Other Error.";
- case 0x6000: "Save or Config Failed / Or Read Config Error.";
- case 0x6200: "No Serial Number.";
- case 0x6900: "Invalid Command - Protocol is right, but task ID is invalid.";
- case 0x6A01: "Unsupported Command – Protocol and task ID are right, but command is invalid – In this State";
- case 0x6A00: "Unsupported Command - Protocol and task ID are right, but command is invalid.";
- case 0x6B00: "Unknown parameter in command - Protocol task ID and command are right, but parameter is invalid.";
- case 0x6C00: "Unknown parameter in command – Protocol task ID and command are right, but length is out of the requirement.";
- case 0x7200: "Device is suspend (MKS suspend or press password suspend).";
- case 0x7300: "PIN DUKPT is STOP (21 bit 1).";
- case 0x7400: "Device is Busy.";
- case 0xE100: "Can not enter sleep mode";
- case 0xE200: "File has existed";
- case 0xE300: "File has not existed";
- case 0xE313: "IO line low -- Card error after session start";

- case 0xE400: "Open File Error";
- case 0xE500: "SmartCard Error";
- case 0xE600: "Get MSR Card data is error";
- case 0xE700: "Command time out";
- case 0xE800: "File read or write is error";
- case 0xE900: "Active 1850 error!";
- case 0xEA00: "Load bootloader error";
- case 0xEF00: "Protocol Error- STX or ETX or check error.";
- case 0xEB00: "Picture is not exist";
- case 0x2C02: "No Microprocessor ICC seated";
- case 0x2C06: "no card seated to request ATR";
- case 0x2D01: "Card Not Supported,";
- case 0x2D03: "Card Not Supported, wants CRC";
- case 0x690D: "Command not supported on reader without ICC support";
- case 0x8100: "ICC error time out on power-up";
- case 0x8200: "invalid TS character received - Wrong operation step";
- case 0x8300: "Decode MSR Error";
- case 0x8400: "TriMagII no Response";
- case 0x8500: "No Swipe MSR Card";
- case 0x8510: "No Financial Card";
- case 0x8600: "Unsupported F, D, or combination of F and D";
- case 0x8700: "protocol not supported EMV TD1 out of range";
- case 0x8800: "power not at proper level";
- case 0x8900: "ATR length too long";
- case 0x8B01: "EMV invalid TA1 byte value";
- case 0x8B02: "EMV TB1 required";
- case 0x8B03: "EMV Unsupported TB1 only 00 allowed";
- case 0x8B04: "EMV Card Error, invalid BWI or CWI";
- case 0x8B06: "EMV TB2 not allowed in ATR";
- case 0x8B07: "EMV TC2 out of range";
- case 0x8B08: "EMV TC2 out of range";
- case 0x8B09: "per EMV96 TA3 must be > 0xF";
- case 0x8B10: "ICC error on power-up";
- case 0x8B11: "EMV T=1 then TB3 required";
- case 0x8B12: "Card Error, invalid BWI or CWI";
- case 0x8B13: "Card Error, invalid BWI or CWI";

- case 0x8B17: "EMV TC1/TB3 conflict-";
- case 0x8B20: "EMV TD2 out of range must be T=1";
- case 0x8C00: "TCK error";
- case 0xA304: "connector has no voltage setting";
- case 0xA305: "ICC error on power-up invalid (SBLK(IFSD) exchange";
- case 0xE301: "ICC error after session start";
- case 0xFF00: "Request to go online";
- case 0xFF01: "EMV: Accept the offline transaction";
- case 0xFF02: "EMV: Decline the offline transaction";
- case 0xFF03: "EMV: Accept the online transaction";
- case 0xFF04: "EMV: Decline the online transaction";
- case 0xFF05: "EMV: Application may fallback to magstripe technology";
- case 0xFF06: "EMV: ICC detected tah the conditions of use are not satisfied";
- case 0xFF07: "EMV: ICC didn't accept transaction";
- case 0xFF08: "EMV: Transaction was cancelled";
- case 0xFF09: "EMV: Application was not selected by kernel or ICC format error or ICC missing data error";
- case 0xFF0A: "EMV: Transaction is terminated";
- case 0xFF0B: "EMV: Other EMV Error";
- case 0xFFFF: "NO RESPONSE";
- case 0xF002: "ICC communication timeout";
- case 0xF003: "ICC communication Error";
- case 0xF00F: "ICC Card Seated and Highest Priority, disable MSR work request";
- case 0xF200: "AID List / Application Data is not exist";
- case 0xF201: "Terminal Data is not exist";
- case 0xF202: "TLV format is error";
- case 0xF203: "AID List is full";
- case 0xF204: "Any CA Key is not exist";
- case 0xF205: "CA Key RID is not exist";
- case 0xF206: "CA Key Index it not exist";
- case 0xF207: "CA Key is full";
- case 0xF208: "CA Key Hash Value is Error";
- case 0xF209: "Transaction format error";
- case 0xF20A: "The command will not be processing";
- case 0xF20B: "CRL is not exist";
- case 0xF20C: "CRL number exceed max number";
- case 0xF20D: "Amount,Other Amount,Trasaction Type are missing";

- case 0xF20E: "The Identification of algorithm is mistake";
- case 0xF20F: "No Financial Card";
- case 0xF210: "In Encrypt Result state, TLV total Length is greater than Max Length";
- case 0x1001: "INVALID ARG";
- case 0x1002: "FILE_OPEN_FAILED";
- case 0x1003: "FILE OPERATION_FAILED";
- case 0x2001: "MEMORY_NOT_ENOUGH";
- case 0x3002: "SMARTCARD_FAIL";
- case 0x3003: "SMARTCARD_INIT_FAILED";
- case 0x3004: "FALLBACK_SITUATION";
- case 0x3005: "SMARTCARD_ABSENT";
- case 0x3006: "SMARTCARD_TIMEOUT";
- case 0x5001: "EMV_PARSING_TAGS_FAILED";
- case 0x5002: "EMV_DUPLICATE_CARD_DATA_ELEMENT";
- case 0x5003: "EMV_DATA_FORMAT_INCORRECT";
- case 0x5004: "EMV_NO_TERM_APP";
- case 0x5005: "EMV_NO_MATCHING_APP";
- case 0x5006: "EMV_MISSING_MANDATORY_OBJECT";
- case 0x5007: "EMV_APP_SELECTION_RETRY";
- case 0x5008: "EMV_GET_AMOUNT_ERROR";
- case 0x5009: "EMV_CARD_REJECTED";
- case 0x5010: "EMV_AIP_NOT_RECEIVED";
- case 0x5011: "EMV_AFL_NOT_RECEIVED";
- case 0x5012: "EMV_AFL_LEN_OUT_OF_RANGE";
- case 0x5013: "EMV_SFI_OUT_OF_RANGE";
- case 0x5014: "EMV_AFL_INCORRECT";
- case 0x5015: "EMV_EXP_DATE_INCORRECT";
- case 0x5016: "EMV_EFF_DATE_INCORRECT";
- case 0x5017: "EMV_ISS_COD_TBL_OUT_OF_RANGE";
- case 0x5018: "EMV_CRYPTOGAM_TYPE_INCORRECT";
- case 0x5019: "EMV_PSE_NOT_SUPPORTED_BY_CARD";
- case 0x5020: "EMV_USER_SELECTED_LANGUAGE";
- case 0x5021: "EMV_SERVICE_NOT_ALLOWED";
- case 0x5022: "EMV_NO_TAG_FOUND";
- case 0x5023: "EMV_CARD_BLOCKED";
- case 0x5024: "EMV_LEN_INCORRECT";

- case 0x5025: "CARD_COM_ERROR";
- case 0x5026: "EMV_TSC_NOT_INCREASED";
- case 0x5027: "EMV_HASH_INCORRECT";
- case 0x5028: "EMV_NO_ARC";
- case 0x5029: "EMV_INVALID_ARC";
- case 0x5030: "EMV_NO_ONLINE_COMM";
- case 0x5031: "TRAN_TYPE_INCORRECT";
- case 0x5032: "EMV_APP_NO_SUPPORT";
- case 0x5033: "EMV_APP_NOT_SELECT";
- case 0x5034: "EMV_LANG_NOT_SELECT";
- case 0x5035: "EMV_NO_TERM_DATA";
- case 0x6001: "CVM_TYPE_UNKNOWN";
- case 0x6002: "CVM_AIP_NOT_SUPPORTED";
- case 0x6003: "CVM_TAG_8E_MISSING";
- case 0x6004: "CVM_TAG_8E_FORMAT_ERROR";
- case 0x6005: "CVM_CODE_IS_NOT_SUPPORTED";
- case 0x6006: "CVM_COND_CODE_IS_NOT_SUPPORTED";
- case 0x6007: "NO_MORE_CVM";
- case 0x6008: "PIN_BYPASSED_BEFORE";
- case 0x7001: "PK_BUFFER_SIZE_TOO_BIG";
- case 0x7002: "PK_FILE_WRITE_ERROR";
- case 0x7003: "PK_HASH_ERROR";
- case 0x8001: "NO_CARD_HOLDER_CONFIRMATION";
- case 0x8002: "GET_ONLINE_PIN";
- case 0xD000: "Data not exist";
- case 0xD001: "Data access error";
- case 0xD100: "RID not exist";
- case 0xD101: "RID existed";
- case 0xD102: "Index not exist";
- case 0xD200: "Maximum exceeded";
- case 0xD201: "Hash error";
- case 0xD205: "System Busy";
- case 0x0E01: "Unable to go online";
- case 0x0E02: "Technical Issue";
- case 0x0E03: "Declined";
- case 0x0E04: "Issuer Referral transaction";

- case 0x0F01: "Decline the online transaction";
- case 0x0F02: "Request to go online";
- case 0x0F03: "Transaction is terminated";
- case 0x0F05: "Application was not selected by kernel or ICC format error or ICC missing data error";
- case 0x0F07: "ICC didn't accept transaction";
- case 0x0F0A: "Application may fallback to magstripe technology";
- case 0x0F0C: "Transaction was cancelled";
- case 0x0F0D: "Timeout";
- case 0x0F0F: "Other EMV Error";
- case 0x0F10: "Accept the offline transaction";
- case 0x0F11: "Decline the offline transaction";
- case 0x0F21: "ICC detected tah the conditions of use are not satisfied";
- case 0x0F22: "No app were found on card matching terminal configuration";
- case 0x0F23: "Terminal file does not exist";
- case 0x0F24: "CAPK file does not exist";
- case 0x0F25: "CRL Entry does not exist";
- case 0x0FFE: "code when blocking is disabled";
- case 0x0FFF: "code when command is not applicable on the selected device";
- case 0xF005: "ICC Encrypted C-APDU Data Structure Length Error Or Format Error.";
- case 0xBBE0: "CM100 Success";
- case 0xBBE1: "CM100 Parameter Error";
- case 0xBBE2: "CM100 Low Output Buffer";
- case 0xBBE3: "CM100 Card Not Found";
- case 0xBBE4: "CM100 Collision Card Exists";
- case 0xBBE5: "CM100 Too Many Cards Exist";
- case 0xBBE6: "CM100 Saved Data Does Not Exist";
- case 0xBBE8: "CM100 No Data Available";
- case 0xBBE9: "CM100 Invalid CID Returned";
- case 0xBBEA: "CM100 Invalid Card Exists";
- case 0xBBEC: "CM100 Command Unsupported";
- case 0xBBED: "CM100 Error In Command Process";
- case 0xBBEE: "CM100 Invalid Command";
- case 0X9031: "Unknown command";
- case 0X9032: "Wrong parameter (such as the length of the command is incorrect)";
- case 0X9038: "Wait (the command couldn't be finished in BWT)";
- case 0X9039: "Busy (a previously command has not been finished)";

- case 0X903A: "Number of retries over limit";
- case 0X9040: "Invalid Manufacturing system data";
- case 0X9041: "Not authenticated";
- case 0X9042: "Invalid Master DUKPT Key";
- case 0X9043: "Invalid MAC Key";
- case 0X9044: "Reserved for future use";
- case 0X9045: "Reserved for future use";
- case 0X9046: "Invalid DATA DUKPT Key";
- case 0X9047: "Invalid PIN Pairing DUKPT Key";
- case 0X9048: "Invalid DATA Pairing DUKPT Key";
- case 0X9049: "No nonce generated";
- case 0X9949: "No GUID available. Perform getVersion first.";
- case 0X9950: "MAC Calculation unsuccessful. Check BDK value.";
- case 0X904A: "Not ready";
- case 0X904B: "Not MAC data";
- case 0X9050: "Invalid Certificate";
- case 0X9051: "Duplicate key detected";
- case 0X9052: "AT checks failed";
- case 0X9053: "TR34 checks failed";
- case 0X9054: "TR31 checks failed";
- case 0X9055: "MAC checks failed";
- case 0X9056: "Firmware download failed";
- case 0X9060: "Log is full";
- case 0X9061: "Removal sensor unengaged";
- case 0X9062: "Any hardware problems";
- case 0X9070: "ICC communication timeout";
- case 0X9071: "ICC data error (such check sum error)";
- case 0X9072: "Smart Card not powered up";

21.1.2.27 **RETURN_CODE** IDTechSDK.IDT_Augusta.device_getRKIStatus (bool *isProd*, bool *isMultiKey*, ref string *status*, ref Dictionary< string, RKI_KEY_TYPE > *keys*, string *ident* = " ")

Get RKI Status From Specified Server Polls the RKI server to validate device RKI eligibility

Parameters

<i>isProd</i>	TRUE = poll production RKI server, FALSE = poll demo RKI server
<i>isMultiKey</i>	True = MultiKey, False = SingleKey
<i>status</i>	RKI status returned as a string
<i>keys</i>	Returns dictionary of available keys

IDTech Android SDK Device ID Not Specified and to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with `errorCode.getErrorString()`. When no data is available, return code = RETURN_CODE_NO_DATA_AVAILABLE

21.1.2.28 RETURN_CODE IDTechSDK.IDT_Augusta.device_getTransArmorID (ref string *TID*, string *ident* = " ")

Get TransArmor ID

Parameters

<i>TID</i>	TransArmor ID
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with `device_getResponseCodeString`

21.1.2.29 bool IDTechSDK.IDT_Augusta.device_isSRED (string *ident* = " ")

Check if the device support SRED feature.

Parameters

<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.
--------------	--

Returns

Success flag. true is with SRED.

21.1.2.30 RETURN_CODE IDTechSDK.IDT_Augusta.device_readConfigurationToMemory (string *memo*, Configs.InstallRules *rules*, List< Configs.DeviceCommand > *cmds*, string *ident* = " ", bool *isForeground* = false)

Read Device Configuration To Memory Executes a ViVOconfig read of device settings and stores the results in memory

Once a ViVOconfig read starts, all messages will be returned to the callback as `DeviceState.ViVOconfig`. The ViVOconfig operation is complete when the return code from a `DeviceState.ViVOconfig` message is one of the following values:

- RETURN_CODE_SDK_VIVOCONFIG_READ_CONFIG_SUCCESS = Read process completed successfully. Execute `device_getConfigurationFromMemory` to retrieve the configuration data
- RETURN_CODE_SDK_VIVOCONFIG_READ_CONFIG_FAILED = Read process FAILED.

Parameters

<i>memo</i>	Optional: populates the memo field of the captured .json file
<i>rules</i>	Optional: populates the .json file with available install rules
<i>cmds</i>	Optional: populates the .json file with commands to execute
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.
<i>isForeground</i>	If TRUE, will block program until update complete. Otherwise, FALSE performs update on background.

Returns

RETURN_CODE: Values can be parsed with `errorCode.getErrorString()`. When no data is available, return code = RETURN_CODE_NO_DATA_AVAILABLE

21.1.2.31 RETURN_CODE IDTechSDK.IDT_Augusta.device_rebootDevice (string *ident* = " ")

Reboot Device

Executes a command to restart the device.

Parameters

<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.
--------------	--

Returns

RETURN_CODE: Values can be parsed with `device_getResponseCodeString`

21.1.2.32 RETURN_CODE IDTechSDK.IDT_Augusta.device_RemoteKeyInjection (RKI_KEY_TYPE *type*, string *keyName*, string *ident* = " ", bool *performOnForeground* = false)

Remote Key Injection Performs a remote key injection for the device

Parameters

<i>type</i>	Remote Key Injection Type
<i>keyName</i>	Name of key (optional)
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.
<i>performOnForeground</i>	TRUE = block during process (default is false)

Returns

RETURN_CODE: Values can be parsed with `errorCode.getErrorString()`. When no data is available, return code = RETURN_CODE_NO_DATA_AVAILABLE

21.1.2.33 RETURN_CODE IDTechSDK.IDT_Augusta.device_selfCheck (string *ident* = " ")

Self check for TTK If Self-Test function Failed, then work into De-activation State. If device work into De-activation State, All Sensitive Data will be erased and it need be fixed in Manufacture.

Parameters

<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.
--------------	--

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString

21.1.2.34 RETURN_CODE IDTechSDK.IDT_Augusta.device_sendConfiguration (string filename, VC_OPERATION_TYPE type, bool matchFW, string ident = "", bool isForeground = false)

Send Configuration Executes a ViVOconfig update or verify to a device

Once a ViVOconfig write or verify starts, all messages will be returned to the callback as DeviceState.ViVOconfig. The ViVOconfig operation is complete when the return code from a DeviceState.ViVOconfig message is one of the following values:

- RETURN_CODE_SDK_VIVOCONFIG_VERIFY_SUCCESS = Verification process completed successfully. No differences found.
- RETURN_CODE_SDK_VIVOCONFIG_VERIFY_SUCCESS_WITH_WARNING = Verification process completed with warnings.
- RETURN_CODE_SDK_VIVOCONFIG_VERIFY_FAILED = Verification process FAILED
- RETURN_CODE_SDK_VIVOCONFIG_WRITE_SUCCESS = Write process completed successfully.
- RETURN_CODE_SDK_VIVOCONFIG_WRITE_SUCCESS_WITH_WARNING = Write process completed with warnings
- RETURN_CODE_SDK_VIVOCONFIG_WRITE_FAILED = Write process FAILED

Parameters

<i>filename</i>	The .json configuration file to update the device with NOTE: You can also use this parameter to pass the path of where the .json file is located at on local storage instead
<i>filename</i>	The name of the .json configuration file
<i>type</i>	VC_OPERATION_TYPE

- VC_OPERATION_TYPE_WRITE = Write File To Device, Hash must validate
- VC_OPERATION_TYPE_VERIFY = Verify Device With File
- VC_OPERATION_TYPE_WRITE_IGNORE_HASH = Write File To Device, Ignore Hash
- VC_OPERATION_TYPE_WRITE_FIX_HASH = Write File To Device, Fix Hash If Necessary

Parameters

<i>matchFW</i>	TRUE = Device FW must match file FW, FALSE = Don't validate device FW against file
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.
<i>isForeground</i>	If TRUE, will block program until update complete. Otherwise, FALSE performs update on background.

Returns

RETURN_CODE: Values can be parsed with `errorCode.getErrorString()`. When no data is available, return code = RETURN_CODE_NO_DATA_AVAILABLE

21.1.2.35 RETURN_CODE IDTechSDK.IDT_Augusta.device_sendConfigurationFromZip (byte[] *zip*, string *filename*, VC_OPERATION_TYPE *type*, bool *matchFW*, string *ident* = "", bool *isForeground* = false)

Send Configuration From Zip Executes a ViVOconfig update or verify to a device using files from a .zip archive

Once a ViVOconfig write or update starts, all messages will be returned to the callback as DeviceState.ViVOconfig. The ViVOconfig operation is complete when the return code from a DeviceState.ViVOconfig message is one of the following values:

- RETURN_CODE_SDK_VIVOCONFIG_VERIFY_SUCCESS = Verification process completed successfully. No differences found.
- RETURN_CODE_SDK_VIVOCONFIG_VERIFY_SUCCESS_WITH_WARNING = Verification process completed with warnings.
- RETURN_CODE_SDK_VIVOCONFIG_VERIFY_FAILED = Verification process FAILED
- RETURN_CODE_SDK_VIVOCONFIG_WRITE_SUCCESS = Write process completed successfully.
- RETURN_CODE_SDK_VIVOCONFIG_WRITE_SUCCESS_WITH_WARNING = Write process completed with warnings
- RETURN_CODE_SDK_VIVOCONFIG_WRITE_FAILED = Write process FAILED

Parameters

<i>zip</i>	.Zip archive containing the .json configuration file and also any associated image assets NOTE: You can also use this parameter to pass the path of where the .zip file is located at on local storage instead
<i>filename</i>	The name of the .json configuration file
<i>type</i>	VC_OPERATION_TYPE

- VC_OPERATION_TYPE_WRITE = Write File To Device, Hash must validate
- VC_OPERATION_TYPE_VERIFY = Verify Device With File
- VC_OPERATION_TYPE_WRITE_IGNORE_HASH = Write File To Device, Ignore Hash
- VC_OPERATION_TYPE_WRITE_FIX_HASH = Write File To Device, Fix Hash If Necessary

Parameters

<i>matchFW</i>	TRUE = Device FW must match file FW, FALSE = Don't validate device FW against file
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.
<i>isForeground</i>	If TRUE, will block program until update complete. Otherwise, FALSE performs update on background.

Returns

RETURN_CODE: Values can be parsed with `errorCode.getErrorString()`. When no data is available, return code = RETURN_CODE_NO_DATA_AVAILABLE

21.1.2.36 **RETURN_CODE** IDTechSDK.IDT_Augusta.device_sendDataCommand (string *cmd*, bool *calcLRC*, ref byte[] *response*, string *ident* = " ")

Send a data command to the device

Sends a command to the device.

Parameters

<i>cmd</i>	String representation of command to execute
<i>calcLRC</i>	If TRUE , this will wrap command with start/length/lrc/sum/end: '{STX}{Len_Low}{Len_High} data {CheckLRC} {CheckSUM} {ETX}'
<i>response</i>	Response data
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString

21.1.2.37 **RETURN_CODE** IDTechSDK.IDT_Augusta.device_sendDataCommand_ext (string *cmd*, bool *calcLRC*, ref byte[] *response*, int *timeout*, bool *noResponse*, string *ident* = " ")

Send a data command to the device - extended

Sends a command to the device.

Parameters

<i>cmd</i>	String representation of command to execute
<i>calcLRC</i>	If TRUE , this will wrap command as NGA with start/length/lrc/sum/end: '{STX}{Len_Low}{Len_High} data {CheckLRC} {CheckSUM} {ETX}'
<i>response</i>	Response data
<i>timeout</i>	Timeout value waiting for response, in milliseconds (1000 = 1 second)
<i>noResponse</i>	if TRUE , this will not wait for a response and immediately return SUCCESS
<i>calcITP</i>	If TRUE , this will wrap command as ITP with start/end/lrc: '{STX} data {ETX}{CheckLRC}'
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString

21.1.2.38 **RETURN_CODE** IDTechSDK.IDT_Augusta.device_sendPAE (string *command*, ref string *response*, int *timeout*, string *ident* = " ")

Send Payment Application Engine Command

Executes a PAE command

Parameters

<i>command</i>	ASCII command string, should start with "*PAE"
<i>response</i>	command response
<i>timeout</i>	timeout waiting for PAE response
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString

21.1.2.39 RETURN_CODE IDTechSDK.IDT_Augusta.device_setDateTime (string *ident* = " ")**Set Date Time**

Set current system date and time to the device.

Parameters

<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.
--------------	--

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString

21.1.2.40 RETURN_CODE IDTechSDK.IDT_Augusta.device_setQuickChipMode (bool *turnON*, string *ident* = " ")**Set QuickChip Mode**

Sets QuickChip Mode. This tells the firmware to automatically complete a transaction with "Z3"

Parameters

<i>turnON</i>	TRUE = Turn Quickchip Mode ON, FALSE = Turn QuickChip Mode OFF
---------------	--

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString

21.1.2.41 RETURN_CODE IDTechSDK.IDT_Augusta.device_setTransArmorEncryption (byte[] *cert*, string *ident* = " ")**Set TransArmor Encryption****Parameters**

<i>cert</i>	Certificate in PEM format or DER format. PEM format must be string data (converted to binary) starting with "----". DER format is binary data
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString

21.1.2.42 RETURN_CODE IDTechSDK.IDT_Augusta.device_setTransArmorID (string *TID*, string *ident* = " ")**Set TransArmor ID**

Parameters

<i>TID</i>	TransArmor ID
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString

21.1.2.43 RETURN_CODE IDTechSDK.IDT_Augusta.device_StartRKI (int *type*, string *ident* = " ")

Start Remote Key Injection

Starts a remote key injection request with IDTech RKI servers. Set/Get RKI url with IDT_Device.RKI_URL.

Parameters

<i>type</i>	0 = Symmetric RKI Demo Unit 1 = Symmetric RKI Production Unit 2 = PKI-RKI Demo Unit 3 = PKI-RKI Production Unit
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString

21.1.2.44 RETURN_CODE IDTechSDK.IDT_Augusta.device_startRKI (string *ident* = " ")

Start Remote Key Injection

Starts a remote key injection request with IDTech RKI servers.

Parameters

<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used
--------------	---

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString

21.1.2.45 RETURN_CODE IDTechSDK.IDT_Augusta.device_startTransaction (double *amount*, double *amtOther*, int *exponent*, int *type*, int *timeout*, byte[] *tags*, bool *forceOnline*, bool *isFastEMV* = false, string *ident* = " ")

Start EMV Transaction Request

Authorizes the EMV transaction for an ICC card

The tags will be returned in the callback routine.

Parameters

<i>amount</i>	Transaction amount value (tag value 9F02)
<i>amtOther</i>	Other amount value, if any (tag value 9F03)
<i>exponent</i>	Number of characters after decimile point

Parameters

<i>type</i>	Transaction type (tag value 9C).
<i>timeout</i>	Timeout value in seconds.
<i>tags</i>	Any other tags to be included in the request. Passed as a TLV data. Example, tag 9F02 with amount 0x000000000100 would be 0x9F0206000000000100 If tags 9F02 (amount), 9F03 (other amount), or 9C (transaction type) are included, they will take priority over these values supplied as individual parameters to this method.
<i>forceOnline</i>	TRUE = do not allow offline approval, FALSE = allow ICC to approve offline if terminal capable Note: To request tags to be included in default response, use tag DFEE1A, and specify tag list. Example four tags 9F02, 9F36, 95, 9F37 to be included in response = DFEE1A079F029F369f9F37
<i>isFastEMV</i>	If TRUE, it will populate the IDTTransactionData.fastEMV with ASCII data similar to IDTech FastEMV KB output, after performing an auto-authenticate and auto-complete with ResultCode = Could Not Contact Host
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString

```
21.1.2.46 static RETURN_CODE IDTechSDK.IDT_Augusta.device_updateDeviceFirmware ( byte[] firmwareData, string ident = "" ) [static]
```

Update Firmware

Updates the firmware .

Parameters

<i>firmwareData</i>	Signed binary data of a firmware file provided by IDTech
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with errorCode.getErrorString()

After you pass the firmwareData file, a new thread will start to execute the firmware download. You will receive status of the progress through callbacks to the IDTechSDK.Callback() delegate. The following parameters will be passed back:

- sender = IDT_DEVICE_Types.IDT_DEVICE_AUGUSTA
- state = DeviceState.FirmwareUpdate
- transactionResultCode = status of the firmware update (starting, entering bootloader, applying update, block success, firmware success)
- data = File Progress. Four bytes, with bytes [0][1] = current block, and bytes [2][3] = total blocks. 0x00030010 = block 3 of 16

Example code starting a firmware update

```
OpenFileDialog diag = new OpenFileDialog();
diag.Filter = "NGA FW Files|*.fm";
```

```

if (diag.ShowDialog() == DialogResult.OK)
{
    byte[] file = File.ReadAllBytes(diag.FileName);
    RETURN_CODE rt = IDT_Device.SharedController.device_updateDeviceFirmware(file);
    if (rt == RETURN_CODE.RETURN_CODE_DO_SUCCESS)
    {
        //Was a success
    }
    else
    {
        //Error starting firmware download
    }
}

```

Example monitoring firmware update status / success

```

private void MessageCallBack(IDTechSDK.IDT_DEVICE_Types type, DeviceState state, byte[] data,
    IDTTransactionData cardData, EMV_Callback emvCallback, RETURN_CODE transactionResultCode)
{
    switch (state)
    {
        case DeviceState.FirmwareUpdate:
            switch (transactionResultCode)
            {
                case RETURN_CODE.RETURN_CODE_FW_STARTING_UPDATE:
                    SetOutputText("Starting Firmware Update\n");
                    break;
                case RETURN_CODE.RETURN_CODE_DO_SUCCESS:
                    SetOutputText("Firmware Update Successful\n");
                    break;
                case RETURN_CODE.RETURN_CODE_APPLYING_FIRMWARE_UPDATE:
                    SetOutputText("Applying Firmware Update...\n");
                    break;
                case RETURN_CODE.RETURN_CODE_ENTERING_BOOTLOADER_MODE:
                    SetOutputText("Entering Bootloader Mode...\n");
                    break;
                case RETURN_CODE.RETURN_CODE_BLOCK_TRANSFER_SUCCESS:
                    int start = data[0] * 0x100 + data[1];
                    int end = data[2] * 0x100 + data[3];

                    SetOutputText("Sent block " + start.ToString() + " of " + end.ToString() + "\n");
                    break;
                default:
                    SetOutputText("Firmware Update Error Code: " + "0x" + String.Format("{0:X}", (ushort)
                        transactionResultCode) + ": " + IDTechSDK.errorCode.getErrorString(transactionResultCode) + "\r\n");
                    break;
            }
            break;
    }
}

```

21.1.2.47 RETURN_CODE IDTechSDK.IDT_Augusta.device_updateFirmwareFromZip (byte[] zipfile, string ident = "", bool isForeground = false)

Update Firmware From Zip

Performs one or more device firmware updates from firmware files passed as a compressed archive with a valid configuration file (.json format)

Parameters

<i>zipfile</i>	Zip file containing on or more firmware update files and a .json configuration file
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.
<i>isForeground</i>	If TRUE, will block program until firmware update complete. Otherwise, FALSE performs FW update on background.

Returns

RETURN_CODE: Values can be parsed with `errorCode.getErrorString(string ident = "")`

21.1.2.48 RETURN_CODE IDTechSDK.IDT_Augusta.device_verifyBackdoorKey (string *ident* = " ")

Verify Backdoor Key to Unlock Security

For special firmware only (TTK)

Parameters

<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.
--------------	--

Returns

RETURN_CODE: Values can be parsed with `device_getResponseCodeString`

21.1.2.49 RETURN_CODE IDTechSDK.IDT_Augusta.emv_activateTransaction (int *timeout*, byte[] *tags*, bool *forceOnline*, bool *isFastEMV* = false, string *ident* = " ")

Start EMV Transaction Request

Authorizes the EMV transaction for an ICC card

The tags will be returned in the callback routine.

Parameters

<i>timeout</i>	Timeout value in seconds.
<i>tags</i>	The tags to be included in the request. Passed as a TLV data. Example, tag 9F02 with amount 0x000000000100 would be 0x9F0206000000000100 If tags 9F02 (amount), 9F03 (other amount), or 9C (transaction type) are included, they will take priority over these values
<i>forceOnline</i>	TRUE = do not allow offline approval, FALSE = allow ICC to approve offline if terminal capable Note: To request tags to be included in default response, use tag DFEE1A, and specify tag list. Example four tags 9F02, 9F36, 95, 9F37 to be included in response = DFEE1A079F029F369f9F37
<i>isFastEMV</i>	If TRUE, it will populate the IDTTransactionData.fastEMV with ASCII data similar to IDTech FastEMV KB output, after performing an auto-authenticate and auto-complete with ResultCode = Could Not Contact Host
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with `device_getResponseCodeString`

21.1.2.50 RETURN_CODE IDTechSDK.IDT_Augusta.emv_addTerminalData (byte[] *tlv*, string *ident* = " ")

Add Terminal Data

Adds to the exosting Terminal Data

Parameters

<i>tlv</i>	TerminalData TLV Data
<i>ident</i>	Optional identifier

Return values

<i>RETURN_CODE</i>	Return codes listed as typedef enum in IDTCommon:RETURN_CODE. Values can be parsed with IDT_BTPay::device_getResponseCodeString:()
--------------------	--

21.1.2.51 `static void IDTechSDK.IDT_Augusta.emv_allowFallback (bool allow, string ident = " ") [static]`

Allow fallback for EMV transactions. Default is TRUE

Parameters

<i>allow</i>	TRUE = allow fallback, FALSE = don't allow fallback
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

21.1.2.52 `RETURN_CODE IDTechSDK.IDT_Augusta.emv_authenticateTransaction (byte[] updatedTLV, string ident = " ")`

Authenticate EMV Transaction Request

Authenticates the EMV transaction for an ICC card. Execute this after receiving response with result code 0x10 to `emv_startTransaction`

The tags will be returned in the callback routine.

Parameters

<i>updatedTLV</i>	TLV stream that can be used to update the following values: <ul style="list-style-type: none"> • 9F02: Amount • 9F03: Other amount • 9C: Transaction type • 5F57: Account type In addition tag DFEE1A can be sent to specify tag list to include in results. Example four tags 9F02, 9F36, 95, 9F37 to be included in response = DFEE1A079F029F369f9F37
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with `device_getResponseCodeString`

21.1.2.53 `static void IDTechSDK.IDT_Augusta.emv_autoAuthenticate (bool authenticate, string ident = " ") [static]`

Enables authenticate for EMV transactions. If a `startEMVTransaction` results in code 0x0010 (start transaction success),

Parameters

<i>authenticate</i>	TRUE = auto authenticate, FALSE = manually authenticate
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

21.1.2.54 **static void** IDTechSDK.IDT_Augusta.emv_autoAuthenticateTags (*bool authenticate*, *byte[] tags*, *string ident* = " ")
[static]

Enables authenticate for EMV transactions. If a startEMVTransaction results in code 0x0010 (start transaction success), then emv_authenticateEMVTransaction can automatically execute if parameter is set to TRUE

Parameters

<i>authenticate</i>	TRUE = auto authenticate, FALSE = manually authenticate
<i>tags</i>	Tags to pass during authentication stage;
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

21.1.2.55 **RETURN_CODE** IDTechSDK.IDT_Augusta.emv_callbackResponseLCD (*EMV_LCD_DISPLAY_MODE type*, *byte selection*, *string ident* = " ")

Callback Response LCD Display

Provides menu selection responses to the kernel after a callback was received with DeviceState.EMVCallback, and callbackType = EMV_CALLBACK_TYPE.EMV_CALLBACK_TYPE_LCD, and lcd_displayMode = EMV_LCD_DISPLAY_MODE_MENU, EMV_LCD_DISPLAY_MODE_PROMPT, or EMV_LCD_DISPLAY_MODE_LANGUAGE_SELECT

Parameters

<i>type</i>	If Cancel key pressed during menu selection, then value is EMV_LCD_DISPLAY_MODE_CANCEL. Otherwise, value can be EMV_LCD_DISPLAY_MODE_MENU, EMV_LCD_DISPLAY_MODE_PROMPT, or EMV_LCD_DISPLAY_MODE_LANGUAGE_SELECT
<i>selection</i>	If type = EMV_LCD_DISPLAY_MODE_MENU or EMV_LCD_DISPLAY_MODE_LANGUAGE_SELECT, provide the selection ID line number. Otherwise, if type = EMV_LCD_DISPLAY_MODE_PROMPT supply either 0x43 ('C') for Cancel, or 0x45 ('E') for Enter/accept
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString

21.1.2.56 **RETURN_CODE** IDTechSDK.IDT_Augusta.emv_callbackResponseMSR (*byte[] MSR*, *string ident* = " ")

Callback Response MSR Entry

Provides MSR information to kernel after a callback was received with DeviceState.EMVCallback, and callbackType = EMV_CALLBACK_MSR

Parameters

<i>MSR</i>	Swiped track data
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString

21.1.2.57 **RETURN_CODE** IDTechSDK.IDT_Augusta.emv_callbackResponsePIN (*EMV_PIN_MODE type*, byte[] *KSN*, byte[] *PIN*, string *ident* = " ")

Callback Response PIN Entry

Provides (or cancels) PIN entry information to kernel after a callback was received with DeviceState.EMVCallback, and callbackType = EMV_CALLBACK_TYPE_PINPAD

Parameters

<i>type</i>	If Cancel key pressed during PIN entry, then value is EMV_PIN_MODE_CANCEL. Otherwise, value can be EMV_PIN_MODE_ONLINE_DUKPT, EMV_PIN_MODE_ONLINE_MKSK, or EMV_PIN_MODE_OFFLINE
<i>KSN</i>	If enciphered PIN, this is either the PIN DUKPT Key (EMV_PIN_MODE_ONLINE_DUKPT) or PIN Session Key (EMV_PIN_MODE_ONLINE_MKSK), or PIN Pairing DUKPT key (EMV_PIN_MODE_OFFLINE)
<i>PIN</i>	If enciphered PIN, this is encrypted PIN block. If device does not implement pairing function, this is plaintext PIN
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString

21.1.2.58 **RETURN_CODE** IDTechSDK.IDT_Augusta.emv_cancelTransaction (string *ident* = " ")

Cancel EMV Transaction

Cancels the currently executing EMV transaction.

Parameters

<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.
--------------	--

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString

21.1.2.59 **RETURN_CODE** IDTechSDK.IDT_Augusta.emv_completeTransaction (bool *commError*, byte[] *authCode*, byte[] *iad*, byte[] *tlvScripts*, byte[] *tlv*, string *ident* = " ")

Complete EMV Transaction Request

Completes the EMV transaction for an ICC card when online authorization request is received from emv_← authenticateTransaction

The tags will be returned in the callback routine.

Parameters

<i>commError</i>	Communication error with host. Set to TRUE if host was unreachable, or FALSE if host response received. If Communication error, authCode, iad, tlvScripts can be null.
<i>authCode</i>	Authorization code from host. Two bytes. Example 0x3030. (Tag value 8A). Required
<i>iad</i>	Issuer Authentication Data, if any. Example 0x11223344556677883030 (tag value 91).
<i>tlvScripts</i>	71/72 scripts, if any
<i>tlv</i>	Additional TVL data to return with transaction results (if any)

NOTE: sending tag DFEF51 with a value of 01 (DFEF510101) as and Additional TLV data will Suppress the results. Used for QuickChip implementation.

Parameters

<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.
--------------	--

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString

NOTE: There are three possible outcomes for Authorization Code: Approval, Refer To Bank, Decline. The kernel maps these three outcomes to valid authorization response codes using tag DFEE1B through 8 bytes: {2 bytes Approval Code}{2 bytes Referral Code}{2 bytes Decline Code}{2 bytes RFU} If your gateway uses "00" for Approval, "01" for Referral, and "05" for Decline, then DFEE1B 08 3030 3031 3035 0000 If you use an authorization code value that is not defined in DFEE1B, the kernel will use the DECLINE value of DFEE1B by default.

21.1.2.60 RETURN_CODE IDTechSDK.IDT_Augusta.emv_getEMVConfigurationCheckValue (ref string *response*, string *ident* = " ")

Get EMV Kernel configuration check value info

Parameters

<i>response</i>	Response returned of Kernel configuration check value info
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString

21.1.2.61 RETURN_CODE IDTechSDK.IDT_Augusta.emv_getEMVKernelCheckValue (ref string *response*, string *ident* = " ")

Get EMV Kernel check value info

Parameters

<i>response</i>	Response returned of Kernel check value info
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString

21.1.2.62 RETURN_CODE IDTechSDK.IDT_Augusta.emv_getEMVKernelVersion (ref string *response*, string *ident* = " ")

Polls device for EMV Kernel Version

Parameters

<i>response</i>	Response returned of Kernel Version
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString

21.1.2.63 RETURN_CODE IDTechSDK.IDT_Augusta.emv_removeAllApplicationData (string *ident* = " ")

Remove All Application Data

Removes all the Application Data

Parameters

<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.
--------------	--

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString

21.1.2.64 RETURN_CODE IDTechSDK.IDT_Augusta.emv_removeAllCAPK (string *ident* = " ")

Remove All Certificate Authority Public Key

Removes all the CAPK

Parameters

<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.
--------------	--

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString

21.1.2.65 RETURN_CODE IDTechSDK.IDT_Augusta.emv_removeAllCRL (string *ident* = " ")

Remove All Certificate Revocation List Entries

Removes all CRLEntry entries

Parameters

<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.
--------------	--

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString

21.1.2.66 RETURN_CODE IDTechSDK.IDT_Augusta.emv_removeApplicationData (byte[] *AID*, string *ident* = " ")

Remove Application Data by AID

Removes the Application Data as specified by the AID name passed as a parameter

Parameters

<i>AID</i>	Name of ApplicationID Must be between 5 and 16 bytes
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString

21.1.2.67 RETURN_CODE IDTechSDK.IDT_Augusta.emv_removeCAPK (byte[] *capk*, string *ident* = " ")

Remove Certificate Authority Public Key

Removes the CAPK as specified by the RID/Index

Parameters

<i>capk</i>	6 byte CAPK = 5 bytes RID + 1 byte INDEX
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString

21.1.2.68 RETURN_CODE IDTechSDK.IDT_Augusta.emv_removeCRL (byte[] *crlList*, string *ident* = " ")

Remove Certificate Revocation List Entries

Removes CRL entries as specified by the RID and Index and serial number passed as 9 bytes

Parameters

<i>crlList</i>	containing the list of CRL to remove: [CRL1][CRL2]...[CRLn] where each [CRL] is 9 bytes: [5 bytes RID][1 byte CAPK Index][3 bytes serial number]
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString

21.1.2.69 RETURN_CODE IDTechSDK.IDT_Augusta.emv_removeTerminalData (string *ident* = " ")

Remove Terminal Data

Removes the Terminal Data

Parameters

<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.
--------------	--

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString

21.1.2.70 RETURN_CODE IDTechSDK.IDT_Augusta.emv_retrieveAIDList (ref byte *response*[[[]], string *ident* = " ")

Retrieve AID list

Returns all the AID names installed on the terminal.

Parameters

<i>response</i>	array of AID name byte arrays
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString

21.1.2.71 RETURN_CODE IDTechSDK.IDT_Augusta.emv_retrieveApplicationData (byte[] *AID*, ref byte[] *tlv*, string *ident* = " ")

Retrieve Application Data by AID

Retrieves the Application Data as specified by the AID name passed as a parameter.

Parameters

<i>AID</i>	Name of ApplicationID. Must be between 5 and 16 bytes
<i>tlv</i>	The TLV elements of the requested AID

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString

21.1.2.72 RETURN_CODE IDTechSDK.IDT_Augusta.emv_retrieveCAPK (byte[] *capk*, ref byte[] *key*, string *ident* = " ")

Retrieve Certificate Authority Public Key

Retrieves the CAPK as specified by the RID/Index passed as a parameter.

Parameters

<i>capk</i>	6 bytes CAPK = 5 bytes RID + 1 byte Index
-------------	---

Parameters

<i>key</i>	<p>Response returned as a CAKey format: [5 bytes RID][1 byte Index][1 byte Hash Algorithm][1 byte Encryption Algorithm][20 bytes HashValue][4 bytes Public Key Exponent][2 bytes Modulus Length][Variable bytes Modulus] Where:</p> <ul style="list-style-type: none"> • Hash Algorithm: The only algorithm supported is SHA-1. The value is set to 0x01 • Encryption Algorithm: The encryption algorithm in which this key is used. Currently support only one type: RSA. The value is set to 0x01. • HashValue: Which is calculated using SHA-1 over the following fields: RID & Index & Modulus & Exponent • Public Key Exponent: Actually, the real length of the exponent is either one byte or 3 bytes. It can have two values: 3 (Format is 0x00 00 00 03), or 65537 (Format is 0x00 01 00 01) • Modulus Length: LenL LenH Indicated the length of the next field. • Modulus: This is the modulus field of the public key. Its length is specified in the field above.
------------	---

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString

21.1.2.73 RETURN_CODE IDTechSDK.IDT_Augusta.emv_retrieveCAPKList (ref byte[] keys, string ident = " ")

Retrieve the Certificate Authority Public Key list

Returns all the CAPK RID and Index installed on the terminal.

Parameters

<i>keys</i>	[key1][key2]...[keyn], each key 6 bytes where key = 5 bytes RID + 1 byte index
-------------	--

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString

21.1.2.74 RETURN_CODE IDTechSDK.IDT_Augusta.emv_retrieveCRLList (ref byte[] list, string ident = " ")

Retrieve the Certificate Revocation List

Returns the CRL entries on the terminal.

Parameters

<i>list</i>	[CRL1][CRL2]...[CRLn], each CRL 9 bytes where CRL = 5 bytes RID + 1 byte index + 3 bytes serial number
-------------	--

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString

21.1.2.75 **RETURN_CODE** IDTechSDK.IDT_Augusta.emv_retrieveTerminalData (ref byte[] *tlv*, string *ident* = " ")

Retrieve Terminal Data

Retrieves the Terminal Data.

Parameters

<i>tlv</i>	Response returned as a TLV
------------	----------------------------

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString

21.1.2.76 **RETURN_CODE** IDTechSDK.IDT_Augusta.emv_retrieveTransactionResult (byte[] *tags*, ref IDTTTransactionData *tlv*, string *ident* = " ")

Retrieve Transaction Results

Retrieves specified EMV tags from the currently executing transaction.

Parameters

<i>tags</i>	Tags to be retrieved. Example 0x9F028A will retrieve tags 9F02 and 8A
<i>tlv</i>	All requested tags returned as unencrypted, encrypted and masked TLV data in IDTTTransactionData object
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString

21.1.2.77 **RETURN_CODE** IDTechSDK.IDT_Augusta.emv_setApplicationData (byte[] *name*, byte[] *tlv*, string *ident* = " ")

Set Application Data by AID

Sets the Application Data as specified by the application name and TLV data

Parameters

<i>name</i>	Application name, 10-32 ASCII hex characters representing 5-16 bytes Example "a0000000031010"
<i>tlv</i>	Application data in TLV format

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString

21.1.2.78 **RETURN_CODE** IDTechSDK.IDT_Augusta.emv_setCAPK (byte[] *key*, string *ident* = " ")

Set Certificate Authority Public Key

Sets the CAPK as specified by the CAKey structure

Parameters

<i>key</i>	<p>CAKey format: [5 bytes RID][1 byte Index][1 byte Hash Algorithm][1 byte Encryption Algorithm][20 bytes HashValue][4 bytes Public Key Exponent][2 bytes Modulus Length][Variable bytes Modulus] Where:</p> <ul style="list-style-type: none"> • Hash Algorithm: The only algorithm supported is SHA-1. The value is set to 0x01 • Encryption Algorithm: The encryption algorithm in which this key is used. Currently support only one type: RSA. The value is set to 0x01. • HashValue: Which is calculated using SHA-1 over the following fields: RID & Index & Modulus & Exponent • Public Key Exponent: Actually, the real length of the exponent is either one byte or 3 bytes. It can have two values: 3 (Format is 0x00 00 00 03), or 65537 (Format is 0x00 01 00 01) • Modulus Length: LenL LenH Indicated the length of the next field. • Modulus: This is the modulus field of the public key. Its length is specified in the field above.
------------	--

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString

21.1.2.79 RETURN_CODE IDTechSDK.IDT_Augusta.emv_setCRL (byte[] *list*, string *ident* = " ")

Set Certificate Revocation List

Sets the CRL

Parameters

<i>list</i>	CRL Entries containing the RID, Index, and serial numbers to set [CRL1][CRL2]...[CRLn] where each [CRL] is 9 bytes: [5 bytes RID][1 byte CAPK Index][3 bytes serial number]
-------------	---

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString

21.1.2.80 RETURN_CODE IDTechSDK.IDT_Augusta.emv_setTerminalData (byte[] *tlv*, string *ident* = " ")

Set Terminal Data

Sets the Terminal Data

Parameters

<i>tlv</i>	TerminalData TLV Data
<i>ident</i>	Optional identifier

Return values

<i>RETURN_CODE</i>	Return codes listed as typedef enum in IDTCommon:RETURN_CODE. Values can be parsed with IDT_BTPay::device_getResponseCodeString:()
--------------------	--

21.1.2.81 RETURN_CODE IDTechSDK.IDT_Augusta.emv_setTerminalMajorConfiguration (int *configuration*, string *ident* = " ")

Set Terminal Major Configuration

Sets the Terminal Data Major Configuration setting

Parameters

<i>configuration</i>	The configuration to set (1-5)
----------------------	--------------------------------

Return values

<i>RETURN_CODE</i>	Return codes listed as typedef enum in IDTCommon:RETURN_CODE. Values can be parsed with IDT_BTPay::device_getResponseCodeString:()
--------------------	--

21.1.2.82 RETURN_CODE IDTechSDK.IDT_Augusta.emv_startTransaction (double *amount*, double *amtOther*, int *exponent*, int *type*, int *timeout*, byte[] *tags*, bool *forceOnline*, string *ident* = " ")

Start EMV Transaction Request

Authorizes the EMV transaction for an ICC card

The tags will be returned in the callback routine.

Parameters

<i>amount</i>	Transaction amount value (tag value 9F02)
<i>amtOther</i>	Other amount value, if any (tag value 9F03)
<i>exponent</i>	Number of characters after decimile point
<i>type</i>	Transaction type (tag value 9C).
<i>timeout</i>	Timeout value in seconds.
<i>tags</i>	Any other tags to be included in the request. Passed as a TLV data. Example, tag 9F02 with amount 0x000000000100 would be 0x9F0206000000000100 If tags 9F02 (amount), 9F03 (other amount), or 9C (transaction type) are included, they will take priority over these values supplied as individual parameters to this method.
<i>forceOnline</i>	TRUE = do not allow offline approval, FALSE = allow ICC to approve offline if terminal capable Note: To request tags to be included in default response, use tag DFEE1A, and specify tag list. Example four tags 9F02, 9F36, 95, 9F37 to be included in response = DFEE1A079F029F369F9F37
<i>isFastEMV</i>	If TRUE, it will populate the IDTTransactionData.fastEMV with ASCII data similar to IDTech FastEMV KB output, after performing an auto-authenticate and auto-complete with ResultCode = Could Not Contact Host
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString

21.1.2.83 RETURN_CODE IDTechSDK.IDT_Augusta.emv_trySetTerminalData (byte[] *tlv*, ref byte[] *rejectedTLV*, ref byte[] *convertedTLV*, bool *overwrite* = false, string *ident* = " ")

Try to Set Terminal Data

Attempts to set the Terminal Data. Use this functions to attempt to set the terminal data to the device. This function will allow the cross-device tag compatible writing of IDTech Legacy/New tag definitions

Parameters

<i>tlv</i>	TerminalData TLV Data
<i>rejectedTLV</i>	Contains the tags (if any) that were rejected by the firmware
<i>convertedTLV</i>	Contains the tags (if any) that were converted and accepted by the firmware
<i>overwrite</i>	TRUE = add TLV to existing tags, FALSE = replace existing tags with TLV
<i>ident</i>	Optional identifier

Return values

<i>RETURN_CODE</i>	Return codes listed as typedef enum in IDTCommon:RETURN_CODE. Values can be parsed with IDT_BTPay::device_getResponseCodeString:()
--------------------	--

21.1.2.84 `static int IDTechSDK.IDT_Augusta.getCommandTimeout (string ident = " ") [static]`

Get Command Timeout

Gets the default timeout (in milliseconds) waiting for a blocking command response

Parameters

<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.
--------------	--

Return values

<i>time</i>	Time
-------------	------

21.1.2.85 `RETURN_CODE IDTechSDK.IDT_Augusta.icc_disable (string ident = " ")`

ICC Function enable/disable Disable ICC function

Parameters

<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.
--------------	--

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString

21.1.2.86 `RETURN_CODE IDTechSDK.IDT_Augusta.icc_enable (bool withNotification, string ident = " ")`

ICC Function enable/disable Enable ICC function with or without seated notification

Parameters

<i>withNotification</i>	<ul style="list-style-type: none"> • true: with notification when ICC seated status changed, • false: without notification.
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString

21.1.2.87 RETURN_CODE IDTechSDK.IDT_Augusta.icc_exchangeAPDU (string *c_APDU*, ref byte[] *response*, string *ident* = " ")

Exchange APDU with plain text For Non-SRED Augusta Only

Sends an APDU packet to the ICC. If successful, response is the APDU data in response parameter.

Parameters

<i>c_APDU</i>	APDU data packet
<i>response</i>	Unencrypted APDU response
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString

21.1.2.88 RETURN_CODE IDTechSDK.IDT_Augusta.icc_exchangeEncryptedAPDU (string *c_APDU*, ref byte[] *response*, string *ident* = " ")

Exchange APDU with encrypted data For SRED Augusta Only

Sends an APDU packet to the ICC. If successful, response is the APDU data in response parameter.

Parameters

<i>c_APDU</i>	KSN + encrypted APDU data packet, or no KSN (use last known KSN) + encrypted APDU data packet With KSN: [0A][KSN][Encrypted C-APDU] Without KSN: [00][Encrypted C-APDU]
---------------	---

The format of Raw C-APDU Data Structure of [m-bytes Encrypted C-APDU] is below:

- m = 2 bytes Valid C-APDU Length + x bytes Valid C-APDU + y bytes Padding (0x00) Note: For TDES mode: 2+x should be multiple of 8. If it was not multiple of 8, unit should padded y bytes 0x00 automatically (2+x+y should be multiple of 8). For AES mode: 2+x should be multiple of 16. If it was not multiple of 16, unit should padded y bytes 0x00 automatically (2+x+y should be multiple of 16).

Parameters

<i>response</i>	<p>encrypted APDU response. Can be three options:</p> <ul style="list-style-type: none"> • [00] + [Plaintext R-APDU] • [01] + [0A] + [KSN] + [n bytes Encrypted R-APDU without Status Bytes] + [2 bytes Status Bytes] • [01] + [00] + [n bytes Encrypted R-APDU without Status Bytes] + [2 bytes Status Bytes] <p>The KSN, when provided, will be 10 bytes. The KSN will only be provided when it has changed since the last provided KSN. Each card Power-On generates a new KSN. During a sequence of commands where the KSN is identical, the first response will have a KSN length set to [0x0A] followed by the KSN, while subsequent commands with the same KSN value will have a KSN length of [0x00] followed by the Encrypted R-APDU without Status Bytes.</p>
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with `device_getResponseCodeString`

21.1.2.89 RETURN_CODE IDTechSDK.IDT_Augusta.icc_getAPDU_KSN (ref byte[] *ksn*, string *ident* = " ")

Get APDU KSN

Retrieves the KSN used in ICC Encrypted APDU usage

Parameters

<i>ksn</i>	Returns the encrypted APDU packet KSN
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with `device_getResponseCodeString`

21.1.2.90 RETURN_CODE IDTechSDK.IDT_Augusta.icc_getFunctionStatus (ref bool *enabled*, ref bool *withNotification*, string *ident* = " ")

Get ICC Function status Get ICC Function status about enable/disable and with or without seated notification

Parameters

<i>enabled</i>	<ul style="list-style-type: none"> • true: ICC Function enabled, • false: means disabled.
<i>withNotification</i>	true means with notification when ICC seated status changed. false means without notification.
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString

21.1.2.91 RETURN_CODE IDTechSDK.IDT_Augusta.icc_getICCReaderStatus (ref byte *status*, string *ident* = " ")

Get Reader Status

Returns the reader status

Parameters

<i>status</i>	Pointer that will return with the ICCReaderStatus results. bit 0: 0 = ICC Power Not Ready, 1 = ICC Powered bit 1: 0 = Card not seated, 1 = card seated
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString

21.1.2.92 RETURN_CODE IDTechSDK.IDT_Augusta.icc_getKeyFormatForICCDUKPT (ref byte *format*, string *ident* = " ")

Get Key Format For ICC DUKPT

Specifies how data will be encrypted with Data Key or PIN key (if DUKPT key loaded)

Parameters

<i>format</i>	Response returned from method: <ul style="list-style-type: none"> • 'TDES': Encrypted card data with TDES if DUKPT Key had been loaded.(default) • 'AES': Encrypted card data with AES if DUKPT Key had been loaded. • 'NONE': No Encryption.
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString

21.1.2.93 RETURN_CODE IDTechSDK.IDT_Augusta.icc_getKeyTypeForICCDUKPT (ref byte *type*, string *ident* = " ")

Get Key Type for ICC DUKPT

Specifies the key type used for ICC DUKPT encryption

Parameters

<i>type</i>	Response returned from method: <ul style="list-style-type: none"> • 'DATA': Encrypted card data with Data Key DUKPT Key had been loaded.(default) • 'PIN': Encrypted card data with PIN Key if DUKPT Key had been loaded.
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString

21.1.2.94 RETURN_CODE IDTechSDK.IDT_Augusta.icc_getSetting (byte *setting*, ref byte *value*, string *ident* = " ")

Get Single ICC Setting value

Returns the encryption used for sweep data

Parameters

<i>setting</i>	ICC Setting to retrieve
<i>value</i>	ICC Setting value
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString

21.1.2.95 RETURN_CODE IDTechSDK.IDT_Augusta.icc_getSettings (byte *setting*, ref byte[] *value*, string *ident* = " ")

Get Multi ICC Setting value

Returns the encryption used for sweep data

Parameters

<i>setting</i>	ICC Setting to retrieve
<i>value</i>	ICC Setting value
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString

21.1.2.96 RETURN_CODE IDTechSDK.IDT_Augusta.icc_powerOffICC (string *ident* = " ")

Power Off ICC

Powers down the ICC

Parameters

<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.
--------------	--

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString

If Success, empty If Failure, ASCII encoded data of error string

21.1.2.97 **RETURN_CODE** IDTechSDK.IDT_Augusta.icc_powerOnICC (ref byte[] *ATR*, string *ident* = " ")

Power On ICC

Power up the currently selected microprocessor card in the ICC reader

Parameters

<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.
--------------	--

Returns

RETURN_CODE: Values can be parsed with `device_getResponseCodeString`

21.1.2.98 **RETURN_CODE** IDTechSDK.IDT_Augusta.icc_setKeyFormatForICCDUKPT (byte *encryption*, string *ident* = " ")

Set Key Format for ICC DUKPT

Sets how data will be encrypted, with either TDES or AES (if DUKPT key loaded)

Parameters

<i>encryption</i>	Encryption Type <ul style="list-style-type: none"> • 00: Encrypt with TDES • 01: Encrypt with AES
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with `device_getResponseCodeString`

21.1.2.99 **RETURN_CODE** IDTechSDK.IDT_Augusta.icc_setKeyTypeForICCDUKPT (byte *encryption*, string *ident* = " ")

Set Key Type for ICC DUKPT Key

Sets which key the data will be encrypted with, with either Data Key or PIN key (if DUKPT key loaded)

Parameters

<i>encryption</i>	Encryption Type <ul style="list-style-type: none"> • 00: Encrypt with Data Key • 01: Encrypt with PIN Key
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with `device_getResponseCodeString`

21.1.2.100 **RETURN_CODE** IDTechSDK.IDT_Augusta.icc_setSetting (byte *setting*, byte *value*, string *ident* = " ")

Set Single ICC Setting value

Parameters

<i>setting</i>	ICC Setting to set
<i>value</i>	ICC Setting value
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString

21.1.2.101 **RETURN_CODE** IDTechSDK.IDT_Augusta.icc_setSetting (byte *setting*, byte[] *value*, string *ident* = " ")

Set Multi ICC Setting value

Parameters

<i>setting</i>	ICC Setting to set
<i>value</i>	ICC Setting value
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString

21.1.2.102 static void IDTechSDK.IDT_Augusta.lcd_retrieveMessage (DisplayMessages.DISPLAY_MESSAGE_LANGUAGE *lang*, DisplayMessages.DISPLAY_MESSAGE_IDENTIFIER *id*, ref string *line1*, ref string *line2*) [static]

Retrieve LCD Message

Returns the string value for a message ID returned for LCD messaging

Parameters

<i>lang</i>	Language.
<i>id</i>	Message ID
<i>line1</i>	Line 1 string value
<i>line2</i>	Line 2 string value
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

21.1.2.103 **RETURN_CODE** IDTechSDK.IDT_Augusta.msr_cancelMSRSwipe (string *ident* = " ")

Disable MSR Swipe Cancels MSR swipe request.

Parameters

<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.
--------------	--

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString

21.1.2.104 RETURN_CODE IDTechSDK.IDT_Augusta.msr_captureMode (bool *isBufferMode*, bool *withNotification* = false, string *ident* = " ")

Set MSR Capture Mode.

For Non-SRED Augusta Only

Switch between Auto mode and Buffer mode. Auto mode only available on KB interface

Parameters

<i>isBufferMode</i>	<ul style="list-style-type: none"> • true: Enter into Buffer mode. • false: Enter into Auto mode. KB mode only. Swipes automatically captured and sent to keyboard buffer
<i>withNotification</i>	<ul style="list-style-type: none"> • true: with notification when swiped MSR data. • false: without notification when swiped MSR data.
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString

21.1.2.105 RETURN_CODE IDTechSDK.IDT_Augusta.msr_disable (string *ident* = " ")

Disable MSR function.

Parameters

<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.
--------------	--

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString

21.1.2.106 RETURN_CODE IDTechSDK.IDT_Augusta.msr_getClearPANID (ref byte *value*, string *ident* = " ")

Get Clear PAN Digits

Returns the number of digits that begin the PAN that will be in the clear

Parameters

<i>value</i>	Number of digits in clear. Values are char '0' - '6':
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString

21.1.2.107 RETURN_CODE IDTechSDK.IDT_Augusta.msr_getExpirationMask (ref byte *value*, string *ident* = " ")

Get Expiration Masking

Get the flag that determines if to mask the expiration date

Parameters

<i>value</i>	'0' = masked, '1' = not-masked
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString

21.1.2.108 RETURN_CODE IDTechSDK.IDT_Augusta.msr_getFunctionStatus (ref bool *enabled*, ref bool *isBufferMode*, ref bool *withNotification*, string *ident* = " ")

Get MSR Function status Get MSR Function status about enable/disable and with or without seated notification

Parameters

<i>enabled</i>	<ul style="list-style-type: none"> • true: MSR Function enabled. • false: MSR Function disabled.
<i>isBufferMode</i>	<ul style="list-style-type: none"> • true: in the buffer mode. • false: in the auto mode.
<i>withNotification</i>	<ul style="list-style-type: none"> • true: with notification when swiped MSR Card. • false: without notification when swiped MSR Card.
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString

21.1.2.109 RETURN_CODE IDTechSDK.IDT_Augusta.msr_getSetting (byte *setting*, ref byte *value*, string *ident* = " ")

Get Single MSR Setting value

Returns the encryption used for sweep data

Parameters

<i>setting</i>	MSR Setting to retrieve
<i>value</i>	MSR Setting value
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString

21.1.2.110 RETURN_CODE IDTechSDK.IDT_Augusta.msr_getSettings (byte *setting*, ref byte[] *value*, string *ident* = " ")

Get Multi MSR Setting value

Returns the encryption used for sweep data

Parameters

<i>setting</i>	MSR Setting to retrieve
<i>value</i>	MSR Setting value
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString

21.1.2.111 RETURN_CODE IDTechSDK.IDT_Augusta.msr_getSwipeEncryption (ref byte *encryption*, string *ident* = " ")

Get Swipe Data Encryption

For Non-SRED Augusta Only

Returns the encryption used for swipe data

Parameters

<i>encryption</i>	1 = TDES, 2 = AES, 0 = NONE
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString

21.1.2.112 RETURN_CODE IDTechSDK.IDT_Augusta.msr_getSwipeForcedEncryptionOption (ref byte *option*, string *ident* = " ")

Get Swipe Data Encryption

Gets the swipe force encryption options

Parameters

<i>option</i>	Byte using lower four bits as flags. 0 = Force Encryption Off, 1 = Force Encryption On bit0 = Track 1 bit1 = Track 2 bit2 = Track 3 bit4 = Track 3 Card Option 0
---------------	--

Example: Response 0x03 = Track1/Track2 Forced Encryption, Track3/Track3-0 no Forced Encryption

Parameters

<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used
--------------	---

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString

21.1.2.113 RETURN_CODE IDTechSDK.IDT_Augusta.msr_getSwipeMaskOption (ref byte *option*, string *ident* = " ")

Get Swipe Mask Option

Gets the swipe mask/clear data sending option

Parameters

<i>option</i>	Byte using lower three bits as flags. 0 = Mask Option Off, 1 = Mask Option On bit0 = Track 1 bit1 = Track 2 bit2 = Track 3
---------------	--

Example: Response 0x03 = Track1/Track2 Masked Option ON, Track3 Masked Option Off

Parameters

<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.
--------------	--

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString

21.1.2.114 RETURN_CODE IDTechSDK.IDT_Augusta.msr_RetrieveWhiteList (ref byte[] *value*, string *ident* = " ")

Retrieve MSR White List

For Non-SRED Augusta Only

Parameters

<i>value</i>	is the white list data which is ASN.1 Block format
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString

21.1.2.115 **RETURN_CODE** IDTechSDK.IDT_Augusta.msr_setClearPANID (*byte val*, *string ident* = " ")

Set Clear PAN Digits

Sets the amount of digits shown in the clear (not masked) at the beginning of the returned PAN value

Parameters

<i>val</i>	Number of digits to show in clear. Range 0-6.
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString

21.1.2.116 **RETURN_CODE** IDTechSDK.IDT_Augusta.msr_setExpirationMask (*bool mask*, *string ident* = " ")

Set Expiration Masking

Sets the flag to mask the expiration date

Parameters

<i>mask</i>	TRUE = mask expiration
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString

21.1.2.117 **RETURN_CODE** IDTechSDK.IDT_Augusta.msr_setSetting (*byte setting*, *byte value*, *string ident* = " ")

Set Single MSR Setting value

Parameters

<i>setting</i>	MSR Setting to set
<i>value</i>	MSR Setting value
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString

21.1.2.118 **RETURN_CODE** IDTechSDK.IDT_Augusta.msr_setSettings (*byte setting*, *byte[] value*, *string ident* = " ")

Set Multi MSR Setting value

Parameters

<i>setting</i>	MSR Setting to set
----------------	--------------------

Parameters

<i>value</i>	MSR Setting value
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString

21.1.2.119 RETURN_CODE IDTechSDK.IDT_Augusta.msr_setSwipeEncryption (byte *encryption*, string *ident* = " ")

Set Swipe Data Encryption

For Non-SRED Augusta Only

Sets the swipe encryption method

Parameters

<i>encryption</i>	1 = TDES, 2 = AES
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString

21.1.2.120 RETURN_CODE IDTechSDK.IDT_Augusta.msr_setSwipeForcedEncryptionOption (bool *track1*, bool *track2*, bool *track3*, bool *track3card0*, string *ident* = " ")

Set Swipe Force Encryption

Sets the swipe force encryption options

Parameters

<i>track1</i>	Force encrypt track 1
<i>track2</i>	Force encrypt track 2
<i>track3</i>	Force encrypt track 3
<i>track3card0</i>	Force encrypt track 3 when card type is 0
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString

21.1.2.121 RETURN_CODE IDTechSDK.IDT_Augusta.msr_setSwipeMaskOption (bool *track1*, bool *track2*, bool *track3*, string *ident* = " ")

Set Swipe Mask Option

Sets the swipe mask/clear data sending option

Parameters

<i>track1</i>	Mask track 1 allowed
<i>track2</i>	Mask track 2 allowed
<i>track3</i>	Mask track 3 allowed
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString

21.1.2.122 RETURN_CODE IDTechSDK.IDT_Augusta.msr_setWhiteList (byte[] *value*, byte[] *MAC*, string *ident* = " ")

Set MSR White List

For Non-SRED Augusta Only

Parameters

<i>value</i>	the white list data which is ASN.1 Block format
<i>MAC</i>	Message Authenticate Code. For non-PCI, use null
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString

21.1.2.123 RETURN_CODE IDTechSDK.IDT_Augusta.msr_setWhiteListFromBDK (byte[] *val*, byte[] *BDK*, string *ident* = " ")

Set MSR White List from BDK

For SRED Augusta Only

Parameters

<i>value</i>	the white list data which is ASN.1 Block format
<i>BDK</i>	The Device BDK
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString

21.1.2.124 RETURN_CODE IDTechSDK.IDT_Augusta.msr_startMSRSwipe (int *timeout*, string *ident* = " ")

Enable MSR Swipe

Enables MSR, waiting for swipe to occur. Allows track selection. Returns IDTMSRData instance to deviceDelegate->::swipeMSRData:()

Parameters

<i>timeout</i>	Swipe Timeout Value
----------------	---------------------

Parameters

<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.
--------------	--

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString

21.1.2.125 RETURN_CODE IDTechSDK.IDT_Augusta.msr_switchUSBInterfaceMode (bool *blsUSBKeyboardMode*)

Switch the USB interface mode between USB HID and USB KB mode.

For Non-SRED Augusta Only

Parameters

<i>blsUSBKeyboardMode</i>	USB interface mode <ul style="list-style-type: none"> • true: Enter into USB Keyboard mode • false: Enter into USB HID mode
---------------------------	---

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString

21.1.2.126 RETURN_CODE IDTechSDK.IDT_Augusta.msr_switchUSBInterfaceMode (bool *blsUSBKeyboardMode*, ref string *ident*)

Switch the USB interface mode between USB HID and USB KB mode.

For Non-SRED Augusta Only

Parameters

<i>blsUSBKeyboardMode</i>	USB interface mode <ul style="list-style-type: none"> • true: Enter into USB Keyboard mode • false: Enter into USB HID mode
<i>ident</i>	Device ID to send command to.

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString

21.1.2.127 static String IDTechSDK.IDT_Augusta.SDK_Version () [static]

SDK Version

- All Devices

Returns the current version of SDK

Returns

Framework version

21.1.2.128 `static void IDTechSDK.IDT_Augusta.setCallback (Callback my_Callback)` `[static]`

Set Callback

Sets the class callback

Parameters

<i>my_Callback</i>	The callback function to receive the response message from device. defined as follows. delegate void CallBack(IDT_DEVICE_Types sender, DeviceState state, byte[] data, IDTTransactionData card, EMV_Callback emvCallback, RETURN_CODE transactionResultCode);
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

21.1.2.129 `static void IDTechSDK.IDT_Augusta.setCallback (IntPtr my_Callback, SynchronizationContext context)`
`[static]`

Set Callback

Sets the class callback

Parameters

<i>my_Callback</i>	The callback function to receive the response message from device. defined as follows. public unsafe delegate void MFCCallBack(Parameters parameters);
<i>context</i>	The context of the UI thread

21.1.2.130 `static void IDTechSDK.IDT_Augusta.setCommandTimeout (int milliseconds, string ident = " ")` `[static]`

Set Command Timeout**Parameters**

<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.
--------------	--

Sets the default timeout (in milliseconds) waiting for a blocking command response

Parameters

<i>milliseconds</i>	Time
---------------------	------

21.1.3 Property Documentation

21.1.3.1 `IDT_Augusta IDTechSDK.IDT_Augusta.SharedController` `[static], [get]`

Singleton Instance

Establishes an singleton instance of [IDT_Augusta](#) class.

Returns

Instance of [IDT_Augusta](#)

The documentation for this class was generated from the following file:

- /Users/randy/Repo/universal-sdk/IDTechSDK_STD/DeviceHeaders/IDT_Augusta.cs

21.2 IDTechSDK.IDT_BTMag Class Reference

Public Member Functions

- RETURN_CODE [config_getSerialNumber](#) (ref string response, string ident="")
- RETURN_CODE [device_sendPAE](#) (string command, ref string response, int timeout, string ident="")
- bool [config_setCmdTimeOutDuration](#) (int newTimeOut, string ident="")
- RETURN_CODE [config_setLEDController](#) (bool firmwareControlMSRLED, bool firmwareControlICCL↵ED=false, string ident="")
- RETURN_CODE [config_getLEDController](#) (ref bool firmwareControlMSRLED, ref bool firmwareControlICC↵LED, string ident="")
- RETURN_CODE [device_controlLED_ICC](#) (int controlMode, int interval, string ident="")
- RETURN_CODE [device_controlBeep](#) (int index, int frequency, int duration, string ident="")
- RETURN_CODE [device_controlLED](#) (byte indexLED, byte control, int intervalOn=500, int intervalOff=500, string ident="")
- RETURN_CODE [config_setEncryptionControl](#) (bool msr, bool icc, string ident="")
- RETURN_CODE [config_getEncryptionControl](#) (ref bool msr, ref bool icc, string ident="")
- RETURN_CODE [device_getKeyStatus](#) (ref byte[] status, string ident="")
- RETURN_CODE [device_setDateTime](#) (string ident="")
- RETURN_CODE [device_getFirmwareVersion](#) (ref string response, string ident="")
- string [device_getResponseCodeString](#) (RETURN_CODE eCode)
- RETURN_CODE [device_rebootDevice](#) (string ident="")
- RETURN_CODE [device_sendDataCommand](#) (string cmd, bool calcLRC, ref byte[] response, string ident="")
- RETURN_CODE [device_controlLED_MSR](#) (byte control, int intervalOn=500, int intervalOff=500, string ident="")
- RETURN_CODE [msr_captureMode](#) (bool isBufferMode, bool withNotification=false, string ident="")
- RETURN_CODE [msr_disable](#) (string ident="")
- RETURN_CODE [msr_getFunctionStatus](#) (ref bool enabled, ref bool isBufferMode, ref bool withNotification, string ident="")
- RETURN_CODE [msr_switchUSBInterfaceMode](#) (bool blsUSBKeyboardMode)
- RETURN_CODE [msr_switchUSBInterfaceMode](#) (bool blsUSBKeyboardMode, ref string ident)
- RETURN_CODE [msr_getClearPANID](#) (ref byte value, string ident="")
- RETURN_CODE [msr_getExpirationMask](#) (ref byte value, string ident="")
- RETURN_CODE [msr_getSwipeEncryption](#) (ref byte encryption, string ident="")
- RETURN_CODE [msr_getSwipeForcedEncryptionOption](#) (ref byte option, string ident="")
- RETURN_CODE [msr_getSwipeMaskOption](#) (ref byte option, string ident="")
- RETURN_CODE [msr_RetrieveWhiteList](#) (ref byte[] value, string ident="")
- RETURN_CODE [msr_getSetting](#) (byte setting, ref byte value, string ident="")
- RETURN_CODE [msr_getSettings](#) (byte setting, ref byte[] value, string ident="")
- RETURN_CODE [msr_setSetting](#) (byte setting, byte value, string ident="")
- RETURN_CODE [msr_setSettings](#) (byte setting, byte[] value, string ident="")
- RETURN_CODE [msr_setClearPANID](#) (byte val, string ident="")
- RETURN_CODE [msr_setExpirationMask](#) (bool mask, string ident="")
- RETURN_CODE [msr_setSwipeEncryption](#) (byte encryption, string ident="")
- RETURN_CODE [msr_setSwipeForcedEncryptionOption](#) (bool track1, bool track2, bool track3, bool track3card0, string ident="")

- RETURN_CODE [msr_setSwipeMaskOption](#) (bool track1, bool track2, bool track3, string ident="")
- RETURN_CODE [msr_setWhiteList](#) (byte[] value, byte[] MAC, string ident="")
- RETURN_CODE [msr_startMSRSwipe](#) (int timeout, string ident="")
- RETURN_CODE [msr_cancelMSRSwipe](#) (string ident="")
- RETURN_CODE [device_getRKIStatus](#) (bool isProd, bool isMultiKey, ref string status, ref Dictionary< string, RKI_KEY_TYPE > keys, string ident="")
- RETURN_CODE [device_getAnyRKIStatus](#) (bool isMultiKey, ref string status, ref Dictionary< string, RKI_KEY_TYPE > keys, string ident="")
- RETURN_CODE [device_updateFirmwareFromZip](#) (byte[] zipfile, string ident="", bool isForeground=false)
- RETURN_CODE [device_sendConfigurationFromZip](#) (byte[] zip, string filename, VC_OPERATION_TYPE type, bool matchFW, string ident="", bool isForeground=false)
- RETURN_CODE [device_sendConfiguration](#) (string filename, VC_OPERATION_TYPE type, bool matchFW, string ident="", bool isForeground=false)
- RETURN_CODE [device_readConfigurationToMemory](#) (string memo, Configs.InstallRules rules, List< Configs.DeviceCommand > cmds, string ident="", bool isForeground=false)
- RETURN_CODE [device_getConfigurationFromMemory](#) (ref string json, string ident="")
- RETURN_CODE [device_RemoteKeyInjection](#) (RKI_KEY_TYPE type, string keyName, string ident="", bool performOnForeground=false)

Static Public Member Functions

- static bool [useSerialPort](#) (int port)
- static bool [useSerialPort](#) (int port, int baud)
- static bool [useSerialPortLinux](#) (string path)
- static bool [useSerialPortLinux](#) (string path, int baud)
- static void [initSC](#) ()
- static int [getCommandTimeout](#) (string ident="")
- static void [setCommandTimeout](#) (int milliseconds, string ident="")
- static void [setCallback](#) (Callback my_Callback)
- static void [setCallback](#) (IntPtr my_Callback, SynchronizationContext context)
- static String [SDK_Version](#) ()

Properties

- static [IDT_BTMag SharedController](#) [get]

21.2.1 Member Function Documentation

21.2.1.1 RETURN_CODE IDTechSDK.IDT_BTMag.config_getEncryptionControl (ref bool msr, ref bool icc, string ident = " ")

Get Encryption Control

Get Encryption Control to switch status between MSR and ICC/EMV function. Following Encryption status supported:

- MSR ON, ICC/EMV ON,
- MSR ON, ICC/EMV OFF,
- MSR OFF, ICC/EMV OFF,

Parameters

<i>msr</i>	<ul style="list-style-type: none"> • true: enabled MSR with Encryption, • false: disabled MSR with Encryption,
<i>icc</i>	<ul style="list-style-type: none"> • true: enabled ICC with Encryption, • false: disabled ICC with Encryption,

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString

21.2.1.2 RETURN_CODE IDTechSDK.IDT_BTMag.config_getLEDController (ref bool *firmwareControlMSRLED*, ref bool *firmwareControlICCLED*, string *ident* = " ")

Get the LED Controller Status Get the MSR / ICC LED controlled status by software or firmware

Parameters

<i>firmwareControlMSRLED</i>	<ul style="list-style-type: none"> • true: firmware control the MSR LED • false: software control the MSR LED
<i>firmwareControlICCLED</i>	<ul style="list-style-type: none"> • true: firmware control the ICC LED • false: software control the ICC LED

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString

21.2.1.3 RETURN_CODE IDTechSDK.IDT_BTMag.config_getSerialNumber (ref string *response*, string *ident* = " ")

Polls device for Serial Number

Parameters

<i>response</i>	Returns Serial Number
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString

21.2.1.4 `bool IDTechSDK.IDT_BTMag.config_setCmdTimeOutDuration (int newTimeOut, string ident = " ")`

Command Acknowledgement Timeout

Sets the amount of seconds to wait for an {ACK} to a command before a timeout. Responses should normally be received under one second. Default is 3 seconds

Parameters

<i>newTimeOut</i>	Timeout value. Valid range 1 - 60 seconds
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

Success flag. Determines if value was set and in range.

21.2.1.5 `RETURN_CODE IDTechSDK.IDT_BTMag.config_setEncryptionControl (bool msr, bool icc, string ident = " ")`

Set Encryption Control

Set Encryption Control to switch status between MSR and ICC/EMV function. Following Encryption status supported:

- MSR ON, ICC/EMV ON,
- MSR ON, ICC/EMV OFF,
- MSR OFF, ICC/EMV OFF,

Parameters

<i>msr</i>	<ul style="list-style-type: none"> • true: enable MSR with Encryption, • false: disable MSR with Encryption,
<i>icc</i>	<ul style="list-style-type: none"> • true: enable ICC with Encryption, • false: disable ICC with Encryption,

Returns

RETURN_CODE: Values can be parsed with `device_getResponseCodeString`

21.2.1.6 `RETURN_CODE IDTechSDK.IDT_BTMag.config_setLEDController (bool firmwareControlMSRLED, bool firmwareControlICCLEd = false, string ident = " ")`

Set the LED Controller Set the MSR / ICC LED controlled by software or firmware NOTE: The ICC LED always controlled by software.

Parameters

<i>firmwareControlMSRLED</i>	<ul style="list-style-type: none"> • true: firmware control the MSR LED • false: software control the MSR LED
<i>firmwareControlICCLED</i>	<ul style="list-style-type: none"> • true: firmware control the ICC LED • false: software control the ICC LED

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString

21.2.1.7 RETURN_CODE IDTechSDK.IDT_BTMag.device_controlBeep (int *index*, int *frequency*, int *duration*, string *ident* = " ")

Control Beep

Controls the Beeper

Parameters

<i>index</i>	For Augusta, must be set to 1 (only one beeper)
<i>frequency</i>	Frequency, range 1000-20000 (suggest minimum 3000)
<i>duration</i>	Duration, in milliseconds (range 1 - 65525)

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString

21.2.1.8 RETURN_CODE IDTechSDK.IDT_BTMag.device_controlLED (byte *indexLED*, byte *control*, int *intervalOn* = 500, int *intervalOff* = 500, string *ident* = " ")

Control MSR LED

Controls the LED for the MSR

Parameters

<i>indexLED</i>	For Augusta, must be set to 1 (MSR LED)
-----------------	---

Parameters

<i>control</i>	LED Status: <ul style="list-style-type: none"> • 00: OFF • 01: RED Solid • 02: RED Blink • 11: GREEN Solid • 12: GREEN Blink • 21: BLUE Solid • 22: BLUE Blink
<i>intervalOn</i>	Blink interval ON, in ms (Range 200 - 2000)
<i>intervalOff</i>	Blink interval OFF, in ms (Range 200 - 2000)

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString

21.2.1.9 RETURN_CODE IDTechSDK.IDT_BTMag.device_controlLED_ICC (int *controlMode*, int *interval*, string *ident* = " ")

Control ICC LED

Controls the LED for the ICC card slot

Parameters

<i>controlMode</i>	0 = off, 1 = solid, 2 = blink
<i>interval</i>	Blink interval, in ms (500 = 500 ms)

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString

21.2.1.10 RETURN_CODE IDTechSDK.IDT_BTMag.device_controlLED_MSR (byte *control*, int *intervalOn* = 500, int *intervalOff* = 500, string *ident* = " ")

Control the MSR LED

Controls the MSR / ICC LED This API not recommended to control ICC LED

Parameters

<i>control</i>	<ul style="list-style-type: none"> • 0x00 = off, • 0x01 = RED Solid, • 0x02 = RED Blink, • 0x11 = GREEN Solid, • 0x12 = GREEN Blink, • 0x21 = BLUE Solid, • 0x22 = BLUE Blink,
<i>intervalOn</i>	Blink interval on time last, in ms (500 = 500 ms, valid from 200 to 2000)
<i>intervalOff</i>	Blink interval off time last, in ms (500 = 500 ms, valid from 200 to 2000)

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString

21.2.1.11 RETURN_CODE IDTechSDK.IDT_BTMag.device_getAnyRKIStatus (bool *isMultiKey*, ref string *status*, ref Dictionary< string, RKI_KEY_TYPE > *keys*, string *ident* = " ")

Get RKI Status Polls the RKI servers to validate device RKI eligiblity. Note: if device type is known in advance (production or demo device), it is more efficient to use device_getRKIStatus instead

Parameters

<i>isMultiKey</i>	True = MultiKey, False = SingleKey
<i>status</i>	RKI status returned as a string
<i>keys</i>	Returns all available keys
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with errorCode.getErrorString(). When no data is available, return code = RETURN_CODE_NO_DATA_AVAILABLE

21.2.1.12 RETURN_CODE IDTechSDK.IDT_BTMag.device_getConfigurationFromMemory (ref string *json*, string *ident* = " ")

Read Device Configuration From Memory Retrieves the ViVOconfig configuration data from memory after a successful device_readConfigurationToMemory

Parameters

<i>json</i>	Returns the .json file data that can be saved to disk
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with `errorCode.getErrorString()`. When no data is available, return code = RETURN_CODE_NO_DATA_AVAILABLE

21.2.1.13 RETURN_CODE IDTechSDK.IDT_BTMag.device_getFirmwareVersion (ref string *response*, string *ident* = " ")

Polls device for Firmware Version

Parameters

<i>response</i>	Response returned of Firmware Version
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with `device_getResponseCodeString`

21.2.1.14 RETURN_CODE IDTechSDK.IDT_BTMag.device_getKeyStatus (ref byte[] *status*, string *ident* = " ")

Get Key Status

Gets the status of loaded keys

Parameters

<i>status</i>	when the newFormat is false, data format as follows. byte 0: PIN DUKPT Key, Does not support, always 0 byte 1: PIN Master Key, Does not support, always 0 byte 2: PIN Session Key, Does not support, always 0 byte 3: Account/MSR DUKPT Key, 1 Exists, 0 None, 0xFF STOP byte 4: Account/ICC DUKPT Key, 1 Exists, 0 None, 0xFF STOP byte 5: Admin DUKPT Key, 1 Exists, 0 None, 0xFF STOP
---------------	--

when the newFormat is true, data format as follows. [Block Length] [KeyStatusBlock1] [KeyStatusBlock2] ... [KeyStatusBlockN] Where: [Block Length] is 2 bytes, format is Len_L Len_H, is KeyStatusBlock Number [KeyStatusBlockX] is 4 bytes, format is [Key Index and Key Name] [key slot] [key status]: [Key Index and Key Name] is 1 byte. Please refer to following table 0x14 LCL-KEK to Encrypt Other Keys 0x02 Data encryption Key to Encrypt ICC/MSR 0x05 MAC DUKPT Key for Host-Device – MAC Verification 0x05 MTK DUKPT Key for TTK Self-Test 0x0C RKI-KEK for Remote Key Injection [key slot] is 2 bytes. Range is 0 – 9999 the MTK DUKPT Key slot is 16, the others are all 0 [key status] is 1 byte. 0 – Not Exist 1 – Exist 0xFF – (Stop. Only Valid for DUKPT Key)

Parameters

<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.
--------------	--

Returns

RETURN_CODE: Values can be parsed with `device_getResponseCodeString`

21.2.1.15 string IDTechSDK.IDT_BTMag.device_getResponseCodeString (RETURN_CODE *eCode*)

Get the description of response result.

Parameters

<i>eCode</i>	the response result.
--------------	----------------------

Return values

<i>the</i>	string for description of response result
------------	---

- case 0: "no error, beginning task";
- case 1: "no response from reader";
- case 2: "invalid response data";
- case 3: "time out for task or CMD";
- case 4: "wrong parameter";
- case 5: "SDK is doing MSR or ICC task";
- case 6: "SDK is doing PINPad task";
- case 7: "SDK is doing CTLS task";
- case 8: "SDK is doing EMV task";
- case 9: "SDK is doing Other task";
- case 10: "err response or data";
- case 11: "no reader attached";
- case 12: "mono audio is enabled";
- case 13: "did connection";
- case 14: "audio volume is too low";
- case 15: "task or CMD be canceled";
- case 16: "UF wrong string format";
- case 17: "UF file not found";
- case 18: "UF wrong file format";
- case 19: "Attempt to contact online host failed";
- case 20: "Attempt to perform RKI failed";
- case 0x300: "Key Type(TDES) of Session Key is not same as the related Master Key.";
- case 0x400: "Related Key was not loaded.";
- case 0x500: "Key Same.";
- case 0x501: "Key is all zero";
- case 0x502: "TR-31 format error";
- case 0x702: "PAN is Error Key.";
- case 0x705: "No Internal MSR PAN (or Internal MSR PAN is erased timeout)";
- case 0X0C01: "Incorrect Frame Tag";

- case 0X0C02: "Incorrect Frame Type";
- case 0X0C03: "Unknown Frame Type";
- case 0X0C04: "Unknown Command";
- case 0X0C05: "Unknown Sub-Command";
- case 0X0C06: "CRC Error";
- case 0X0C07: "Failed";
- case 0X0C08: "Timeout";
- case 0X0C0A: "Incorrect Parameter";
- case 0X0C0B: "Command Not Supported";
- case 0X0C0C: "Sub-Command Not Supported";
- case 0X0C0D: "Parameter Not Supported / Status Abort Command";
- case 0X0C0F: "Sub-Command Not Allowed";
- case 0X0D01: "Incorrect Header Tag";
- case 0X0D02: "Unknown Command";
- case 0X0D03: "Unknown Sub-Command";
- case 0X0D04: "CRC Error in Frame";
- case 0X0D05: "Incorrect Parameter";
- case 0X0D06: "Parameter Not Supported";
- case 0X0D07: "Mal-formatted Data";
- case 0X0D08: "Timeout";
- case 0X0D0A: "Failed / NACK";
- case 0X0D0B: "Command not Allowed";
- case 0X0D0C: "Sub-Command not Allowed";
- case 0X0D0D: "Buffer Overflow (Data Length too large for reader buffer)";
- case 0X0D0E: "User Interface Event";
- case 0X0D11: "Communication type not supported, VT-1, burst, etc.";
- case 0X0D12: "Secure interface is not functional or is in an intermediate state.";
- case 0X0D13: "Data field is not mod 8";
- case 0X0D14: "Pad 0x80 not found where expected";
- case 0X0D15: "Specified key type is invalid";
- case 0X0D1: "Could not retrieve key from the SAM(InitSecureComm)";
- case 0X0D17: "Hash code problem";
- case 0X0D18: "Could not store the key into the SAM(InstallKey)";
- case 0X0D19: "Frame is too large";
- case 0X0D1A: "Unit powered up in authentication state but POS must resend the InitSecureComm command";

- case 0X0D1B: "The EEPROM may not be initialized because SecCommInterface does not make sense";
- case 0X0D1C: "Problem encoding APDU";
- case 0X0D20: "Unsupported Index(ILM) SAM Transceiver error – problem communicating with the SAM(Key Mgr)";
- case 0X0D2: "Unexpected Sequence Counter in multiple frames for single bitmap(ILM) Length error in data returned from the SAM(Key Mgr)";
- case 0X0D22: "Improper bit map(ILM)";
- case 0X0D23: "Request Online Authorization";
- case 0X0D24: "ViVOCard3 raw data read successful";
- case 0X0D25: "Message index not available(ILM) ViVOcomm activate transaction card type(ViVOcomm)";
- case 0X0D26: "Version Information Mismatch(ILM)";
- case 0X0D27: "Not sending commands in correct index message index(ILM)";
- case 0X0D28: "Time out or next expected message not received(ILM)";
- case 0X0D29: "ILM languages not available for viewing(ILM)";
- case 0X0D2A: "Other language not supported(ILM)";
- case 0X0D41: "Unknown Error from SAM";
- case 0X0D42: "Invalid data detected by SAM";
- case 0X0D43: "Incomplete data detected by SAM";
- case 0X0D44: "Reserved";
- case 0X0D45: "Invalid key hash algorithm";
- case 0X0D46: "Invalid key encryption algorithm";
- case 0X0D47: "Invalid modulus length";
- case 0X0D48: "Invalid exponent";
- case 0X0D49: "Key already exists";
- case 0X0D4A: "No space for new RID";
- case 0X0D4B: "Key not found";
- case 0X0D4C: "Crypto not responding";
- case 0X0D4D: "Crypto communication error";
- case 0X0D4E: "Module-specific error for Key Manager";
- case 0X0D4F: "All key slots are full (maximum number of keys has been installed)";
- case 0X0D50: "Auto-Switch OK";
- case 0X0D51: "Auto-Switch failed";
- case 0X0D90: "Account DUKPT Key not exist";
- case 0X0D91: "Account DUKPT Key KSN exhausted";
- case 0x0D00: "This Key had been loaded.";
- case 0x0E00: "Base Time was loaded.";
- case 0x0F00: "Encryption Or Decryption Failed.";

- case 0x1000: "Battery Low Warning (It is High Priority Response while Battery is Low.);";
- case 0x1800: "Send "Cancel Command" after send "Get Encrypted PIN" &"Get Numeric "& "Get Amount""";
- case 0x1900: "Press "Cancel" key after send "Get Encrypted PIN" &"Get Numeric "& "Get Amount""";
- case 0x30FF: "Security Chip is not connect";
- case 0x3000: "Security Chip is deactivation & Device is In Removal Legally State.";
- case 0x3101: "Security Chip is activation & Device is In Removal Legally State.";
- case 0x5500: "No Admin DUKPT Key.";
- case 0x5501: "Admin DUKPT Key STOP.";
- case 0x5502: "Admin DUKPT Key KSN is Error.";
- case 0x5503: "Get Authentication Code1 Failed.";
- case 0x5504: "Validate Authentication Code Error.";
- case 0x5505: "Encrypt or Decrypt data failed.";
- case 0x5506: "Not Support the New Key Type.";
- case 0x5507: "New Key Index is Error.";
- case 0x5508: "Step Error.";
- case 0x5509: "KSN Error";
- case 0x550A: "MAC Error.";
- case 0x550B: "Key Usage Error.";
- case 0x550C: "Mode Of Use Error.";
- case 0x550F: "Other Error.";
- case 0x6000: "Save or Config Failed / Or Read Config Error.";
- case 0x6200: "No Serial Number.";
- case 0x6900: "Invalid Command - Protocol is right, but task ID is invalid.";
- case 0x6A01: "Unsupported Command – Protocol and task ID are right, but command is invalid – In this State";
- case 0x6A00: "Unsupported Command - Protocol and task ID are right, but command is invalid.";
- case 0x6B00: "Unknown parameter in command - Protocol task ID and command are right, but parameter is invalid.";
- case 0x6C00: "Unknown parameter in command – Protocol task ID and command are right, but length is out of the requirement.";
- case 0x7200: "Device is suspend (MKSX suspend or press password suspend).";
- case 0x7300: "PIN DUKPT is STOP (21 bit 1).";
- case 0x7400: "Device is Busy.";
- case 0xE100: "Can not enter sleep mode";
- case 0xE200: "File has existed";
- case 0xE300: "File has not existed";
- case 0xE313: "IO line low -- Card error after session start";

- case 0xE400: "Open File Error";
- case 0xE500: "SmartCard Error";
- case 0xE600: "Get MSR Card data is error";
- case 0xE700: "Command time out";
- case 0xE800: "File read or write is error";
- case 0xE900: "Active 1850 error!";
- case 0xEA00: "Load bootloader error";
- case 0xEF00: "Protocol Error- STX or ETX or check error.";
- case 0xEB00: "Picture is not exist";
- case 0x2C02: "No Microprocessor ICC seated";
- case 0x2C06: "no card seated to request ATR";
- case 0x2D01: "Card Not Supported,";
- case 0x2D03: "Card Not Supported, wants CRC";
- case 0x690D: "Command not supported on reader without ICC support";
- case 0x8100: "ICC error time out on power-up";
- case 0x8200: "invalid TS character received - Wrong operation step";
- case 0x8300: "Decode MSR Error";
- case 0x8400: "TriMagII no Response";
- case 0x8500: "No Swipe MSR Card";
- case 0x8510: "No Financial Card";
- case 0x8600: "Unsupported F, D, or combination of F and D";
- case 0x8700: "protocol not supported EMV TD1 out of range";
- case 0x8800: "power not at proper level";
- case 0x8900: "ATR length too long";
- case 0x8B01: "EMV invalid TA1 byte value";
- case 0x8B02: "EMV TB1 required";
- case 0x8B03: "EMV Unsupported TB1 only 00 allowed";
- case 0x8B04: "EMV Card Error, invalid BWI or CWI";
- case 0x8B06: "EMV TB2 not allowed in ATR";
- case 0x8B07: "EMV TC2 out of range";
- case 0x8B08: "EMV TC2 out of range";
- case 0x8B09: "per EMV96 TA3 must be > 0xF";
- case 0x8B10: "ICC error on power-up";
- case 0x8B11: "EMV T=1 then TB3 required";
- case 0x8B12: "Card Error, invalid BWI or CWI";
- case 0x8B13: "Card Error, invalid BWI or CWI";

- case 0x8B17: "EMV TC1/TB3 conflict-";
- case 0x8B20: "EMV TD2 out of range must be T=1";
- case 0x8C00: "TCK error";
- case 0xA304: "connector has no voltage setting";
- case 0xA305: "ICC error on power-up invalid (SBLK(IFSD) exchange";
- case 0xE301: "ICC error after session start";
- case 0xFF00: "Request to go online";
- case 0xFF01: "EMV: Accept the offline transaction";
- case 0xFF02: "EMV: Decline the offline transaction";
- case 0xFF03: "EMV: Accept the online transaction";
- case 0xFF04: "EMV: Decline the online transaction";
- case 0xFF05: "EMV: Application may fallback to magstripe technology";
- case 0xFF06: "EMV: ICC detected tah the conditions of use are not satisfied";
- case 0xFF07: "EMV: ICC didn't accept transaction";
- case 0xFF08: "EMV: Transaction was cancelled";
- case 0xFF09: "EMV: Application was not selected by kernel or ICC format error or ICC missing data error";
- case 0xFF0A: "EMV: Transaction is terminated";
- case 0xFF0B: "EMV: Other EMV Error";
- case 0xFFFF: "NO RESPONSE";
- case 0xF002: "ICC communication timeout";
- case 0xF003: "ICC communication Error";
- case 0xF00F: "ICC Card Seated and Highest Priority, disable MSR work request";
- case 0xF200: "AID List / Application Data is not exist";
- case 0xF201: "Terminal Data is not exist";
- case 0xF202: "TLV format is error";
- case 0xF203: "AID List is full";
- case 0xF204: "Any CA Key is not exist";
- case 0xF205: "CA Key RID is not exist";
- case 0xF206: "CA Key Index it not exist";
- case 0xF207: "CA Key is full";
- case 0xF208: "CA Key Hash Value is Error";
- case 0xF209: "Transaction format error";
- case 0xF20A: "The command will not be processing";
- case 0xF20B: "CRL is not exist";
- case 0xF20C: "CRL number exceed max number";
- case 0xF20D: "Amount,Other Amount,Trasaction Type are missing";

- case 0xF20E: "The Identification of algorithm is mistake";
- case 0xF20F: "No Financial Card";
- case 0xF210: "In Encrypt Result state, TLV total Length is greater than Max Length";
- case 0x1001: "INVALID ARG";
- case 0x1002: "FILE_OPEN_FAILED";
- case 0x1003: "FILE OPERATION_FAILED";
- case 0x2001: "MEMORY_NOT_ENOUGH";
- case 0x3002: "SMARTCARD_FAIL";
- case 0x3003: "SMARTCARD_INIT_FAILED";
- case 0x3004: "FALLBACK_SITUATION";
- case 0x3005: "SMARTCARD_ABSENT";
- case 0x3006: "SMARTCARD_TIMEOUT";
- case 0x5001: "EMV_PARSING_TAGS_FAILED";
- case 0x5002: "EMV_DUPLICATE_CARD_DATA_ELEMENT";
- case 0x5003: "EMV_DATA_FORMAT_INCORRECT";
- case 0x5004: "EMV_NO_TERM_APP";
- case 0x5005: "EMV_NO_MATCHING_APP";
- case 0x5006: "EMV_MISSING_MANDATORY_OBJECT";
- case 0x5007: "EMV_APP_SELECTION_RETRY";
- case 0x5008: "EMV_GET_AMOUNT_ERROR";
- case 0x5009: "EMV_CARD_REJECTED";
- case 0x5010: "EMV_AIP_NOT_RECEIVED";
- case 0x5011: "EMV_AFL_NOT_RECEIVED";
- case 0x5012: "EMV_AFL_LEN_OUT_OF_RANGE";
- case 0x5013: "EMV_SFI_OUT_OF_RANGE";
- case 0x5014: "EMV_AFL_INCORRECT";
- case 0x5015: "EMV_EXP_DATE_INCORRECT";
- case 0x5016: "EMV_EFF_DATE_INCORRECT";
- case 0x5017: "EMV_ISS_COD_TBL_OUT_OF_RANGE";
- case 0x5018: "EMV_CRYPTOGAM_TYPE_INCORRECT";
- case 0x5019: "EMV_PSE_NOT_SUPPORTED_BY_CARD";
- case 0x5020: "EMV_USER_SELECTED_LANGUAGE";
- case 0x5021: "EMV_SERVICE_NOT_ALLOWED";
- case 0x5022: "EMV_NO_TAG_FOUND";
- case 0x5023: "EMV_CARD_BLOCKED";
- case 0x5024: "EMV_LEN_INCORRECT";

- case 0x5025: "CARD_COM_ERROR";
- case 0x5026: "EMV_TSC_NOT_INCREASED";
- case 0x5027: "EMV_HASH_INCORRECT";
- case 0x5028: "EMV_NO_ARC";
- case 0x5029: "EMV_INVALID_ARC";
- case 0x5030: "EMV_NO_ONLINE_COMM";
- case 0x5031: "TRAN_TYPE_INCORRECT";
- case 0x5032: "EMV_APP_NO_SUPPORT";
- case 0x5033: "EMV_APP_NOT_SELECT";
- case 0x5034: "EMV_LANG_NOT_SELECT";
- case 0x5035: "EMV_NO_TERM_DATA";
- case 0x6001: "CVM_TYPE_UNKNOWN";
- case 0x6002: "CVM_AIP_NOT_SUPPORTED";
- case 0x6003: "CVM_TAG_8E_MISSING";
- case 0x6004: "CVM_TAG_8E_FORMAT_ERROR";
- case 0x6005: "CVM_CODE_IS_NOT_SUPPORTED";
- case 0x6006: "CVM_COND_CODE_IS_NOT_SUPPORTED";
- case 0x6007: "NO_MORE_CVM";
- case 0x6008: "PIN_BYPASSED_BEFORE";
- case 0x7001: "PK_BUFFER_SIZE_TOO_BIG";
- case 0x7002: "PK_FILE_WRITE_ERROR";
- case 0x7003: "PK_HASH_ERROR";
- case 0x8001: "NO_CARD_HOLDER_CONFIRMATION";
- case 0x8002: "GET_ONLINE_PIN";
- case 0xD000: "Data not exist";
- case 0xD001: "Data access error";
- case 0xD100: "RID not exist";
- case 0xD101: "RID existed";
- case 0xD102: "Index not exist";
- case 0xD200: "Maximum exceeded";
- case 0xD201: "Hash error";
- case 0xD205: "System Busy";
- case 0x0E01: "Unable to go online";
- case 0x0E02: "Technical Issue";
- case 0x0E03: "Declined";
- case 0x0E04: "Issuer Referral transaction";

- case 0x0F01: "Decline the online transaction";
- case 0x0F02: "Request to go online";
- case 0x0F03: "Transaction is terminated";
- case 0x0F05: "Application was not selected by kernel or ICC format error or ICC missing data error";
- case 0x0F07: "ICC didn't accept transaction";
- case 0x0F0A: "Application may fallback to magstripe technology";
- case 0x0F0C: "Transaction was cancelled";
- case 0x0F0D: "Timeout";
- case 0x0F0F: "Other EMV Error";
- case 0x0F10: "Accept the offline transaction";
- case 0x0F11: "Decline the offline transaction";
- case 0x0F21: "ICC detected tah the conditions of use are not satisfied";
- case 0x0F22: "No app were found on card matching terminal configuration";
- case 0x0F23: "Terminal file does not exist";
- case 0x0F24: "CAPK file does not exist";
- case 0x0F25: "CRL Entry does not exist";
- case 0x0FFE: "code when blocking is disabled";
- case 0x0FFF: "code when command is not applicable on the selected device";
- case 0xF005: "ICC Encrypted C-APDU Data Structure Length Error Or Format Error.";
- case 0xBBE0: "CM100 Success";
- case 0xBBE1: "CM100 Parameter Error";
- case 0xBBE2: "CM100 Low Output Buffer";
- case 0xBBE3: "CM100 Card Not Found";
- case 0xBBE4: "CM100 Collision Card Exists";
- case 0xBBE5: "CM100 Too Many Cards Exist";
- case 0xBBE6: "CM100 Saved Data Does Not Exist";
- case 0xBBE8: "CM100 No Data Available";
- case 0xBBE9: "CM100 Invalid CID Returned";
- case 0xBBEA: "CM100 Invalid Card Exists";
- case 0xBBEC: "CM100 Command Unsupported";
- case 0xBBED: "CM100 Error In Command Process";
- case 0xBBEE: "CM100 Invalid Command";
- case 0X9031: "Unknown command";
- case 0X9032: "Wrong parameter (such as the length of the command is incorrect)";
- case 0X9038: "Wait (the command couldn't be finished in BWT)";
- case 0X9039: "Busy (a previously command has not been finished)";

- case 0X903A: "Number of retries over limit";
- case 0X9040: "Invalid Manufacturing system data";
- case 0X9041: "Not authenticated";
- case 0X9042: "Invalid Master DUKPT Key";
- case 0X9043: "Invalid MAC Key";
- case 0X9044: "Reserved for future use";
- case 0X9045: "Reserved for future use";
- case 0X9046: "Invalid DATA DUKPT Key";
- case 0X9047: "Invalid PIN Pairing DUKPT Key";
- case 0X9048: "Invalid DATA Pairing DUKPT Key";
- case 0X9049: "No nonce generated";
- case 0X9949: "No GUID available. Perform getVersion first.";
- case 0X9950: "MAC Calculation unsuccessful. Check BDK value.";
- case 0X904A: "Not ready";
- case 0X904B: "Not MAC data";
- case 0X9050: "Invalid Certificate";
- case 0X9051: "Duplicate key detected";
- case 0X9052: "AT checks failed";
- case 0X9053: "TR34 checks failed";
- case 0X9054: "TR31 checks failed";
- case 0X9055: "MAC checks failed";
- case 0X9056: "Firmware download failed";
- case 0X9060: "Log is full";
- case 0X9061: "Removal sensor unengaged";
- case 0X9062: "Any hardware problems";
- case 0X9070: "ICC communication timeout";
- case 0X9071: "ICC data error (such check sum error)";
- case 0X9072: "Smart Card not powered up";

21.2.1.16 **RETURN_CODE** IDTechSDK.IDT_BTMag.device_getRKIStatus (bool *isProd*, bool *isMultiKey*, ref string *status*, ref Dictionary< string, RKI_KEY_TYPE > *keys*, string *ident* = " ")

Get RKI Status From Specified Server Polls the RKI server to validate device RKI eligibility

Parameters

<i>isProd</i>	TRUE = poll production RKI server, FALSE = poll demo RKI server
<i>isMultiKey</i>	True = MultiKey, False = SingleKey
<i>status</i>	RKI status returned as a string
<i>keys</i>	Returns dictionary of available keys
<i>ident</i>	Device ID to send command to. If not specified, current SDK device ID will be used

Returns

RETURN_CODE: Values can be parsed with `errorCode.getErrorString()`. When no data is available, return code = RETURN_CODE_NO_DATA_AVAILABLE

21.2.1.17 RETURN_CODE IDTechSDK.IDT_BTMag.device_readConfigurationToMemory (string *memo*, Configs.InstallRules *rules*, List< Configs.DeviceCommand > *cmds*, string *ident* = " ", bool *isForeground* = false)

Read Device Configuration To Memory Executes a ViVOconfig read of device settings and stores the results in memory

Once a ViVOconfig read starts, all messages will be returned to the callback as DeviceState.ViVOconfig. The ViVOconfig operation is complete when the return code from a DeviceState.ViVOconfig message is one of the following values:

- RETURN_CODE_SDK_VIVOCONFIG_READ_CONFIG_SUCCESS = Read process completed successfully. Execute `device_getConfigurationFromMemory` to retrieve the configuration data
- RETURN_CODE_SDK_VIVOCONFIG_READ_CONFIG_FAILED = Read process FAILED.

Parameters

<i>memo</i>	Optional: populates the memo field of the captured .json file
<i>rules</i>	Optional: populates the .json file with available install rules
<i>cmds</i>	Optional: populates the .json file with commands to execute
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.
<i>isForeground</i>	If TRUE, will block program until update complete. Otherwise, FALSE performs update on background.

Returns

RETURN_CODE: Values can be parsed with `errorCode.getErrorString()`. When no data is available, return code = RETURN_CODE_NO_DATA_AVAILABLE

21.2.1.18 RETURN_CODE IDTechSDK.IDT_BTMag.device_rebootDevice (string *ident* = " ")

Reboot Device

Executes a command to restart the device.

Parameters

<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.
--------------	--

Returns

RETURN_CODE: Values can be parsed with `device_getResponseCodeString`

21.2.1.19 RETURN_CODE IDTechSDK.IDT_BTMag.device_RemoteKeyInjection (RKI_KEY_TYPE *type*, string *keyName*, string *ident* = " ", bool *performOnForeground* = false)

Remote Key Injection Performs a remote key injection for the device

Parameters

<i>type</i>	Remote Key Injection Type
<i>keyName</i>	Name of key (optional)
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.
<i>performOnForeground</i>	TRUE = block during process (default is false)

Returns

RETURN_CODE: Values can be parsed with `errorCode.getErrorString()`. When no data is available, return code = RETURN_CODE_NO_DATA_AVAILABLE

21.2.1.20 RETURN_CODE IDTechSDK.IDT_BTMag.device_sendConfiguration (string filename, VC_OPERATION_TYPE type, bool matchFW, string ident = " ", bool isForeground = false)

Send Configuration Executes a ViVOconfig update or verify to a device

Once a ViVOconfig write or verify starts, all messages will be returned to the callback as DeviceState.ViVOconfig. The ViVOconfig operation is complete when the return code from a DeviceState.ViVOconfig message is one of the following values:

- RETURN_CODE_SDK_VIVOCONFIG_VERIFY_SUCCESS = Verification process completed successfully. No differences found.
- RETURN_CODE_SDK_VIVOCONFIG_VERIFY_SUCCESS_WITH_WARNING = Verification process completed with warnings.
- RETURN_CODE_SDK_VIVOCONFIG_VERIFY_FAILED = Verification process FAILED
- RETURN_CODE_SDK_VIVOCONFIG_WRITE_SUCCESS = Write process completed successfully.
- RETURN_CODE_SDK_VIVOCONFIG_WRITE_SUCCESS_WITH_WARNING = Write process completed with warnings
- RETURN_CODE_SDK_VIVOCONFIG_WRITE_FAILED = Write process FAILED

Parameters

<i>filename</i>	The .json configuration file to update the device with NOTE: You can also use this parameter to pass the path of where the .json file is located at on local storage instead
<i>filename</i>	The name of the .json configuration file
<i>type</i>	VC_OPERATION_TYPE

- VC_OPERATION_TYPE_WRITE = Write File To Device, Hash must validate
- VC_OPERATION_TYPE_VERIFY = Verify Device With File
- VC_OPERATION_TYPE_WRITE_IGNORE_HASH = Write File To Device, Ignore Hash
- VC_OPERATION_TYPE_WRITE_FIX_HASH = Write File To Device, Fix Hash If Necessary

Parameters

<i>matchFW</i>	TRUE = Device FW must match file FW, FALSE = Don't validate device FW against file
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.
<i>isForeground</i>	If TRUE, will block program until update complete. Otherwise, FALSE performs update on background.

Returns

RETURN_CODE: Values can be parsed with `errorCode.getErrorString()`. When no data is available, return code = RETURN_CODE_NO_DATA_AVAILABLE

21.2.1.21 RETURN_CODE IDTechSDK.IDT_BTMag.device_sendConfigurationFromZip (byte[] zip, string filename, VC_OPERATION_TYPE type, bool matchFW, string ident = "", bool isForeground = false)

Send Configuration From Zip Executes a ViVOconfig update or verify to a device using files from a .zip archive

Once a ViVOconfig write or update starts, all messages will be returned to the callback as DeviceState.ViVOconfig. The ViVOconfig operation is complete when the return code from a DeviceState.ViVOconfig message is one of the following values:

- RETURN_CODE_SDK_VIVOCONFIG_VERIFY_SUCCESS = Verification process completed successfully. No differences found.
- RETURN_CODE_SDK_VIVOCONFIG_VERIFY_SUCCESS_WITH_WARNING = Verification process completed with warnings.
- RETURN_CODE_SDK_VIVOCONFIG_VERIFY_FAILED = Verification process FAILED
- RETURN_CODE_SDK_VIVOCONFIG_WRITE_SUCCESS = Write process completed successfully.
- RETURN_CODE_SDK_VIVOCONFIG_WRITE_SUCCESS_WITH_WARNING = Write process completed with warnings
- RETURN_CODE_SDK_VIVOCONFIG_WRITE_FAILED = Write process FAILED

Parameters

<i>zip</i>	.Zip archive containing the .json configuration file and also any associated image assets NOTE: You can also use this parameter to pass the path of where the .zip file is located at on local storage instead
<i>filename</i>	The name of the .json configuration file
<i>type</i>	VC_OPERATION_TYPE

- VC_OPERATION_TYPE_WRITE = Write File To Device, Hash must validate
- VC_OPERATION_TYPE_VERIFY = Verify Device With File
- VC_OPERATION_TYPE_WRITE_IGNORE_HASH = Write File To Device, Ignore Hash
- VC_OPERATION_TYPE_WRITE_FIX_HASH = Write File To Device, Fix Hash If Necessary

Parameters

<i>matchFW</i>	TRUE = Device FW must match file FW, FALSE = Don't validate device FW against file
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.
<i>isForeground</i>	If TRUE, will block program until update complete. Otherwise, FALSE performs update on background.

Returns

RETURN_CODE: Values can be parsed with `errorCode.getErrorString()`. When no data is available, return code = RETURN_CODE_NO_DATA_AVAILABLE

21.2.1.22 **RETURN_CODE** IDTechSDK.IDT_BTMag.device_sendDataCommand (string *cmd*, bool *calcLRC*, ref byte[] *response*, string *ident* = " ")

Send a NSData object to device

Sends a command represented by the provide NSData object to the device through the accessory protocol.

Parameters

<i>cmd</i>	NSData representation of command to execute
<i>calcLRC</i>	If TRUE, this will wrap command with start/length/lrc/sum/end: '{STX}{Len_Low}{Len_High} data {CheckLRC} {CheckSUM} {ETX}'
<i>response</i>	Response data
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString

21.2.1.23 **RETURN_CODE** IDTechSDK.IDT_BTMag.device_sendPAE (string *command*, ref string *response*, int *timeout*, string *ident* = " ")

Send Payment Application Engine Command

Executes a PAE command

Parameters

<i>command</i>	ASCII command string, should start with "*PAE"
<i>response</i>	command response
<i>timeout</i>	timeout waiting for PAE response
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString

21.2.1.24 **RETURN_CODE** IDTechSDK.IDT_BTMag.device_setDateTime (string *ident* = " ")

Set Date Time

Set current system date and time to the device.

Parameters

<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.
--------------	--

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString

21.2.1.25 **RETURN_CODE** IDTechSDK.IDT_BTMag.device_updateFirmwareFromZip (byte[] *zipfile*, string *ident* = " ", bool *isForeground* = false)

Update Firmware From Zip

Performs one or more device firmware updates from firmware files passed as a compressed archive with a valid configuration file (.json format)

Parameters

<i>zipfile</i>	Zip file containing on or more firmware update files and a .json configuration file
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.
<i>isForeground</i>	If TRUE, will block program until firmware update complete. Otherwise, FALSE performs FW update on background.

Returns

RETURN_CODE: Values can be parsed with `errorCode.getErrorString(string ident = "")`

21.2.1.26 **static int** IDTechSDK.IDT_BTMag.getCommandTimeout (string *ident* = " ") [static]

Get Command Timeout

Gets the default timeout (in milliseconds) waiting for a blocking command response

Parameters

<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.
--------------	--

Return values

<i>time</i>	Time
-------------	------

21.2.1.27 **RETURN_CODE** IDTechSDK.IDT_BTMag.msr_cancelMSRSwipe (string *ident* = " ")

Disable MSR Swipe Cancels MSR swipe request.

Parameters

<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.
--------------	--

Returns

RETURN_CODE: Values can be parsed with `device_getResponseCodeString`

21.2.1.28 **RETURN_CODE** IDTechSDK.IDT_BTMag.msr_captureMode (bool *isBufferMode*, bool *withNotification* = false, string *ident* = " ")

Set MSR Capture Mode.

For Non-SRED Augusta Only

Switch between Auto mode and Buffer mode. Auto mode only available on KB interface

Parameters

<i>isBufferMode</i>	<ul style="list-style-type: none"> • true: Enter into Buffer mode. • false: Enter into Auto mode. KB mode only. Swipes automatically captured and sent to keyboard buffer
<i>withNotification</i>	<ul style="list-style-type: none"> • true: with notification when swiped MSR data. • false: without notification when swiped MSR data.
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString

21.2.1.29 RETURN_CODE IDTechSDK.IDT_BTMag.msr_disable (string *ident* = " ")

Disable MSR function.

Parameters

<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.
--------------	--

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString

21.2.1.30 RETURN_CODE IDTechSDK.IDT_BTMag.msr_getClearPANID (ref byte *value*, string *ident* = " ")

Get Clear PAN Digits

Returns the number of digits that begin the PAN that will be in the clear

Parameters

<i>value</i>	Number of digits in clear. Values are char '0' - '6':
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString

21.2.1.31 RETURN_CODE IDTechSDK.IDT_BTMag.msr_getExpirationMask (ref byte *value*, string *ident* = " ")

Get Expiration Masking

Get the flag that determines if to mask the expiration date

Parameters

<i>value</i>	'0' = masked, '1' = not-masked
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString

21.2.1.32 RETURN_CODE IDTechSDK.IDT_BTMag.msr_getFunctionStatus (ref bool *enabled*, ref bool *isBufferMode*, ref bool *withNotification*, string *ident* = " ")

Get MSR Function status Get MSR Function status about enable/disable and with or without seated notification

Parameters

<i>enabled</i>	<ul style="list-style-type: none"> • true: MSR Function enabled. • false: MSR Function disabled.
<i>isBufferMode</i>	<ul style="list-style-type: none"> • true: in the buffer mode. • false: in the auto mode.
<i>withNotification</i>	<ul style="list-style-type: none"> • true: with notification when swiped MSR Card. • false: without notification when swiped MSR Card.
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString

21.2.1.33 RETURN_CODE IDTechSDK.IDT_BTMag.msr_getSetting (byte *setting*, ref byte *value*, string *ident* = " ")

Get Single MSR Setting value

Returns the encryption used for sweep data

Parameters

<i>setting</i>	MSR Setting to retrieve
<i>value</i>	MSR Setting value
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString

21.2.1.34 RETURN_CODE IDTechSDK.IDT_BTMag.msr_getSettings (byte *setting*, ref byte[] *value*, string *ident* = " ")

Get Multi MSR Setting value

Returns the encryption used for sweep data

Parameters

<i>setting</i>	MSR Setting to retrieve
<i>value</i>	MSR Setting value
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString

21.2.1.35 RETURN_CODE IDTechSDK.IDT_BTMag.msr_getSwipeEncryption (ref byte *encryption*, string *ident* = " ")

Get Swipe Data Encryption

For Non-SRED Augusta Only

Returns the encryption used for swipe data

Parameters

<i>encryption</i>	1 = TDES, 2 = AES, 0 = NONE
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString

21.2.1.36 RETURN_CODE IDTechSDK.IDT_BTMag.msr_getSwipeForcedEncryptionOption (ref byte *option*, string *ident* = " ")

Get Swipe Data Encryption

Gets the swipe force encryption options

Parameters

<i>option</i>	Byte using lower four bits as flags. 0 = Force Encryption Off, 1 = Force Encryption On bit0 = Track 1 bit1 = Track 2 bit2 = Track 3 bit4 = Track 3 Card Option 0
---------------	--

Example: Response 0x03 = Track1/Track2 Forced Encryption, Track3/Track3-0 no Forced Encryption

Parameters

<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used
--------------	---

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString

21.2.1.37 RETURN_CODE IDTechSDK.IDT_BTMag.msr_getSwipeMaskOption (ref byte *option*, string *ident* = " ")

Get Swipe Mask Option

Gets the swipe mask/clear data sending option

Parameters

<i>option</i>	Byte using lower three bits as flags. 0 = Mask Option Off, 1 = Mask Option On bit0 = Track 1 bit1 = Track 2 bit2 = Track 3
---------------	--

Example: Response 0x03 = Track1/Track2 Masked Option ON, Track3 Masked Option Off

Parameters

<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.
--------------	--

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString

21.2.1.38 RETURN_CODE IDTechSDK.IDT_BTMag.msr_RetrieveWhiteList (ref byte[] *value*, string *ident* = " ")

Retrieve MSR White List

For Non-SRED Augusta Only

Parameters

<i>value</i>	is the white list data which is ASN.1 Block format
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString

21.2.1.39 RETURN_CODE IDTechSDK.IDT_BTMag.msr_setClearPANID (byte *val*, string *ident* = " ")

Set Clear PAN Digits

Sets the amount of digits shown in the clear (not masked) at the beginning of the returned PAN value

Parameters

<i>val</i>	Number of digits to show in clear. Range 0-6.
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString

21.2.1.40 RETURN_CODE IDTechSDK.IDT_BTMag.msr_setExpirationMask (bool *mask*, string *ident* = " ")

Set Expiration Masking

Sets the flag to mask the expiration date

Parameters

<i>mask</i>	TRUE = mask expiration
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString

21.2.1.41 RETURN_CODE IDTechSDK.IDT_BTMag.msr_setSetting (byte *setting*, byte *value*, string *ident* = " ")

Set Single MSR Setting value

Parameters

<i>setting</i>	MSR Setting to set
<i>value</i>	MSR Setting value
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString

21.2.1.42 RETURN_CODE IDTechSDK.IDT_BTMag.msr_setSettings (byte *setting*, byte[] *value*, string *ident* = " ")

Set Multi MSR Setting value

Parameters

<i>setting</i>	MSR Setting to set
<i>value</i>	MSR Setting value
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString

21.2.1.43 RETURN_CODE IDTechSDK.IDT_BTMag.msr_setSwipeEncryption (byte *encryption*, string *ident* = " ")

Set Swipe Data Encryption

For Non-SRED Augusta Only

Sets the swipe encryption method

Parameters

<i>encryption</i>	1 = TDES, 2 = AES
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString

21.2.1.44 RETURN_CODE IDTechSDK.IDT_BTMag.msr_setSwipeForcedEncryptionOption (bool *track1*, bool *track2*, bool *track3*, bool *track3card0*, string *ident* = " ")

Set Swipe Force Encryption

Sets the swipe force encryption options

Parameters

<i>track1</i>	Force encrypt track 1
<i>track2</i>	Force encrypt track 2
<i>track3</i>	Force encrypt track 3
<i>track3card0</i>	Force encrypt track 3 when card type is 0
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString

21.2.1.45 RETURN_CODE IDTechSDK.IDT_BTMag.msr_setSwipeMaskOption (bool *track1*, bool *track2*, bool *track3*, string *ident* = " ")

Set Swipe Mask Option

Sets the swipe mask/clear data sending option

Parameters

<i>track1</i>	Mask track 1 allowed
<i>track2</i>	Mask track 2 allowed
<i>track3</i>	Mask track 3 allowed
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString

21.2.1.46 RETURN_CODE IDTechSDK.IDT_BTMag.msr_setWhiteList (byte[] *value*, byte[] *MAC*, string *ident* = " ")

Set MSR White List

For Non-SRED Augusta Only

Parameters

<i>value</i>	the white list data which is ASN.1 Block format
<i>MAC</i>	Message Authenticate Code. For non-PCI, use null
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString

21.2.1.47 RETURN_CODE IDTechSDK.IDT_BTMag.msr_startMSRSwipe (int *timeout*, string *ident* = " ")

Enable MSR Swipe

Enables MSR, waiting for swipe to occur. Allows track selection. Returns IDTMSRData instance to deviceDelegate::swipeMSRData:()

Parameters

<i>timeout</i>	Swipe Timeout Value
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString

21.2.1.48 RETURN_CODE IDTechSDK.IDT_BTMag.msr_switchUSBInterfaceMode (bool *blsUSBKeyboardMode*)

Switch the USB interface mode between USB HID and USB KB mode.

For Non-SRED Augusta Only

Parameters

<i>blsUSBKeyboardMode</i>	USB interface mode <ul style="list-style-type: none"> • true: Enter into USB Keyboard mode • false: Enter into USB HID mode
---------------------------	---

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString

21.2.1.49 RETURN_CODE IDTechSDK.IDT_BTMag.msr_switchUSBInterfaceMode (bool *blsUSBKeyboardMode*, ref string *ident*)

Switch the USB interface mode between USB HID and USB KB mode.

For Non-SRED Augusta Only

Parameters

<i>bIsUSBKeyboardMode</i>	USB interface mode <ul style="list-style-type: none"> • true: Enter into USB Keyboard mode • false: Enter into USB HID mode
<i>ident</i>	Device ID to send command to.

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString

21.2.1.50 `static String IDTechSDK.IDT_BTMag.SDK_Version () [static]`

SDK Version

- All Devices

Returns the current version of SDK

Returns

Framework version

21.2.1.51 `static void IDTechSDK.IDT_BTMag.setCallback (Callback my_Callback) [static]`

Set Callback

Sets the class callback

Parameters

<i>my_Callback</i>	The callback function to receive the response message from device. defined as follows. delegate void CallBack(IDT_DEVICE_Types sender, DeviceState state, byte[] data, IDTTransactionData card, EMV_Callback emvCallback, RETURN_CODE transactionResultCode);
--------------------	--

21.2.1.52 `static void IDTechSDK.IDT_BTMag.setCallback (IntPtr my_Callback, SynchronizationContext context) [static]`

Set Callback

Sets the class callback

Parameters

<i>my_Callback</i>	The callback function to receive the response message from device. defined as follows. public unsafe delegate void MFCCallBack(Parameters parameters);
<i>context</i>	The context of the UI thread

21.2.1.53 `static void IDTechSDK.IDT_BTMag.setCommandTimeout (int milliseconds, string ident = " ") [static]`

Set Command Timeout

Sets the default timeout (in milliseconds) waiting for a blocking command response

Parameters

<i>milliseconds</i>	Time
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

21.2.1.54 `static bool IDTechSDK.IDT_BTMag.useSerialPort (int port) [static]`

Use Serial Port Interface

Instructs SDK to attempt to use the Serial Port for communication with BTMag using default baud rate

Parameters

<i>port</i>	Serial Port to use. Example COM1 = 1.
-------------	---------------------------------------

Returns

bool TRUE=successful, FALSE=failure

21.2.1.55 `static bool IDTechSDK.IDT_BTMag.useSerialPort (int port, int baud) [static]`

Use Serial Port Interface with baud rate

Instructs SDK to attempt to use the Serial Port for communication with BTPay

Parameters

<i>port</i>	Serial Port to use. Example COM1 = 1.
<i>baud</i>	Baud rate to override default. Example 115200;

Returns

bool TRUE=successful, FALSE=failure

21.2.1.56 `static bool IDTechSDK.IDT_BTMag.useSerialPortLinux (string path) [static]`

Use Serial Port Interface on Linux

Instructs SDK to attempt to use the Serial Port for communication with BTMag using default baud rate on Linux implementations

Parameters

<i>path</i>	Path to use. Example /dev/ttyUSB*
-------------	-----------------------------------

Returns

bool TRUE=successful, FALSE=failure

21.2.1.57 static bool IDTechSDK.IDT_BTMag.useSerialPortLinux (string *path*, int *baud*) [static]

Use Serial Port Interface on Linux with baud rate

Instructs SDK to attempt to use the Serial Port for communication with BTPay on Linux implementations

Parameters

<i>path</i>	Path to use. Example /dev/ttyUSB*
<i>baud</i>	Baud rate to override default. Example 115200;

Returns

bool TRUE=successful, FALSE=failure

21.2.2 Property Documentation

21.2.2.1 IDT_BTMag IDTechSDK.IDT_BTMag.SharedController [static],[get]

Singleton Instance

Establishes an singleton instance of [IDT_BTPay](#) class.

Returns

Instance of [IDT_BTPay](#)

The documentation for this class was generated from the following file:

- /Users/randy/Repo/universal-sdk/IDTechSDK_STD/DeviceHeaders/IDT_BTMag.cs

21.3 IDTechSDK.IDT_BTPay Class Reference**Public Member Functions**

- RETURN_CODE [device_sendPAE](#) (string command, ref string response, int timeoutstring, string ident="")
- RETURN_CODE [emv_startTransaction](#) (double amount, double amtOther, int exponent, int type, int timeout, byte[] tags, bool forceOnline, bool isFastEMV=false, string ident="")
- RETURN_CODE [emv_authenticateTransaction](#) (byte[] updatedTLV, string ident="")
- RETURN_CODE [emv_completeTransaction](#) (bool commError, byte[] authCode, byte[] iad, byte[] tlvScripts, byte[] tlv, string ident="")
- RETURN_CODE [emv_callbackResponseLCD](#) (EMV_LCD_DISPLAY_MODE type, byte selection, string ident="")
- RETURN_CODE [emv_callbackResponsePIN](#) (EMV_PIN_MODE type, byte[] KSN, byte[] PIN, string ident="")
- RETURN_CODE [emv_callbackResponseMSR](#) (byte[] MSR, string ident="")
- RETURN_CODE [emv_cancelTransaction](#) (string ident="")
- RETURN_CODE [emv_getEMVKernelVersion](#) (ref string response, string ident="")
- RETURN_CODE [emv_getEMVKernelCheckValue](#) (ref string response, string ident="")
- RETURN_CODE [emv_getEMVConfigurationCheckValue](#) (ref string response, string ident="")
- RETURN_CODE [emv_retrieveTransactionResult](#) (byte[] tags, ref IDTTransactionData tlv, string ident="")

- RETURN_CODE [emv_removeApplicationData](#) (byte[] AID, string ident="")
- RETURN_CODE [emv_removeAllApplicationData](#) (string ident="")
- RETURN_CODE [emv_removeCAPK](#) (byte[] capk, string ident="")
- RETURN_CODE [emv_removeAllCAPK](#) (string ident="")
- RETURN_CODE [emv_removeCRL](#) (byte[] crlList, string ident="")
- RETURN_CODE [emv_removeAllCRL](#) (string ident="")
- RETURN_CODE [emv_removeTerminalData](#) (string ident="")
- RETURN_CODE [emv_retrieveAIDList](#) (ref byte[][] response, string ident="")
- RETURN_CODE [emv_retrieveApplicationData](#) (byte[] AID, ref byte[] tlv, string ident="")
- RETURN_CODE [emv_retrieveCAPK](#) (byte[] capk, ref byte[] key, string ident="")
- RETURN_CODE [emv_retrieveCAPKList](#) (ref byte[] keys, string ident="")
- RETURN_CODE [emv_retrieveCRLList](#) (ref byte[] list, string ident="")
- RETURN_CODE [emv_retrieveTerminalData](#) (ref byte[] tlv, string ident="")
- RETURN_CODE [emv_setApplicationData](#) (byte[] name, byte[] tlv, string ident="")
- RETURN_CODE [emv_setCAPK](#) (byte[] key, string ident="")
- RETURN_CODE [emv_setCRL](#) (byte[] list, string ident="")
- RETURN_CODE [emv_trySetTerminalData](#) (byte[] tlv, ref byte[] rejectedTLV, ref byte[] convertedTLV, bool overwrite=false, string ident="")
- RETURN_CODE [emv_setTerminalData](#) (byte[] tlv, string ident="")
- RETURN_CODE [emv_addTerminalData](#) (byte[] tlv, string ident="")
- RETURN_CODE [emv_setTerminalMajorConfiguration](#) (int configuration, string ident="")
- RETURN_CODE [config_getModelNumber](#) (ref string response, string ident="")
- RETURN_CODE [config_getSerialNumber](#) (ref string response, string ident="")
- bool [config_setCmdTimeOutDuration](#) (int newTimeOut, string ident="")
- RETURN_CODE [config_setLEDController](#) (bool firmwareControlMSRLED, bool firmwareControlICCL↵
ED=false, string ident="")
- RETURN_CODE [config_getLEDController](#) (ref bool firmwareControlMSRLED, ref bool firmwareControlICCL↵
LED, string ident="")
- RETURN_CODE [config_setBeeperController](#) (bool firmwareControlBeeper, string ident="")
- RETURN_CODE [config_getBeeperController](#) (ref bool firmwareControlBeeper, string ident="")
- RETURN_CODE [device_controlLED_ICC](#) (int controlMode, int interval, string ident="")
- RETURN_CODE [device_controlBeep](#) (int index, int frequency, int duration, string ident="")
- RETURN_CODE [device_controlLED](#) (byte indexLED, byte control, int intervalOn=500, int intervalOff=500, string ident="")
- RETURN_CODE [config_setEncryptionControl](#) (bool msr, bool icc, string ident="")
- RETURN_CODE [config_getEncryptionControl](#) (ref bool msr, ref bool icc, string ident="")
- RETURN_CODE [device_getKeyStatus](#) (ref byte[] status, string ident="")
- RETURN_CODE [device_setDateTime](#) (string ident="")
- RETURN_CODE [device_startRKI](#) (string ident="")
- RETURN_CODE [device_getFirmwareVersion](#) (ref string response, string ident="")
- string [device_getResponseCodeString](#) (RETURN_CODE eCode)
- RETURN_CODE [device_rebootDevice](#) (string ident="")
- RETURN_CODE [device_sendDataCommand](#) (string cmd, bool calcLRC, ref byte[] response, string ident="")
- RETURN_CODE [device_controlLED_MSR](#) (byte control, int intervalOn=500, int intervalOff=500, string ident="")
- RETURN_CODE [icc_enable](#) (bool withNotification, string ident="")
- RETURN_CODE [icc_disable](#) (string ident="")
- RETURN_CODE [icc_getFunctionStatus](#) (ref bool enabled, ref bool withNotification, string ident="")
- RETURN_CODE [icc_exchangeAPDU](#) (string c_APDU, ref byte[] response, string ident="")
- RETURN_CODE [icc_exchangeEncryptedAPDU](#) (string c_APDU, ref byte[] response, string ident="")
- RETURN_CODE [icc_getAPDU_KSN](#) (ref byte[] ksn, string ident="")
- RETURN_CODE [icc_getICCReaderStatus](#) (ref byte status, string ident="")
- RETURN_CODE [icc_getKeyFormatForICCDUKPT](#) (ref byte format, string ident="")
- RETURN_CODE [icc_getKeyTypeForICCDUKPT](#) (ref byte type, string ident="")
- RETURN_CODE [icc_powerOffICC](#) (string ident="")

- RETURN_CODE [icc_powerOnICC](#) (ref byte[] ATR, string ident="")
- RETURN_CODE [icc_setKeyFormatForICCDUKPT](#) (byte encryption, string ident="")
- RETURN_CODE [icc_setKeyTypeForICCDUKPT](#) (byte encryption, string ident="")
- RETURN_CODE [msr_captureMode](#) (bool isBufferMode, bool withNotification=false, string ident="")
- RETURN_CODE [msr_disable](#) (string ident="")
- RETURN_CODE [msr_getFunctionStatus](#) (ref bool enabled, ref bool isBufferMode, ref bool withNotification, string ident="")
- RETURN_CODE [msr_switchUSBInterfaceMode](#) (bool blsUSBKeyboardMode)
- RETURN_CODE [msr_switchUSBInterfaceMode](#) (bool blsUSBKeyboardMode, ref string ident)
- RETURN_CODE [msr_getClearPANID](#) (ref byte value, string ident="")
- RETURN_CODE [msr_getExpirationMask](#) (ref byte value, string ident="")
- RETURN_CODE [msr_getSwipeEncryption](#) (ref byte encryption, string ident="")
- RETURN_CODE [msr_getSwipeForcedEncryptionOption](#) (ref byte option, string ident="")
- RETURN_CODE [msr_getSwipeMaskOption](#) (ref byte option, string ident="")
- RETURN_CODE [msr_RetrieveWhiteList](#) (ref byte[] value, string ident="")
- RETURN_CODE [msr_getSetting](#) (byte setting, ref byte value, string ident="")
- RETURN_CODE [msr_getSettings](#) (byte setting, ref byte[] value, string ident="")
- RETURN_CODE [msr_setSetting](#) (byte setting, byte value, string ident="")
- RETURN_CODE [msr_setSettings](#) (byte setting, byte[] value, string ident="")
- RETURN_CODE [msr_setClearPANID](#) (byte val, string ident="")
- RETURN_CODE [msr_setExpirationMask](#) (bool mask, string ident="")
- RETURN_CODE [msr_setSwipeEncryption](#) (byte encryption, string ident="")
- RETURN_CODE [msr_setSwipeForcedEncryptionOption](#) (bool track1, bool track2, bool track3, bool track3card0, string ident="")
- RETURN_CODE [msr_setSwipeMaskOption](#) (bool track1, bool track2, bool track3, string ident="")
- RETURN_CODE [msr_setWhiteList](#) (byte[] value, byte[] MAC, string ident="")
- RETURN_CODE [msr_startMSRSwipe](#) (int timeout, string ident="")
- RETURN_CODE [msr_cancelMSRSwipe](#) (string ident="")
- RETURN_CODE [device_getRKIStatus](#) (bool isProd, bool isMultiKey, ref string status, ref Dictionary< string, RKI_KEY_TYPE > keys, string ident="")
- RETURN_CODE [device_getAnyRKIStatus](#) (bool isMultiKey, ref string status, ref Dictionary< string, RKI_KEY_TYPE > keys, string ident="")
- RETURN_CODE [device_sendConfigurationFromZip](#) (byte[] zip, string filename, VC_OPERATION_TYPE type, bool matchFW, string ident="", bool isForeground=false)
- RETURN_CODE [device_sendConfiguration](#) (string filename, VC_OPERATION_TYPE type, bool matchFW, string ident="", bool isForeground=false)
- RETURN_CODE [device_readConfigurationToMemory](#) (string memo, Configs.InstallRules rules, List< Configs.DeviceCommand > cmds, string ident="", bool isForeground=false)
- RETURN_CODE [device_getConfigurationFromMemory](#) (ref string json, string ident="")
- RETURN_CODE [device_RemoteKeyInjection](#) (RKI_KEY_TYPE type, string keyName, string ident="", bool performOnForeground=false)

Static Public Member Functions

- static bool [useSerialPort](#) (int port)
- static bool [useSerialPort](#) (int port, int baud)
- static bool [useSerialPortLinux](#) (string path)
- static bool [useSerialPortLinux](#) (string path, int baud)
- static void [initSC](#) ()
- static int [getCommandTimeout](#) (string ident="")
- static void [setCommandTimeout](#) (int milliseconds, string ident="")
- static void [setCallback](#) (Callback my_Callback)
- static void [setCallback](#) (IntPtr my_Callback, SynchronizationContext context)
- static void [emv_autoAuthenticate](#) (bool authenticate, string ident="")
- static void [emv_autoAuthenticateTags](#) (bool authenticate, byte[] tags, string ident="")
- static void [emv_allowFallback](#) (bool allow, string ident="")
- static String [SDK_Version](#) ()

Properties

- static [IDT_BTPay SharedController](#) [get]

21.3.1 Member Function Documentation

21.3.1.1 **RETURN_CODE** IDTechSDK.IDT_BTPay.config_getBeeperController (ref bool *firmwareControlBeeper*, string *ident* = " ")

Get the Beeper Controller Status Set the Beeper controlled Status by software or firmware

Parameters

<i>firmwareControlBeeper</i>	true means firmware control the beeper, false means software control beeper.
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString

21.3.1.2 **RETURN_CODE** IDTechSDK.IDT_BTPay.config_getEncryptionControl (ref bool *msr*, ref bool *icc*, string *ident* = " ")

Get Encryption Control

Get Encryption Control to switch status between MSR and ICC/EMV function. Following Encryption status supported:

- MSR ON, ICC/EMV ON,
- MSR ON, ICC/EMV OFF,
- MSR OFF, ICC/EMV OFF,

Parameters

<i>msr</i>	<ul style="list-style-type: none"> true: enabled MSR with Encryption, false: disabled MSR with Encryption,
<i>icc</i>	<ul style="list-style-type: none"> true: enabled ICC with Encryption, false: disabled ICC with Encryption,
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString

21.3.1.3 RETURN_CODE IDTechSDK.IDT_BTPay.config_getLEDController (ref bool *firmwareControlMSRLED*, ref bool *firmwareControlICCLED*, string *ident* = " ")

Get the LED Controller Status Get the MSR / ICC LED controlled status by software or firmware

Parameters

<i>firmwareControlMSRLED</i>	<ul style="list-style-type: none"> • true: firmware control the MSR LED • false: software control the MSR LED
<i>firmwareControlICCLED</i>	<ul style="list-style-type: none"> • true: firmware control the ICC LED • false: software control the ICC LED
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString

21.3.1.4 RETURN_CODE IDTechSDK.IDT_BTPay.config_getModelNumber (ref string *response*, string *ident* = " ")

Polls device for Model Number

@param response Returns Model Number

Parameters

<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.
--------------	--

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString

21.3.1.5 RETURN_CODE IDTechSDK.IDT_BTPay.config_getSerialNumber (ref string *response*, string *ident* = " ")

Polls device for Serial Number

@param response Returns Serial Number

Parameters

<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.
--------------	--

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString

21.3.1.6 RETURN_CODE IDTechSDK.IDT_BTPay.config_setBeeperController (bool *firmwareControlBeeper*, string *ident* = " ")

Set the Beeper Controller Set the Beeper controlled by software or firmware

Parameters

<i>firmwareControlBeeper</i>	true means firmware control the beeper, false means software control beeper.
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString

21.3.1.7 bool IDTechSDK.IDT_BTPay.config_setCmdTimeOutDuration (int *newTimeOut*, string *ident* = " ")

Command Acknowledgement Timeout

Sets the amount of seconds to wait for an {ACK} to a command before a timeout. Responses should normally be received under one second. Default is 3 seconds

Parameters

<i>newTimeOut</i>	Timeout value. Valid range 1 - 60 seconds
-------------------	---

Returns

Success flag. Determines if value was set and in range.

21.3.1.8 RETURN_CODE IDTechSDK.IDT_BTPay.config_setEncryptionControl (bool *msr*, bool *icc*, string *ident* = " ")

Set Encryption Control

Set Encryption Control to switch status between MSR and ICC/EMV function. Following Encryption status supported:

- MSR ON, ICC/EMV ON,
- MSR ON, ICC/EMV OFF,
- MSR OFF, ICC/EMV OFF,

Parameters

<i>msr</i>	<ul style="list-style-type: none"> • true: enable MSR with Encryption, • false: disable MSR with Encryption,
------------	--

Parameters

<i>icc</i>	<ul style="list-style-type: none"> • true: enable ICC with Encryption, • false: disable ICC with Encryption,
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString

21.3.1.9 RETURN_CODE IDTechSDK.IDT_BTPay.config_setLEDController (bool *firmwareControlMSRLED*, bool *firmwareControlICCLED* = false, string *ident* = " ")

Set the LED Controller Set the MSR / ICC LED controlled by software or firmware NOTE: The ICC LED always controlled by software.

Parameters

<i>firmwareControlMSRLED</i>	<ul style="list-style-type: none"> • true: firmware control the MSR LED • false: software control the MSR LED
<i>firmwareControlICCLED</i>	<ul style="list-style-type: none"> • true: firmware control the ICC LED • false: software control the ICC LED
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString

21.3.1.10 RETURN_CODE IDTechSDK.IDT_BTPay.device_controlBeep (int *index*, int *frequency*, int *duration*, string *ident* = " ")

Control Beep

Controls the Beeper

Parameters

<i>index</i>	For Augusta, must be set to 1 (only one beeper)
<i>frequency</i>	Frequency, range 1000-20000 (suggest minimum 3000)
<i>duration</i>	Duration, in milliseconds (range 1 - 65525)
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString

21.3.1.11 RETURN_CODE IDTechSDK.IDT_BTPay.device_controlLED (byte *indexLED*, byte *control*, int *intervalOn* = 500, int *intervalOff* = 500, string *ident* = " ")

Control MSR LED

Controls the LED for the MSR

Parameters

<i>indexLED</i>	For Augusta, must be set to 1 (MSR LED)
<i>control</i>	LED Status: <ul style="list-style-type: none">• 00: OFF• 01: RED Solid• 02: RED Blink• 11: GREEN Solid• 12: GREEN Blink• 21: BLUE Solid• 22: BLUE Blink
<i>intervalOn</i>	Blink interval ON, in ms (Range 200 - 2000)
<i>intervalOff</i>	Blink interval OFF, in ms (Range 200 - 2000)
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString

21.3.1.12 RETURN_CODE IDTechSDK.IDT_BTPay.device_controlLED_ICC (int *controlMode*, int *interval*, string *ident* = " ")

Control ICC LED

Controls the LED for the ICC card slot

Parameters

<i>controlMode</i>	0 = off, 1 = solid, 2 = blink
<i>interval</i>	Blink interval, in ms (500 = 500 ms)
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString

21.3.1.13 RETURN_CODE IDTechSDK.IDT_BTPay.device_controlLED_MSR (byte control, int intervalOn = 500, int intervalOff = 500, string ident = " ")

Control the MSR LED

Controls the MSR / ICC LED This API not recommended to control ICC LED

Parameters

<i>control</i>	<ul style="list-style-type: none"> • 0x00 = off, • 0x01 = RED Solid, • 0x02 = RED Blink, • 0x11 = GREEN Solid, • 0x12 = GREEN Blink, • 0x21 = BLUE Solid, • 0x22 = BLUE Blink,
<i>intervalOn</i>	Blink interval on time last, in ms (500 = 500 ms, valid from 200 to 2000)
<i>intervalOff</i>	Blink interval off time last, in ms (500 = 500 ms, valid from 200 to 2000)
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString

21.3.1.14 RETURN_CODE IDTechSDK.IDT_BTPay.device_getAnyRKIStatus (bool isMultiKey, ref string status, ref Dictionary< string, RKI_KEY_TYPE > keys, string ident = " ")

Get RKI Status Polls the RKI servers to validate device RKI eligibility. Note: if device type is known in advance (production or demo device), it is more efficient to use device_getRKIStatus instead

Parameters

<i>isMultiKey</i>	True = MultiKey, False = SingleKey
<i>status</i>	RKI status returned as a string
<i>keys</i>	Returns all available keys
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with errorCode.getErrorString(). When no data is available, return code = RETURN_CODE_NO_DATA_AVAILABLE

21.3.1.15 RETURN_CODE IDTechSDK.IDT_BTPay.device_getConfigurationFromMemory (ref string json, string ident = " ")

Read Device Configuration From Memory Retrieves the ViVOconfig configuration data from memory after a successful device_readConfigurationToMemory

Parameters

<i>json</i>	Returns the .json file data that can be saved to disk
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with errorCode.getErrorString(). When no data is available, return code = RETURN_CODE_NO_DATA_AVAILABLE

21.3.1.16 RETURN_CODE IDTechSDK.IDT_BTPay.device_getFirmwareVersion (ref string response, string ident = " ")

Polls device for Firmware Version

Parameters

<i>response</i>	Response returned of Firmware Version
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString

21.3.1.17 RETURN_CODE IDTechSDK.IDT_BTPay.device_getKeyStatus (ref byte[] status, string ident = " ")

Get Key Status

Gets the status of loaded keys

Parameters

<i>status</i>	when the newFormat is false, data format as follows. byte 0: PIN DUKPT Key, Does not support, always 0 byte 1: PIN Master Key, Does not support, always 0 byte 2: PIN Session Key, Does not support, always 0 byte 3: Account/MSR DUKPT Key, 1 Exists, 0 None, 0xFF STOP byte 4: Account/ICC DUKPT Key, 1 Exists, 0 None, 0xFF STOP byte 5: Admin DUKPT Key, 1 Exists, 0 None, 0xFF STOP
---------------	--

when the newFormat is true, data format as follows. [Block Length] [KeyStatusBlock1] [KeyStatusBlock2] ... [KeyStatusBlockN] Where: [Block Length] is 2 bytes, format is Len_L Len_H, is KeyStatusBlock Number [KeyStatusBlockX] is 4 bytes, format is [Key Index and Key Name] [key slot] [key status]: [Key Index and Key Name] is 1 byte. Please refer to following table 0x14 LCL-KEK to Encrypt Other Keys 0x02 Data encryption Key to Encrypt ICC/MSR 0x05 MAC DUKPT Key for Host-Device – MAC Verification 0x05 MTK DUKPT Key for TTK Self-Test 0x0C RKI-KEK for Remote Key Injection [key slot] is 2 bytes. Range is 0 – 9999 the MTK DUKPT Key slot is 16, the others are all 0 [key status] is 1 byte. 0 – Not Exist 1 – Exist 0xFF – (Stop. Only Valid for DUKPT Key)

Parameters

<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.
--------------	--

Returns

RETURN_CODE: Values can be parsed with `errorCode.getErrorString()`

21.3.1.18 `string IDTechSDK.IDT_BTPay.device_getResponseCodeString (RETURN_CODE eCode)`

Get the description of response result.

Parameters

<i>eCode</i>	the response result.
--------------	----------------------

Return values

<i>the</i>	string for description of response result
------------	---

- case 0: "no error, beginning task";
- case 1: "no response from reader";
- case 2: "invalid response data";
- case 3: "time out for task or CMD";
- case 4: "wrong parameter";
- case 5: "SDK is doing MSR or ICC task";
- case 6: "SDK is doing PINPad task";
- case 7: "SDK is doing CTLS task";
- case 8: "SDK is doing EMV task";
- case 9: "SDK is doing Other task";
- case 10: "err response or data";
- case 11: "no reader attached";
- case 12: "mono audio is enabled";
- case 13: "did connection";
- case 14: "audio volume is too low";
- case 15: "task or CMD be canceled";
- case 16: "UF wrong string format";
- case 17: "UF file not found";
- case 18: "UF wrong file format";
- case 19: "Attempt to contact online host failed";
- case 20: "Attempt to perform RKI failed";
- case 0x300: "Key Type(TDES) of Session Key is not same as the related Master Key.";
- case 0x400: "Related Key was not loaded.";
- case 0x500: "Key Same.";

- case 0x501: "Key is all zero";
- case 0x502: "TR-31 format error";
- case 0x702: "PAN is Error Key.";
- case 0x705: "No Internal MSR PAN (or Internal MSR PAN is erased timeout)";
- case 0X0C01: "Incorrect Frame Tag";
- case 0X0C02: "Incorrect Frame Type";
- case 0X0C03: "Unknown Frame Type";
- case 0X0C04: "Unknown Command";
- case 0X0C05: "Unknown Sub-Command";
- case 0X0C06: "CRC Error";
- case 0X0C07: "Failed";
- case 0X0C08: "Timeout";
- case 0X0C0A: "Incorrect Parameter";
- case 0X0C0B: "Command Not Supported";
- case 0X0C0C: "Sub-Command Not Supported";
- case 0X0C0D: "Parameter Not Supported / Status Abort Command";
- case 0X0C0F: "Sub-Command Not Allowed";
- case 0X0D01: "Incorrect Header Tag";
- case 0X0D02: "Unknown Command";
- case 0X0D03: "Unknown Sub-Command";
- case 0X0D04: "CRC Error in Frame";
- case 0X0D05: "Incorrect Parameter";
- case 0X0D06: "Parameter Not Supported";
- case 0X0D07: "Mal-formatted Data";
- case 0X0D08: "Timeout";
- case 0X0D0A: "Failed / NACK";
- case 0X0D0B: "Command not Allowed";
- case 0X0D0C: "Sub-Command not Allowed";
- case 0X0D0D: "Buffer Overflow (Data Length too large for reader buffer)";
- case 0X0D0E: "User Interface Event";
- case 0X0D11: "Communication type not supported, VT-1, burst, etc.";
- case 0X0D12: "Secure interface is not functional or is in an intermediate state.";
- case 0X0D13: "Data field is not mod 8";
- case 0X0D14: "Pad 0x80 not found where expected";
- case 0X0D15: "Specified key type is invalid";
- case 0X0D1: "Could not retrieve key from the SAM(InitSecureComm)";

- case 0X0D17: "Hash code problem";
- case 0X0D18: "Could not store the key into the SAM(InstallKey)";
- case 0X0D19: "Frame is too large";
- case 0X0D1A: "Unit powered up in authentication state but POS must resend the InitSecureComm command";
- case 0X0D1B: "The EEPROM may not be initialized because SecCommInterface does not make sense";
- case 0X0D1C: "Problem encoding APDU";
- case 0X0D20: "Unsupported Index(ILM) SAM Transceiver error – problem communicating with the SAM(Key Mgr)";
- case 0X0D2: "Unexpected Sequence Counter in multiple frames for single bitmap(ILM) Length error in data returned from the SAM(Key Mgr)";
- case 0X0D22: "Improper bit map(ILM)";
- case 0X0D23: "Request Online Authorization";
- case 0X0D24: "ViVOCard3 raw data read successful";
- case 0X0D25: "Message index not available(ILM) ViVOcomm activate transaction card type(ViVOcomm)";
- case 0X0D26: "Version Information Mismatch(ILM)";
- case 0X0D27: "Not sending commands in correct index message index(ILM)";
- case 0X0D28: "Time out or next expected message not received(ILM)";
- case 0X0D29: "ILM languages not available for viewing(ILM)";
- case 0X0D2A: "Other language not supported(ILM)";
- case 0X0D41: "Unknown Error from SAM";
- case 0X0D42: "Invalid data detected by SAM";
- case 0X0D43: "Incomplete data detected by SAM";
- case 0X0D44: "Reserved";
- case 0X0D45: "Invalid key hash algorithm";
- case 0X0D46: "Invalid key encryption algorithm";
- case 0X0D47: "Invalid modulus length";
- case 0X0D48: "Invalid exponent";
- case 0X0D49: "Key already exists";
- case 0X0D4A: "No space for new RID";
- case 0X0D4B: "Key not found";
- case 0X0D4C: "Crypto not responding";
- case 0X0D4D: "Crypto communication error";
- case 0X0D4E: "Module-specific error for Key Manager";
- case 0X0D4F: "All key slots are full (maximum number of keys has been installed)";
- case 0X0D50: "Auto-Switch OK";
- case 0X0D51: "Auto-Switch failed";

- case 0X0D90: "Account DUKPT Key not exist";
- case 0X0D91: "Account DUKPT Key KSN exhausted";
- case 0x0D00: "This Key had been loaded.";
- case 0x0E00: "Base Time was loaded.";
- case 0x0F00: "Encryption Or Decryption Failed.";
- case 0x1000: "Battery Low Warning (It is High Priority Response while Battery is Low.)";
- case 0x1800: "Send "Cancel Command" after send "Get Encrypted PIN" & "Get Numeric" & "Get Amount";
- case 0x1900: "Press "Cancel" key after send "Get Encrypted PIN" & "Get Numeric" & "Get Amount";
- case 0x30FF: "Security Chip is not connect";
- case 0x3000: "Security Chip is deactivation & Device is In Removal Legally State.";
- case 0x3101: "Security Chip is activation & Device is In Removal Legally State.";
- case 0x5500: "No Admin DUKPT Key.";
- case 0x5501: "Admin DUKPT Key STOP.";
- case 0x5502: "Admin DUKPT Key KSN is Error.";
- case 0x5503: "Get Authentication Code1 Failed.";
- case 0x5504: "Validate Authentication Code Error.";
- case 0x5505: "Encrypt or Decrypt data failed.";
- case 0x5506: "Not Support the New Key Type.";
- case 0x5507: "New Key Index is Error.";
- case 0x5508: "Step Error.";
- case 0x5509: "KSN Error";
- case 0x550A: "MAC Error.";
- case 0x550B: "Key Usage Error.";
- case 0x550C: "Mode Of Use Error.";
- case 0x550F: "Other Error.";
- case 0x6000: "Save or Config Failed / Or Read Config Error.";
- case 0x6200: "No Serial Number.";
- case 0x6900: "Invalid Command - Protocol is right, but task ID is invalid.";
- case 0x6A01: "Unsupported Command – Protocol and task ID are right, but command is invalid – In this State";
- case 0x6A00: "Unsupported Command - Protocol and task ID are right, but command is invalid.";
- case 0x6B00: "Unknown parameter in command - Protocol task ID and command are right, but parameter is invalid.";
- case 0x6C00: "Unknown parameter in command – Protocol task ID and command are right, but length is out of the requirement.";
- case 0x7200: "Device is suspend (MKSK suspend or press password suspend).";
- case 0x7300: "PIN DUKPT is STOP (21 bit 1).";

- case 0x7400: "Device is Busy.";
- case 0xE100: "Can not enter sleep mode";
- case 0xE200: "File has existed";
- case 0xE300: "File has not existed";
- case 0xE313: "IO line low -- Card error after session start";
- case 0xE400: "Open File Error";
- case 0xE500: "SmartCard Error";
- case 0xE600: "Get MSR Card data is error";
- case 0xE700: "Command time out";
- case 0xE800: "File read or write is error";
- case 0xE900: "Active 1850 error!";
- case 0xEA00: "Load bootloader error";
- case 0xEF00: "Protocol Error- STX or ETX or check error.";
- case 0xEB00: "Picture is not exist";
- case 0x2C02: "No Microprocessor ICC seated";
- case 0x2C06: "no card seated to request ATR";
- case 0x2D01: "Card Not Supported,";
- case 0x2D03: "Card Not Supported, wants CRC";
- case 0x690D: "Command not supported on reader without ICC support";
- case 0x8100: "ICC error time out on power-up";
- case 0x8200: "invalid TS character received - Wrong operation step";
- case 0x8300: "Decode MSR Error";
- case 0x8400: "TriMagII no Response";
- case 0x8500: "No Swipe MSR Card";
- case 0x8510: "No Financial Card";
- case 0x8600: "Unsupported F, D, or combination of F and D";
- case 0x8700: "protocol not supported EMV TD1 out of range";
- case 0x8800: "power not at proper level";
- case 0x8900: "ATR length too long";
- case 0x8B01: "EMV invalid TA1 byte value";
- case 0x8B02: "EMV TB1 required";
- case 0x8B03: "EMV Unsupported TB1 only 00 allowed";
- case 0x8B04: "EMV Card Error, invalid BWI or CWI";
- case 0x8B06: "EMV TB2 not allowed in ATR";
- case 0x8B07: "EMV TC2 out of range";
- case 0x8B08: "EMV TC2 out of range";

- case 0x8B09: "per EMV96 TA3 must be > 0xF";
- case 0x8B10: "ICC error on power-up";
- case 0x8B11: "EMV T=1 then TB3 required";
- case 0x8B12: "Card Error, invalid BWI or CWI";
- case 0x8B13: "Card Error, invalid BWI or CWI";
- case 0x8B17: "EMV TC1/TB3 conflict-";
- case 0x8B20: "EMV TD2 out of range must be T=1";
- case 0x8C00: "TCK error";
- case 0xA304: "connector has no voltage setting";
- case 0xA305: "ICC error on power-up invalid (SBLK(IFSD) exchange";
- case 0xE301: "ICC error after session start";
- case 0xFF00: "Request to go online";
- case 0xFF01: "EMV: Accept the offline transaction";
- case 0xFF02: "EMV: Decline the offline transaction";
- case 0xFF03: "EMV: Accept the online transaction";
- case 0xFF04: "EMV: Decline the online transaction";
- case 0xFF05: "EMV: Application may fallback to magstripe technology";
- case 0xFF06: "EMV: ICC detected tah the conditions of use are not satisfied";
- case 0xFF07: "EMV: ICC didn't accept transaction";
- case 0xFF08: "EMV: Transaction was cancelled";
- case 0xFF09: "EMV: Application was not selected by kernel or ICC format error or ICC missing data error";
- case 0xFF0A: "EMV: Transaction is terminated";
- case 0xFF0B: "EMV: Other EMV Error";
- case 0xFFFF: "NO RESPONSE";
- case 0xF002: "ICC communication timeout";
- case 0xF003: "ICC communication Error";
- case 0xF00F: "ICC Card Seated and Highest Priority, disable MSR work request";
- case 0xF200: "AID List / Application Data is not exist";
- case 0xF201: "Terminal Data is not exist";
- case 0xF202: "TLV format is error";
- case 0xF203: "AID List is full";
- case 0xF204: "Any CA Key is not exist";
- case 0xF205: "CA Key RID is not exist";
- case 0xF206: "CA Key Index it not exist";
- case 0xF207: "CA Key is full";
- case 0xF208: "CA Key Hash Value is Error";

- case 0xF209: "Transaction format error";
- case 0xF20A: "The command will not be processing";
- case 0xF20B: "CRL is not exist";
- case 0xF20C: "CRL number exceed max number";
- case 0xF20D: "Amount,Other Amount,Trasaction Type are missing";
- case 0xF20E: "The Identification of algorithm is mistake";
- case 0xF20F: "No Financial Card";
- case 0xF210: "In Encrypt Result state, TLV total Length is greater than Max Length";
- case 0x1001: "INVALID ARG";
- case 0x1002: "FILE_OPEN_FAILED";
- case 0x1003: "FILE OPERATION_FAILED";
- case 0x2001: "MEMORY_NOT_ENOUGH";
- case 0x3002: "SMARTCARD_FAIL";
- case 0x3003: "SMARTCARD_INIT_FAILED";
- case 0x3004: "FALLBACK_SITUATION";
- case 0x3005: "SMARTCARD_ABSENT";
- case 0x3006: "SMARTCARD_TIMEOUT";
- case 0x5001: "EMV_PARSING_TAGS_FAILED";
- case 0x5002: "EMV_DUPLICATE_CARD_DATA_ELEMENT";
- case 0x5003: "EMV_DATA_FORMAT_INCORRECT";
- case 0x5004: "EMV_NO_TERM_APP";
- case 0x5005: "EMV_NO_MATCHING_APP";
- case 0x5006: "EMV_MISSING_MANDATORY_OBJECT";
- case 0x5007: "EMV_APP_SELECTION_RETRY";
- case 0x5008: "EMV_GET_AMOUNT_ERROR";
- case 0x5009: "EMV_CARD_REJECTED";
- case 0x5010: "EMV_AIP_NOT_RECEIVED";
- case 0x5011: "EMV_AFL_NOT_RECEIVED";
- case 0x5012: "EMV_AFL_LEN_OUT_OF_RANGE";
- case 0x5013: "EMV_SFI_OUT_OF_RANGE";
- case 0x5014: "EMV_AFL_INCORRECT";
- case 0x5015: "EMV_EXP_DATE_INCORRECT";
- case 0x5016: "EMV_EFF_DATE_INCORRECT";
- case 0x5017: "EMV_ISS_COD_TBL_OUT_OF_RANGE";
- case 0x5018: "EMV_CRYPTOGAM_TYPE_INCORRECT";
- case 0x5019: "EMV_PSE_NOT_SUPPORTED_BY_CARD";

- case 0x5020: "EMV_USER_SELECTED_LANGUAGE";
- case 0x5021: "EMV_SERVICE_NOT_ALLOWED";
- case 0x5022: "EMV_NO_TAG_FOUND";
- case 0x5023: "EMV_CARD_BLOCKED";
- case 0x5024: "EMV_LEN_INCORRECT";
- case 0x5025: "CARD_COM_ERROR";
- case 0x5026: "EMV_TSC_NOT_INCREASED";
- case 0x5027: "EMV_HASH_INCORRECT";
- case 0x5028: "EMV_NO_ARC";
- case 0x5029: "EMV_INVALID_ARC";
- case 0x5030: "EMV_NO_ONLINE_COMM";
- case 0x5031: "TRAN_TYPE_INCORRECT";
- case 0x5032: "EMV_APP_NO_SUPPORT";
- case 0x5033: "EMV_APP_NOT_SELECT";
- case 0x5034: "EMV_LANG_NOT_SELECT";
- case 0x5035: "EMV_NO_TERM_DATA";
- case 0x6001: "CVM_TYPE_UNKNOWN";
- case 0x6002: "CVM_AIP_NOT_SUPPORTED";
- case 0x6003: "CVM_TAG_8E_MISSING";
- case 0x6004: "CVM_TAG_8E_FORMAT_ERROR";
- case 0x6005: "CVM_CODE_IS_NOT_SUPPORTED";
- case 0x6006: "CVM_COND_CODE_IS_NOT_SUPPORTED";
- case 0x6007: "NO_MORE_CVM";
- case 0x6008: "PIN_BYPASSED_BEFORE";
- case 0x7001: "PK_BUFFER_SIZE_TOO_BIG";
- case 0x7002: "PK_FILE_WRITE_ERROR";
- case 0x7003: "PK_HASH_ERROR";
- case 0x8001: "NO_CARD_HOLDER_CONFIRMATION";
- case 0x8002: "GET_ONLINE_PIN";
- case 0xD000: "Data not exist";
- case 0xD001: "Data access error";
- case 0xD100: "RID not exist";
- case 0xD101: "RID existed";
- case 0xD102: "Index not exist";
- case 0xD200: "Maximum exceeded";
- case 0xD201: "Hash error";

- case 0xD205: "System Busy";
- case 0x0E01: "Unable to go online";
- case 0x0E02: "Technical Issue";
- case 0x0E03: "Declined";
- case 0x0E04: "Issuer Referral transaction";
- case 0x0F01: "Decline the online transaction";
- case 0x0F02: "Request to go online";
- case 0x0F03: "Transaction is terminated";
- case 0x0F05: "Application was not selected by kernel or ICC format error or ICC missing data error";
- case 0x0F07: "ICC didn't accept transaction";
- case 0x0F0A: "Application may fallback to magstripe technology";
- case 0x0F0C: "Transaction was cancelled";
- case 0x0F0D: "Timeout";
- case 0x0F0F: "Other EMV Error";
- case 0x0F10: "Accept the offline transaction";
- case 0x0F11: "Decline the offline transaction";
- case 0x0F21: "ICC detected tah the conditions of use are not satisfied";
- case 0x0F22: "No app were found on card matching terminal configuration";
- case 0x0F23: "Terminal file does not exist";
- case 0x0F24: "CAPK file does not exist";
- case 0x0F25: "CRL Entry does not exist";
- case 0x0FFE: "code when blocking is disabled";
- case 0x0FFF: "code when command is not applicable on the selected device";
- case 0xF005: "ICC Encrypted C-APDU Data Structure Length Error Or Format Error.";
- case 0xBBE0: "CM100 Success";
- case 0xBBE1: "CM100 Parameter Error";
- case 0xBBE2: "CM100 Low Output Buffer";
- case 0xBBE3: "CM100 Card Not Found";
- case 0xBBE4: "CM100 Collision Card Exists";
- case 0xBBE5: "CM100 Too Many Cards Exist";
- case 0xBBE6: "CM100 Saved Data Does Not Exist";
- case 0xBBE8: "CM100 No Data Available";
- case 0xBBE9: "CM100 Invalid CID Returned";
- case 0xBBEA: "CM100 Invalid Card Exists";
- case 0xBBEC: "CM100 Command Unsupported";
- case 0xBBED: "CM100 Error In Command Process";

- case 0xBBEE: "CM100 Invalid Command";
- case 0X9031: "Unknown command";
- case 0X9032: "Wrong parameter (such as the length of the command is incorrect)";
- case 0X9038: "Wait (the command couldn't be finished in BWT)";
- case 0X9039: "Busy (a previously command has not been finished)";
- case 0X903A: "Number of retries over limit";
- case 0X9040: "Invalid Manufacturing system data";
- case 0X9041: "Not authenticated";
- case 0X9042: "Invalid Master DUKPT Key";
- case 0X9043: "Invalid MAC Key";
- case 0X9044: "Reserved for future use";
- case 0X9045: "Reserved for future use";
- case 0X9046: "Invalid DATA DUKPT Key";
- case 0X9047: "Invalid PIN Pairing DUKPT Key";
- case 0X9048: "Invalid DATA Pairing DUKPT Key";
- case 0X9049: "No nonce generated";
- case 0X9949: "No GUID available. Perform getVersion first.";
- case 0X9950: "MAC Calculation unsuccessful. Check BDK value.";
- case 0X904A: "Not ready";
- case 0X904B: "Not MAC data";
- case 0X9050: "Invalid Certificate";
- case 0X9051: "Duplicate key detected";
- case 0X9052: "AT checks failed";
- case 0X9053: "TR34 checks failed";
- case 0X9054: "TR31 checks failed";
- case 0X9055: "MAC checks failed";
- case 0X9056: "Firmware download failed";
- case 0X9060: "Log is full";
- case 0X9061: "Removal sensor unengaged";
- case 0X9062: "Any hardware problems";
- case 0X9070: "ICC communication timeout";
- case 0X9071: "ICC data error (such check sum error)";
- case 0X9072: "Smart Card not powered up";

21.3.1.19 RETURN_CODE IDTechSDK.IDT_BTPay.device_getRKIStatus (bool *isProd*, bool *isMultiKey*, ref string *status*, ref Dictionary< string, RKI_KEY_TYPE > *keys*, string *ident* = " ")

Get RKI Status From Specified Server Polls the RKI server to validate device RKI eligibility

Parameters

<i>isProd</i>	TRUE = poll production RKI server, FALSE = poll demo RKI server
<i>isMultiKey</i>	True = MultiKey, False = SingleKey
<i>status</i>	RKI status returned as a string
<i>keys</i>	Returns dictionary of available keys
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with `errorCode.getErrorString()`. When no data is available, return code = RETURN_CODE_NO_DATA_AVAILABLE

21.3.1.20 RETURN_CODE IDTechSDK.IDT_BTPay.device_readConfigurationToMemory (string *memo*, Configs.InstallRules *rules*, List< Configs.DeviceCommand > *cmds*, string *ident* = " ", bool *isForeground* = false)

Read Device Configuration To Memory Executes a ViVOconfig read of device settings and stores the results in memory

Once a ViVOconfig read starts, all messages will be returned to the callback as DeviceState.ViVOconfig. The ViVOconfig operation is complete when the return code from a DeviceState.ViVOconfig message is one of the following values:

- RETURN_CODE_SDK_VIVOCONFIG_READ_CONFIG_SUCCESS = Read process completed successfully. Execute `device_getConfigurationFromMemory` to retrieve the configuration data
- RETURN_CODE_SDK_VIVOCONFIG_READ_CONFIG_FAILED = Read process FAILED.

Parameters

<i>memo</i>	Optional: populates the memo field of the captured .json file
<i>rules</i>	Optional: populates the .json file with available install rules
<i>cmds</i>	Optional: populates the .json file with commands to execute
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.
<i>isForeground</i>	If TRUE, will block program until update complete. Otherwise, FALSE performs update on background.

Returns

RETURN_CODE: Values can be parsed with `errorCode.getErrorString()`. When no data is available, return code = RETURN_CODE_NO_DATA_AVAILABLE

21.3.1.21 RETURN_CODE IDTechSDK.IDT_BTPay.device_rebootDevice (string *ident* = " ")

Reboot Device

Executes a command to restart the device.

Parameters

<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.
--------------	--

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString

21.3.1.22 RETURN_CODE IDTechSDK.IDT_BTPay.device_RemoteKeyInjection (RKI_KEY_TYPE type, string keyName, string ident = "", bool performOnForeground = false)

Remote Key Injection Performs a remote key injection for the device

Parameters

type	Remote Key Injection Type
keyName	Name of key (optional)
ident	Device ID to send command to. If not specified, current SDK default device will be used.
performOnForeground	TRUE = block during process (default is false)

Returns

RETURN_CODE: Values can be parsed with errorCode.getErrorString(). When no data is available, return code = RETURN_CODE_NO_DATA_AVAILABLE

21.3.1.23 RETURN_CODE IDTechSDK.IDT_BTPay.device_sendConfiguration (string filename, VC_OPERATION_TYPE type, bool matchFW, string ident = "", bool isForeground = false)

Send Configuration Executes a ViVOconfig update or verify to a device

Once a ViVOconfig write or verify starts, all messages will be returned to the callback as DeviceState.ViVOconfig. The ViVOconfig operation is complete when the return code from a DeviceState.ViVOconfig message is one of the following values:

- RETURN_CODE_SDK_VIVOCONFIG_VERIFY_SUCCESS = Verification process completed successfully. No differences found.
- RETURN_CODE_SDK_VIVOCONFIG_VERIFY_SUCCESS_WITH_WARNING = Verification process completed with warnings.
- RETURN_CODE_SDK_VIVOCONFIG_VERIFY_FAILED = Verification process FAILED
- RETURN_CODE_SDK_VIVOCONFIG_WRITE_SUCCESS = Write process completed successfully.
- RETURN_CODE_SDK_VIVOCONFIG_WRITE_SUCCESS_WITH_WARNING = Write process completed with warnings
- RETURN_CODE_SDK_VIVOCONFIG_WRITE_FAILED = Write process FAILED

Parameters

filename	The .json configuration file to update the device with NOTE: You can also use this parameter to pass the path of where the .json file is located at on local storage instead
filename	The name of the .json configuration file
type	VC_OPERATION_TYPE

- VC_OPERATION_TYPE_WRITE = Write File To Device, Hash must validate
- VC_OPERATION_TYPE_VERIFY = Verify Device With File

- VC_OPERATION_TYPE_WRITE_IGNORE_HASH = Write File To Device, Ignore Hash
- VC_OPERATION_TYPE_WRITE_FIX_HASH = Write File To Device, Fix Hash If Necessary

Parameters

<i>matchFW</i>	TRUE = Device FW must match file FW, FALSE = Don't validate device FW against file
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.
<i>isForeground</i>	If TRUE, will block program until update complete. Otherwise, FALSE performs update on background.

Returns

RETURN_CODE: Values can be parsed with `errorCode.getErrorString()`. When no data is available, return code = RETURN_CODE_NO_DATA_AVAILABLE

21.3.1.24 RETURN_CODE IDTechSDK.IDT_BTPay.device_sendConfigurationFromZip (byte[] zip, string filename, VC_OPERATION_TYPE type, bool matchFW, string ident = "", bool isForeground = false)

Send Configuration From Zip Executes a ViVOconfig update or verify to a device using files from a .zip archive

Once a ViVOconfig write or update starts, all messages will be returned to the callback as DeviceState.ViVOconfig. The ViVOconfig operation is complete when the return code from a DeviceState.ViVOconfig message is one of the following values:

- RETURN_CODE_SDK_VIVOCONFIG_VERIFY_SUCCESS = Verification process completed successfully. No differences found.
- RETURN_CODE_SDK_VIVOCONFIG_VERIFY_SUCCESS_WITH_WARNING = Verification process completed with warnings.
- RETURN_CODE_SDK_VIVOCONFIG_VERIFY_FAILED = Verification process FAILED
- RETURN_CODE_SDK_VIVOCONFIG_WRITE_SUCCESS = Write process completed successfully.
- RETURN_CODE_SDK_VIVOCONFIG_WRITE_SUCCESS_WITH_WARNING = Write process completed with warnings
- RETURN_CODE_SDK_VIVOCONFIG_WRITE_FAILED = Write process FAILED

Parameters

<i>zip</i>	.Zip archive containing the .json configuration file and also any associated image assets NOTE: You can also use this parameter to pass the path of where the .zip file is located at on local storage instead
<i>filename</i>	The name of the .json configuration file
<i>type</i>	VC_OPERATION_TYPE

- VC_OPERATION_TYPE_WRITE = Write File To Device, Hash must validate
- VC_OPERATION_TYPE_VERIFY = Verify Device With File
- VC_OPERATION_TYPE_WRITE_IGNORE_HASH = Write File To Device, Ignore Hash
- VC_OPERATION_TYPE_WRITE_FIX_HASH = Write File To Device, Fix Hash If Necessary

Parameters

<i>matchFW</i>	TRUE = Device FW must match file FW, FALSE = Don't validate device FW against file
----------------	--

Parameters

<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.
<i>isForeground</i>	If TRUE, will block program until update complete. Otherwise, FALSE performs update on background.

Returns

RETURN_CODE: Values can be parsed with `errorCode.getErrorString()`. When no data is available, return code = RETURN_CODE_NO_DATA_AVAILABLE

21.3.1.25 RETURN_CODE IDTechSDK.IDT_BTPay.device_sendDataCommand (string *cmd*, bool *calcLRC*, ref byte[] *response*, string *ident* = " ")

Send a NSData object to device

Sends a command represented by the provide NSData object to the device through the accessory protocol.

Parameters

<i>cmd</i>	NSData representation of command to execute
<i>calcLRC</i>	If TRUE, this will wrap command with start/length/lrc/sum/end: '{STX}{Len_Low}{Len_High} data {CheckLRC} {CheckSUM} {ETX}'
<i>response</i>	Response data
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with `device_getResponseCodeString`

21.3.1.26 RETURN_CODE IDTechSDK.IDT_BTPay.device_sendPAE (string *command*, ref string *response*, int *timeoutstring*, string *ident* = " ")

Send Payment Application Engine Command

Executes a PAE command

Parameters

<i>command</i>	ASCII command string, should start with "*PAE"
<i>response</i>	command response
<i>timeout</i>	timeout waiting for PAE response
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with `errorCode.getErrorString()`

21.3.1.27 RETURN_CODE IDTechSDK.IDT_BTPay.device_setDateTime (string *ident* = " ")

Set Date Time

Set current system date and time to the device.

Parameters

<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.
--------------	--

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString

21.3.1.28 RETURN_CODE IDTechSDK.IDT_BTPay.device_startRKI (string *ident* = " ")

Start Remote Key Injection

Starts a remote key injection request with IDTech RKI servers.

Parameters

<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.
--------------	--

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString

21.3.1.29 RETURN_CODE IDTechSDK.IDT_BTPay.emv_addTerminalData (byte[] *tlv*, string *ident* = " ")

Add Terminal Data

Adds to the exosting Terminal Data

Parameters

<i>tlv</i>	TerminalData TLV Data
<i>ident</i>	Optional identifier

Return values

<i>RETURN_CODE</i>	Return codes listed as typedef enum in IDTCommon:RETURN_CODE. Values can be parsed with IDT_BTPay::device_getResponseCodeString:()
--------------------	--

21.3.1.30 static void IDTechSDK.IDT_BTPay.emv_allowFallback (bool *allow*, string *ident* = " ") [static]

Allow fallback for EMV transactions. Default is TRUE

Parameters

<i>allow</i>	TRUE = allow fallback, FALSE = don't allow fallback
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

21.3.1.31 RETURN_CODE IDTechSDK.IDT_BTPay.emv_authenticateTransaction (byte[] *updatedTLV*, string *ident* = " ")

Authenticate EMV Transaction Request

Authenticates the EMV transaction for an ICC card. Execute this after receiving response with result code 0x10 to `emv_startTransaction`

The tags will be returned in the callback routine.

@param *updatedTLV* TLV stream that can be used to update the following values:

- 9F02: Amount
- 9F03: Other amount
- 9C: Transaction type
- 5F57: Account type

In addition tag DFEE1A can be sent to specify tag list to include in results. Example four tags 9F02, 9F36, 95 to be included in response = DFEE1A079F029F369f9F37

@param *ident* Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with `device_getResponseCodeString`

21.3.1.32 static void IDTechSDK.IDT_BTPay.emv_autoAuthenticate (bool *authenticate*, string *ident* = " ") [static]

Enables authenticate for EMV transactions. If a `startEMVTransaction` results in code 0x0010 (start transaction success), then `emv_authenticateEMVTransaction` can automatically execute if parameter is set to TRUE

Parameters

<i>authenticate</i>	TRUE = auto authenticate, FALSE = manually authenticate
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

21.3.1.33 static void IDTechSDK.IDT_BTPay.emv_autoAuthenticateTags (bool *authenticate*, byte[] *tags*, string *ident* = " ") [static]

Enables authenticate for EMV transactions. If a `startEMVTransaction` results in code 0x0010 (start transaction success), then `emv_authenticateEMVTransaction` can automatically execute if parameter is set to TRUE

Parameters

<i>authenticate</i>	TRUE = auto authenticate, FALSE = manually authenticate
<i>tags</i>	Tags to pass during authentication stage;
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

21.3.1.34 RETURN_CODE IDTechSDK.IDT_BTPay.emv_callbackResponseLCD (EMV_LCD_DISPLAY_MODE *type*, byte *selection*, string *ident* = " ")

Callback Response LCD Display

Provides menu selection responses to the kernel after a callback was received with `DeviceState.EMVCallback`, and `callbackType = EMV_CALLBACK_TYPE.EMV_CALLBACK_TYPE_LCD`, and `lcd_displayMode = EMV_LCD_DISPLAY_MODE_MENU`, `EMV_LCD_DISPLAY_MODE_PROMPT`, or `EMV_LCD_DISPLAY_MODE_LANGUAGE_SELECT`

Parameters

<i>type</i>	If Cancel key pressed during menu selection, then value is EMV_LCD_DISPLAY_MODE_CANCEL. Otherwise, value can be EMV_LCD_DISPLAY_MODE_MENU, EMV_LCD_DISPLAY_MODE_PROMPT, or EMV_LCD_DISPLAY_MODE_LANGUAGE_SELECT
<i>selection</i>	If type = EMV_LCD_DISPLAY_MODE_MENU or EMV_LCD_DISPLAY_MODE_LANGUAGE_SELECT, provide the selection ID line number. Otherwise, if type = EMV_LCD_DISPLAY_MODE_PROMPT supply either 0x43 ('C') for Cancel, or 0x45 ('E') for Enter/accept
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString

21.3.1.35 RETURN_CODE IDTechSDK.IDT_BTPay.emv_callbackResponseMSR (byte[] MSR, string ident = " ")

Callback Response MSR Entry

Provides MSR information to kernel after a callback was received with DeviceState.EMVCallback, and callbackType = EMV_CALLBACK_MSR

Parameters

<i>MSR</i>	Swiped track data
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString

21.3.1.36 RETURN_CODE IDTechSDK.IDT_BTPay.emv_callbackResponsePIN (EMV_PIN_MODE type, byte[] KSN, byte[] PIN, string ident = " ")

Callback Response PIN Entry

Provides (or cancels) PIN entry information to kernel after a callback was received with DeviceState.EMVCallback, and callbackType = EMV_CALLBACK_TYPE_PINPAD

Parameters

<i>type</i>	If Cancel key pressed during PIN entry, then value is EMV_PIN_MODE_CANCEL. Otherwise, value can be EMV_PIN_MODE_ONLINE_DUKPT, EMV_PIN_MODE_ONLINE_MKSK, or EMV_PIN_MODE_OFFLINE
<i>KSN</i>	If enciphered PIN, this is either the PIN DUKPT Key (EMV_PIN_MODE_ONLINE_DUKPT) or PIN Session Key (EMV_PIN_MODE_ONLINE_MKSK), or PIN Pairing DUKPT key (EMV_PIN_MODE_OFFLINE)
<i>PIN</i>	If enciphered PIN, this is encrypted PIN block. If device does not implement pairing function, this is plaintext PIN
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString

21.3.1.37 RETURN_CODE IDTechSDK.IDT_BTPay.emv_cancelTransaction (string *ident* = " ")

Cancel EMV Transaction

Cancels the currently executing EMV transaction.

Parameters

<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.
--------------	--

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString

21.3.1.38 RETURN_CODE IDTechSDK.IDT_BTPay.emv_completeTransaction (bool *commError*, byte[] *authCode*, byte[] *iad*, byte[] *tlvScripts*, byte[] *tlv*, string *ident* = " ")

Complete EMV Transaction Request

Completes the EMV transaction for an ICC card when online authorization request is received from emv_↔
authenticateTransaction

The tags will be returned in the callback routine.

Parameters

<i>commError</i>	Communication error with host. Set to TRUE if host was unreachable, or FALSE if host response received. If Communication error, authCode, iad, tlvScripts can be null.
<i>authCode</i>	Authorization code from host. Two bytes. Example 0x3030. (Tag value 8A). Required
<i>iad</i>	Issuer Authentication Data, if any. Example 0x11223344556677883030 (tag value 91).
<i>tlvScripts</i>	71/72 scripts, if any
<i>tlv</i>	Additional TVL data to return with transaction results (if any)

NOTE: sending tag DFEE51 with a value of 01 (DFEE510101) as and Additional TLV data will Suppress the results. Used for QuickChip implementation.

Parameters

<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.
--------------	--

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString

NOTE: There are three possible outcomes for Authorization Code: Approval, Refer To Bank, Decline. The kernel maps these three outcomes to valid authorization response codes using tag DFEE1B through 8 bytes: {2 bytes Approval Code}{2 bytes Referral Code}{2 bytes Decline Code}{2 bytes RFU} If your gateway uses "00" for Approval, "01" for Referral, and "05" for Decline, then DFEE1B 08 3030 3031 3035 0000 If you use an authorization code value that is not defined in DFEE1B, the kernel will use the DECLINE value of DFEE1B by default.

21.3.1.39 **RETURN_CODE** IDTechSDK.IDT_BTPay.emv_getEMVConfigurationCheckValue (ref string *response*, string *ident* = " ")

Get EMV Kernel configuration check value info

Parameters

<i>response</i>	Response returned of Kernel configuration check value info
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString

21.3.1.40 **RETURN_CODE** IDTechSDK.IDT_BTPay.emv_getEMVKernelCheckValue (ref string *response*, string *ident* = " ")

Get EMV Kernel check value info

Parameters

<i>response</i>	Response returned of Kernel check value info
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString

21.3.1.41 **RETURN_CODE** IDTechSDK.IDT_BTPay.emv_getEMVKernelVersion (ref string *response*, string *ident* = " ")

Polls device for EMV Kernel Version

Parameters

<i>response</i>	Response returned of Kernel Version
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString

21.3.1.42 **RETURN_CODE** IDTechSDK.IDT_BTPay.emv_removeAllApplicationData (string *ident* = " ")

Remove All Application Data

Removes all the Application Data

Parameters

<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.
--------------	--

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString

21.3.1.43 RETURN_CODE IDTechSDK.IDT_BTPay.emv_removeAllCAPK (string *ident* = " ")

Remove All Certificate Authority Public Key

Removes all the CAPK

Parameters

<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.
--------------	--

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString

21.3.1.44 RETURN_CODE IDTechSDK.IDT_BTPay.emv_removeAllCRL (string *ident* = " ")

Remove All Certificate Revocation List Entries

Removes all CRLEntry entries

Parameters

<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.
--------------	--

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString

21.3.1.45 RETURN_CODE IDTechSDK.IDT_BTPay.emv_removeApplicationData (byte[] *AID*, string *ident* = " ")

Remove Application Data by AID

Removes the Application Data as specified by the AID name passed as a parameter

Parameters

<i>AID</i>	Name of ApplicationID Must be between 5 and 16 bytes
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString

21.3.1.46 RETURN_CODE IDTechSDK.IDT_BTPay.emv_removeCAPK (byte[] *capk*, string *ident* = " ")

Remove Certificate Authority Public Key

Removes the CAPK as specified by the RID/Index

Parameters

<i>capk</i>	6 byte CAPK = 5 bytes RID + 1 byte INDEX
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString

21.3.1.47 RETURN_CODE IDTechSDK.IDT_BTPay.emv_removeCRL (byte[] *crlList*, string *ident* = " ")

Remove Certificate Revocation List Entries

Removes CRL entries as specified by the RID and Index and serial number passed as 9 bytes

Parameters

<i>crlList</i>	containing the list of CRL to remove: [CRL1][CRL2]...[CRLn] where each [CRL] is 9 bytes: [5 bytes RID][1 byte CAPK Index][3 bytes serial number]
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString

21.3.1.48 RETURN_CODE IDTechSDK.IDT_BTPay.emv_removeTerminalData (string *ident* = " ")

Remove Terminal Data

Removes the Terminal Data

Parameters

<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.
--------------	--

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString

21.3.1.49 RETURN_CODE IDTechSDK.IDT_BTPay.emv_retrieveAIDList (ref byte *response*[], string *ident* = " ")

Retrieve AID list

Returns all the AID names installed on the terminal.

Parameters

<i>response</i>	array of AID name byte arrays
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString

21.3.1.50 RETURN_CODE IDTechSDK.IDT_BTPay.emv_retrieveApplicationData (byte[] AID, ref byte[] tlv, string ident = " ")

Retrieve Application Data by AID

Retrieves the Application Data as specified by the AID name passed as a parameter.

Parameters

<i>AID</i>	Name of ApplicationID. Must be between 5 and 16 bytes
<i>tlv</i>	The TLV elements of the requested AID
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString

21.3.1.51 RETURN_CODE IDTechSDK.IDT_BTPay.emv_retrieveCAPK (byte[] capk, ref byte[] key, string ident = " ")

Retrieve Certificate Authority Public Key

Retrieves the CAPK as specified by the RID/Index passed as a parameter.

Parameters

<i>capk</i>	6 bytes CAPK = 5 bytes RID + 1 byte Index
<i>key</i>	<p>Response returned as a CAKey format: [5 bytes RID][1 byte Index][1 byte Hash Algorithm][1 byte Encryption Algorithm][20 bytes HashValue][4 bytes Public Key Exponent][2 bytes Modulus Length][Variable bytes Modulus] Where:</p> <ul style="list-style-type: none"> • Hash Algorithm: The only algorithm supported is SHA-1. The value is set to 0x01 • Encryption Algorithm: The encryption algorithm in which this key is used. Currently support only one type: RSA. The value is set to 0x01. • HashValue: Which is calculated using SHA-1 over the following fields: RID & Index & Modulus & Exponent • Public Key Exponent: Actually, the real length of the exponent is either one byte or 3 bytes. It can have two values: 3 (Format is 0x00 00 00 03), or 65537 (Format is 0x00 01 00 01) • Modulus Length: LenL LenH Indicated the length of the next field. • Modulus: This is the modulus field of the public key. Its length is specified in the field above.
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString

21.3.1.52 RETURN_CODE IDTechSDK.IDT_BTPay.emv_retrieveCAPKList (ref byte[] keys, string ident = " ")

Retrieve the Certificate Authority Public Key list

Returns all the CAPK RID and Index installed on the terminal.

Parameters

<i>keys</i>	[key1][key2]...[keyn], each key 6 bytes where key = 5 bytes RID + 1 byte index
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString

21.3.1.53 RETURN_CODE IDTechSDK.IDT_BTPay.emv_retrieveCRLList (ref byte[] list, string ident = " ")

Retrieve the Certificate Revocation List

Returns the CRL entries on the terminal.

Parameters

<i>list</i>	[CRL1][CRL2]...[CRLn], each CRL 9 bytes where CRL = 5 bytes RID + 1 byte index + 3 bytes serial number
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString

21.3.1.54 RETURN_CODE IDTechSDK.IDT_BTPay.emv_retrieveTerminalData (ref byte[] tlv, string ident = " ")

Retrieve Terminal Data

Retrieves the Terminal Data.

Parameters

<i>tlv</i>	Response returned as a TLV
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString

21.3.1.55 RETURN_CODE IDTechSDK.IDT_BTPay.emv_retrieveTransactionResult (byte[] tags, ref IDTTransactionData tlv, string ident = " ")

Retrieve Transaction Results

Retrieves specified EMV tags from the currently executing transaction.

Parameters

<i>tags</i>	Tags to be retrieved. Example 0x9F028A will retrieve tags 9F02 and 8A
<i>tlv</i>	All requested tags returned as unencrypted, encrypted and masked TLV data in <code>IDDTransactionData</code> object
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with `device_getResponseCodeString`

21.3.1.56 RETURN_CODE `IDTechSDK.IDT_BTPay.emv_setApplicationData (byte[] name, byte[] tlv, string ident = " ")`

Set Application Data by AID

Sets the Application Data as specified by the application name and TLV data

Parameters

<i>name</i>	Application name, 10-32 ASCII hex characters representing 5-16 bytes Example "a0000000031010"
<i>tlv</i>	Application data in TLV format
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with `device_getResponseCodeString`

21.3.1.57 RETURN_CODE `IDTechSDK.IDT_BTPay.emv_setCAPK (byte[] key, string ident = " ")`

Set Certificate Authority Public Key

Sets the CAPK as specified by the CAKey structure

Parameters

<i>key</i>	<p>CAKey format: [5 bytes RID][1 byte Index][1 byte Hash Algorithm][1 byte Encryption Algorithm][20 bytes HashValue][4 bytes Public Key Exponent][2 bytes Modulus Length][Variable bytes Modulus] Where:</p> <ul style="list-style-type: none"> • Hash Algorithm: The only algorithm supported is SHA-1. The value is set to 0x01 • Encryption Algorithm: The encryption algorithm in which this key is used. Currently support only one type: RSA. The value is set to 0x01. • HashValue: Which is calculated using SHA-1 over the following fields: RID & Index & Modulus & Exponent • Public Key Exponent: Actually, the real length of the exponent is either one byte or 3 bytes. It can have two values: 3 (Format is 0x00 00 00 03), or 65537 (Format is 0x00 01 00 01) • Modulus Length: LenL LenH Indicated the length of the next field. • Modulus: This is the modulus field of the public key. Its length is specified in the field above.
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString

21.3.1.58 RETURN_CODE IDTechSDK.IDT_BTPay.emv_setCRL (byte[] list, string ident = " ")

Set Certificate Revocation List

Sets the CRL

Parameters

<i>list</i>	CRL Entries containing the RID, Index, and serial numbers to set [CRL1][CRL2]...[CRLn] where each [CRL] is 9 bytes: [5 bytes RID][1 byte CAPK Index][3 bytes serial number]
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString

21.3.1.59 RETURN_CODE IDTechSDK.IDT_BTPay.emv_setTerminalData (byte[] tlv, string ident = " ")

Set Terminal Data

Sets the Terminal Data

Parameters

<i>tlv</i>	TerminalData TLV Data
<i>ident</i>	Optional identifier

Return values

<i>RETURN_CODE</i>	Return codes listed as typedef enum in IDTCommon:RETURN_CODE. Values can be parsed with IDT_BTPay::device_getResponseCodeString:()
--------------------	--

21.3.1.60 RETURN_CODE IDTechSDK.IDT_BTPay.emv_setTerminalMajorConfiguration (int configuration, string ident = " ")

Set Terminal Major Configuration

Sets the Terminal Data Major Configuration setting

Parameters

<i>configuration</i>	The configuration to set (1-5)
----------------------	--------------------------------

Return values

<i>RETURN_CODE</i>	Return codes listed as typedef enum in IDTCommon:RETURN_CODE. Values can be parsed with IDT_BTPay::device_getResponseCodeString:()
--------------------	--

21.3.1.61 **RETURN_CODE** IDTechSDK.IDT_BTPay.emv_startTransaction (double *amount*, double *amtOther*, int *exponent*, int *type*, int *timeout*, byte[] *tags*, bool *forceOnline*, bool *isFastEMV* = false, string *ident* = " ")

Start EMV Transaction Request

Authorizes the EMV transaction for an ICC card

The tags will be returned in the callback routine.

Parameters

<i>amount</i>	Transaction amount value (tag value 9F02)
<i>amtOther</i>	Other amount value, if any (tag value 9F03)
<i>exponent</i>	Number of characters after decimile point
<i>type</i>	Transaction type (tag value 9C).
<i>timeout</i>	Timeout value in seconds.
<i>tags</i>	Any other tags to be included in the request. Passed as TLV Data. Example, tag 9F02 with amount 0x000000000100 would be 0x9F0206000000000100 If tags 9F02 (amount),9F03 (other amount), or 9C (transaction type) are included, they will take priority over these values supplied as individual parameters to this method.
<i>forceOnline</i>	TRUE = do not allow offline approval, FALSE = allow ICC to approve offline if terminal capable Note: To request tags to be included in default response, use tag DFEE1A, and specify tag list. Example four tags 9F02, 9F36, 95, 9F37 to be included in response = DFEE1A079F029F369F37
<i>isFastEMV</i>	If TRUE, it will populate the IDTTransactionData.fastEMV with ASCII data similar to IDTech FastEMV KB output, after performing an auto-authenticate and auto-complete with ResultCode = Could Not Contact Host
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString

21.3.1.62 **RETURN_CODE** IDTechSDK.IDT_BTPay.emv_trySetTerminalData (byte[] *tlv*, ref byte[] *rejectedTLV*, ref byte[] *convertedTLV*, bool *overwrite* = false, string *ident* = " ")

Try to Set Terminal Data

Attempts to set the Terminal Data. Use this functions to attempt to set the terminal data to the device. This function will allow the cross-device tag compatible writing of IDTech Legacy/New tag definitions

Parameters

<i>tlv</i>	TerminalData TLV Data
<i>rejectedTLV</i>	Contains the tags (if any) that were rejected by the firmware
<i>convertedTLV</i>	Contains the tags (if any) that were converted and accepted by the firmware
<i>overwrite</i>	TRUE = add TLV to existing tags, FALSE = replace existing tags with TLV
<i>ident</i>	Optional identifier

Return values

RETURN_CODE	Return codes listed as typedef enum in IDTCommon:RETURN_CODE. Values can be parsed with IDT_BTPay::device_getResponseCodeString:()
--------------------	--

21.3.1.63 `static int IDTechSDK.IDT_BTPay.getCommandTimeout (string ident = " ") [static]`

Get Command Timeout

Gets the default timeout (in milliseconds) waiting for a blocking command response

Parameters

<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.
--------------	--

Return values

<i>time</i>	Time
-------------	------

21.3.1.64 `RETURN_CODE IDTechSDK.IDT_BTPay.icc_disable (string ident = " ")`

ICC Function enable/disable Disable ICC function

Parameters

<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.
--------------	--

Returns

RETURN_CODE: Values can be parsed with `device_getResponseCodeString`

21.3.1.65 `RETURN_CODE IDTechSDK.IDT_BTPay.icc_enable (bool withNotification, string ident = " ")`

ICC Function enable/disable Enable ICC function with or without seated notification

Parameters

<i>withNotification</i>	<ul style="list-style-type: none"> • true: with notification when ICC seated status changed, • false: without notification.
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with `device_getResponseCodeString`

21.3.1.66 `RETURN_CODE IDTechSDK.IDT_BTPay.icc_exchangeAPDU (string c_APDU, ref byte[] response, string ident = " ")`

Exchange APDU with plain text For Non-SRED Augusta Only

Sends an APDU packet to the ICC. If successful, response is the APDU data in response parameter.

Parameters

<i>c_APDU</i>	APDU data packet
---------------	------------------

Parameters

<i>response</i>	Unencrypted APDU response
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString

21.3.1.67 RETURN_CODE IDTechSDK.IDT_BTPay.icc_exchangeEncryptedAPDU (string *c_APDU*, ref byte[] *response*, string *ident* = " ")

Exchange APDU with encrypted data For SRED Augusta Only

Sends an APDU packet to the ICC. If successful, response is the APDU data in response parameter.

Parameters

<i>c_APDU</i>	KSN + encrypted APDU data packet, or no KSN (use last known KSN) + encrypted APDU data packet With KSN: [0A][KSN][Encrypted C-APDU] Without KSN: [00][Encrypted C-APDU]
---------------	---

The format of Raw C-APDU Data Structure of [m-bytes Encrypted C-APDU] is below:

- m = 2 bytes Valid C-APDU Length + x bytes Valid C-APDU + y bytes Padding (0x00) Note: For TDES mode: 2+x should be multiple of 8. If it was not multiple of 8, unit should padded y bytes 0x00 automatically (2+x+y should be multiple of 8). For AES mode: 2+x should be multiple of 16. If it was not multiple of 16, unit should padded y bytes 0x00 automatically (2+x+y should be multiple of 16).

Parameters

<i>response</i>	<p>encrypted APDU response. Can be three options:</p> <ul style="list-style-type: none"> • [00] + [Plaintext R-APDU] • [01] + [0A] + [KSN] + [n bytes Encrypted R-APDU without Status Bytes] + [2 bytes Status Bytes] • [01] + [00] + [n bytes Encrypted R-APDU without Status Bytes] + [2 bytes Status Bytes] <p>The KSN, when provided, will be 10 bytes. The KSN will only be provided when it has changed since the last provided KSN. Each card Power-On generates a new KSN. During a sequence of commands where the KSN is identical, the first response will have a KSN length set to [0x0A] followed by the KSN, while subsequent commands with the same KSN value will have a KSN length of [0x00] followed by the Encrypted R-APDU without Status Bytes.</p>
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString

21.3.1.68 RETURN_CODE IDTechSDK.IDT_BTPay.icc_getAPDU_KSN (ref byte[] *ksn*, string *ident* = " ")

Get APDU KSN

Retrieves the KSN used in ICC Encrypted APDU usage

@param ksn Returns the encrypted APDU packet KSN

Parameters

<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.
--------------	--

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString

21.3.1.69 RETURN_CODE IDTechSDK.IDT_BTPay.icc_getFunctionStatus (ref bool *enabled*, ref bool *withNotification*, string *ident* = " ")

Get ICC Function status Get ICC Function status about enable/disable and with or without seated notification

Parameters

<i>enabled</i>	<ul style="list-style-type: none"> • true: ICC Function enabled, • false: means disabled.
<i>withNotification</i>	true means with notification when ICC seated status changed. false means without notification.
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString

21.3.1.70 RETURN_CODE IDTechSDK.IDT_BTPay.icc_getICCReaderStatus (ref byte *status*, string *ident* = " ")

Get Reader Status

Returns the reader status

Parameters

<i>status</i>	Pointer that will return with the ICCReaderStatus results. bit 0: 0 = ICC Power Not Ready, 1 = ICC Powered bit 1: 0 = Card not seated, 1 = card seated
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString

21.3.1.71 RETURN_CODE IDTechSDK.IDT_BTPay.icc_getKeyFormatForICCDUKPT (ref byte *format*, string *ident* = " ")

Get Key Format For ICC DUKPT

Specifies how data will be encrypted with Data Key or PIN key (if DUKPT key loaded)

Parameters

<i>format</i>	Response returned from method: <ul style="list-style-type: none"> • 'TDES': Encrypted card data with TDES if DUKPT Key had been loaded.(default) • 'AES': Encrypted card data with AES if DUKPT Key had been loaded. • 'NONE': No Encryption.
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString

21.3.1.72 RETURN_CODE IDTechSDK.IDT_BTPay.icc_getKeyTypeForICCDUKPT (ref byte *type*, string *ident* = " ")

Get Key Type for ICC DUKPT

Specifies the key type used for ICC DUKPT encryption

Parameters

<i>type</i>	Response returned from method: <ul style="list-style-type: none"> • 'DATA': Encrypted card data with Data Key DUKPT Key had been loaded.(default) • 'PIN': Encrypted card data with PIN Key if DUKPT Key had been loaded.
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString

21.3.1.73 RETURN_CODE IDTechSDK.IDT_BTPay.icc_powerOffICC (string *ident* = " ")

Power Off ICC

Powers down the ICC

Parameters

<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.
--------------	--

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString

If Success, empty If Failure, ASCII encoded data of error string

21.3.1.74 RETURN_CODE IDTechSDK.IDT_BTPay.icc_powerOnICC (ref byte[] *ATR*, string *ident* = " ")**Power On ICC**

Power up the currently selected microprocessor card in the ICC reader

Parameters

<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.
--------------	--

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString

21.3.1.75 RETURN_CODE IDTechSDK.IDT_BTPay.icc_setKeyFormatForICCDUKPT (byte *encryption*, string *ident* = " ")**Set Key Format for ICC DUKPT**

Sets how data will be encrypted, with either TDES or AES (if DUKPT key loaded)

Parameters

<i>encryption</i>	Encryption Type <ul style="list-style-type: none">• 00: Encrypt with TDES• 01: Encrypt with AES
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString

21.3.1.76 RETURN_CODE IDTechSDK.IDT_BTPay.icc_setKeyTypeForICCDUKPT (byte *encryption*, string *ident* = " ")**Set Key Type for ICC DUKPT Key**

Sets which key the data will be encrypted with, with either Data Key or PIN key (if DUKPT key loaded)

Parameters

<i>encryption</i>	Encryption Type <ul style="list-style-type: none">• 00: Encrypt with Data Key• 01: Encrypt with PIN Key
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString

21.3.1.77 RETURN_CODE IDTechSDK.IDT_BTPay.msr_cancelMSRSwipe (string *ident* = " ")

Disable MSR Swipe Cancels MSR swipe request.

Parameters

<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.
--------------	--

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString

21.3.1.78 RETURN_CODE IDTechSDK.IDT_BTPay.msr_captureMode (bool *isBufferMode*, bool *withNotification* = false, string *ident* = " ")

Set MSR Capture Mode.

For Non-SRED Augusta Only

Switch between Auto mode and Buffer mode. Auto mode only available on KB interface

Parameters

<i>isBufferMode</i>	<ul style="list-style-type: none"> • true: Enter into Buffer mode. • false: Enter into Auto mode. KB mode only. Swipes automatically captured and sent to keyboard buffer
<i>withNotification</i>	<ul style="list-style-type: none"> • true: with notification when swiped MSR data. • false: without notification when swiped MSR data.
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString

21.3.1.79 RETURN_CODE IDTechSDK.IDT_BTPay.msr_disable (string *ident* = " ")

Disable MSR function.

Parameters

<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.
--------------	--

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString

21.3.1.80 RETURN_CODE IDTechSDK.IDT_BTPay.msr_getClearPANID (ref byte *value*, string *ident* = " ")

Get Clear PAN Digits

Returns the number of digits that begin the PAN that will be in the clear

Parameters

<i>value</i>	Number of digits in clear. Values are char '0' - '6':
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString

21.3.1.81 RETURN_CODE IDTechSDK.IDT_BTPay.msr_getExpirationMask (ref byte *value*, string *ident* = " ")

Get Expiration Masking

Get the flag that determines if to mask the expiration date

Parameters

<i>value</i>	'0' = masked, '1' = not-masked
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString

21.3.1.82 RETURN_CODE IDTechSDK.IDT_BTPay.msr_getFunctionStatus (ref bool *enabled*, ref bool *isBufferMode*, ref bool *withNotification*, string *ident* = " ")

Get MSR Function status Get MSR Function status about enable/disable and with or without seated notification

Parameters

<i>enabled</i>	<ul style="list-style-type: none"> • true: MSR Function enabled. • false: MSR Function disabled.
<i>isBufferMode</i>	<ul style="list-style-type: none"> • true: in the buffer mode. • false: in the auto mode.
<i>withNotification</i>	<ul style="list-style-type: none"> • true: with notification when swiped MSR Card. • false: without notification when swiped MSR Card.
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString

21.3.1.83 RETURN_CODE IDTechSDK.IDT_BTPay.msr_getSetting (byte *setting*, ref byte *value*, string *ident* = " ")

Get Single MSR Setting value

Returns the encryption used for sweep data

Parameters

<i>setting</i>	MSR Setting to retrieve
<i>value</i>	MSR Setting value
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString

21.3.1.84 RETURN_CODE IDTechSDK.IDT_BTPay.msr_getSettings (byte *setting*, ref byte[] *value*, string *ident* = " ")

Get Multi MSR Setting value

Returns the encryption used for sweep data

Parameters

<i>setting</i>	MSR Setting to retrieve
<i>value</i>	MSR Setting value
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString

21.3.1.85 RETURN_CODE IDTechSDK.IDT_BTPay.msr_getSwipeEncryption (ref byte *encryption*, string *ident* = " ")

Get Swipe Data Encryption

For Non-SRED Augusta Only

Returns the encryption used for swipe data

Parameters

<i>encryption</i>	1 = TDES, 2 = AES, 0 = NONE
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString

21.3.1.86 RETURN_CODE IDTechSDK.IDT_BTPay.msr_getSwipeForcedEncryptionOption (ref byte *option*, string *ident* = " ")

Get Swipe Data Encryption

Gets the swipe force encryption options

Parameters

<i>option</i>	Byte using lower four bits as flags. 0 = Force Encryption Off, 1 = Force Encryption On bit0 = Track 1 bit1 = Track 2 bit2 = Track 3 bit4 = Track 3 Card Option 0
---------------	--

Example: Response 0x03 = Track1/Track2 Forced Encryption, Track3/Track3-0 no Forced Encryption

Parameters

<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.
--------------	--

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString

21.3.1.87 RETURN_CODE IDTechSDK.IDT_BTPay.msr_getSwipeMaskOption (ref byte *option*, string *ident* = " ")

Get Swipe Mask Option

Gets the swipe mask/clear data sending option

Parameters

<i>option</i>	Byte using lower three bits as flags. 0 = Mask Option Off, 1 = Mask Option On bit0 = Track 1 bit1 = Track 2 bit2 = Track 3
---------------	--

Example: Response 0x03 = Track1/Track2 Masked Option ON, Track3 Masked Option Off

@param ident Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString

21.3.1.88 RETURN_CODE IDTechSDK.IDT_BTPay.msr_RetrieveWhiteList (ref byte[] *value*, string *ident* = " ")

Retrieve MSR White List

For Non-SRED Augusta Only

Parameters

<i>value</i>	is the white list data which is ASN.1 Block format
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString

21.3.1.89 RETURN_CODE IDTechSDK.IDT_BTPay.msr_setClearPANID (byte *val*, string *ident* = " ")

Set Clear PAN Digits

Sets the amount of digits shown in the clear (not masked) at the beginning of the returned PAN value

Parameters

<i>val</i>	Number of digits to show in clear. Range 0-6.
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString

21.3.1.90 RETURN_CODE IDTechSDK.IDT_BTPay.msr_setExpirationMask (bool *mask*, string *ident* = " ")

Set Expiration Masking

Sets the flag to mask the expiration date

Parameters

<i>mask</i>	TRUE = mask expiration
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString

21.3.1.91 RETURN_CODE IDTechSDK.IDT_BTPay.msr_setSetting (byte *setting*, byte *value*, string *ident* = " ")

Set Single MSR Setting value

@param setting MSR Setting to set
@param value MSR Setting value

Parameters

<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.
--------------	--

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString

21.3.1.92 RETURN_CODE IDTechSDK.IDT_BTPay.msr_setSettings (byte *setting*, byte[] *value*, string *ident* = " ")

Set Multi MSR Setting value

Parameters

<i>setting</i>	MSR Setting to set
<i>value</i>	MSR Setting value
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString

21.3.1.93 RETURN_CODE IDTechSDK.IDT_BTPay.msr_setSwipeEncryption (byte *encryption*, string *ident* = " ")

Set Swipe Data Encryption

For Non-SRED Augusta Only

Sets the swipe encryption method

Parameters

<i>encryption</i>	1 = TDES, 2 = AES
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString

21.3.1.94 RETURN_CODE IDTechSDK.IDT_BTPay.msr_setSwipeForcedEncryptionOption (bool *track1*, bool *track2*, bool *track3*, bool *track3card0*, string *ident* = " ")

Set Swipe Force Encryption

Sets the swipe force encryption options

Parameters

<i>track1</i>	Force encrypt track 1
<i>track2</i>	Force encrypt track 2
<i>track3</i>	Force encrypt track 3
<i>track3card0</i>	Force encrypt track 3 when card type is 0
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString

21.3.1.95 **RETURN_CODE** IDTechSDK.IDT_BTPay.msr_setSwipeMaskOption (*bool track1*, *bool track2*, *bool track3*, *string ident = " "*)

Set Swipe Mask Option

Sets the swipe mask/clear data sending option

Parameters

<i>track1</i>	Mask track 1 allowed
<i>track2</i>	Mask track 2 allowed
<i>track3</i>	Mask track 3 allowed
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with `device_getResponseCodeString`

21.3.1.96 **RETURN_CODE** IDTechSDK.IDT_BTPay.msr_setWhiteList (*byte[] value*, *byte[] MAC*, *string ident = " "*)

Set MSR White List

For Non-SRED Augusta Only

Parameters

<i>value</i>	the white list data which is ASN.1 Block format
<i>MAC</i>	Message Authenticate Code. For non-PCI, use null
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with `device_getResponseCodeString`

21.3.1.97 **RETURN_CODE** IDTechSDK.IDT_BTPay.msr_startMSRSwipe (*int timeout*, *string ident = " "*)

Enable MSR Swipe

Enables MSR, waiting for swipe to occur. Allows track selection. Returns IDTMSRData instance to `deviceDelegate->::swipeMSRData:()`

Parameters

<i>timeout</i>	Swipe Timeout Value
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with `device_getResponseCodeString`

21.3.1.98 **RETURN_CODE** IDTechSDK.IDT_BTPay.msr_switchUSBInterfaceMode (*bool blsUSBKeyboardMode*)

Switch the USB interface mode between USB HID and USB KB mode.

For Non-SRED Augusta Only

Parameters

<i>blsUSBKeyboardMode</i>	USB interface mode <ul style="list-style-type: none"> • true: Enter into USB Keyboard mode • false: Enter into USB HID mode
---------------------------	---

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString

21.3.1.99 RETURN_CODE IDTechSDK.IDT_BTPay.msr_switchUSBInterfaceMode (bool *blsUSBKeyboardMode*, ref string *ident*)

Switch the USB interface mode between USB HID and USB KB mode.

For Non-SRED Augusta Only

Parameters

<i>blsUSBKeyboardMode</i>	USB interface mode <ul style="list-style-type: none"> • true: Enter into USB Keyboard mode • false: Enter into USB HID mode
<i>ident</i>	Device ID to send command to.

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString

21.3.1.100 static String IDTechSDK.IDT_BTPay.SDK_Version () [static]

SDK Version

- All Devices

Returns the current version of SDK

Returns

Framework version

21.3.1.101 static void IDTechSDK.IDT_BTPay.setCallback (CallBack *my_Callback*) [static]

Set Callback

Sets the class callback

Parameters

<i>my_Callback</i>	The callback function to receive the response message from device. defined as follows. delegate void CallBack(IDT_DEVICE_Types sender, DeviceState state, byte[] data, IDTTTransactionData card, EMV_Callback emvCallback, RETURN_CODE transactionResultCode);
--------------------	---

21.3.1.102 static void IDTechSDK.IDT_BTPay.setCallback (IntPtr *my_Callback*, SynchronizationContext *context*)
[static]

Set Callback

Sets the class callback

Parameters

<i>my_Callback</i>	The callback function to receive the response message from device. defined as follows. public unsafe delegate void MFCCallBack(Parameters parameters);
<i>context</i>	The context of the UI thread

21.3.1.103 static void IDTechSDK.IDT_BTPay.setCommandTimeout (int *milliseconds*, string *ident* = " ") [static]

Set Command Timeout

Sets the default timeout (in milliseconds) waiting for a blocking command response

Parameters

<i>milliseconds</i>	Time
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

21.3.1.104 static bool IDTechSDK.IDT_BTPay.useSerialPort (int *port*) [static]

Use Serial Port Interface

Instructs SDK to attempt to use the Serial Port for communication with BTPay using default baud rate

Parameters

<i>port</i>	Serial Port to use. Example COM1 = 1.
-------------	---------------------------------------

Returns

bool TRUE=successful, FALSE=failure

21.3.1.105 static bool IDTechSDK.IDT_BTPay.useSerialPort (int *port*, int *baud*) [static]

Use Serial Port Interface with baud rate

Instructs SDK to attempt to use the Serial Port for communication with BTPay

Parameters

<i>port</i>	Serial Port to use. Example COM1 = 1.
<i>baud</i>	Baud rate to override default. Example 115200;

Returns

bool TRUE=successful, FALSE=failure

21.3.1.106 static bool IDTechSDK.IDT_BTPay.useSerialPortLinux (string *path*) [static]

Use Serial Port Interface on Linux

Instructs SDK to attempt to use the Serial Port for communication with BTMag using default baud rate on Linux implementations

Parameters

<i>path</i>	Path to use. Example /dev/ttyUSB*
-------------	-----------------------------------

Returns

bool TRUE=successful, FALSE=failure

21.3.1.107 static bool IDTechSDK.IDT_BTPay.useSerialPortLinux (string *path*, int *baud*) [static]

Use Serial Port Interface on Linux with baud rate

Instructs SDK to attempt to use the Serial Port for communication with BTPay on Linux implementations

Parameters

<i>path</i>	Path to use. Example /dev/ttyUSB*
<i>baud</i>	Baud rate to override default. Example 115200;

Returns

bool TRUE=successful, FALSE=failure

21.3.2 Property Documentation

21.3.2.1 IDT_BTPay IDTechSDK.IDT_BTPay.SharedController [static],[get]

Singleton Instance

Establishes an singleton instance of [IDT_BTPay](#) class.

Returns

Instance of [IDT_BTPay](#)

The documentation for this class was generated from the following file:

- /Users/randy/Repo/universal-sdk/IDTechSDK_STD/DeviceHeaders/IDT_BTPay.cs

21.4 IDTechSDK.IDT_CM100 Class Reference

Public Member Functions

- RETURN_CODE [device_getFirmwareVersion](#) (ref string response)
- RETURN_CODE [device_sendPAE](#) (string command, ref string response, int timeout, string ip="")
- RETURN_CODE [device_getVersion](#) (byte version, ref string response)
- RETURN_CODE [device_sendDataCommand](#) (string cmd, bool calcLRC, ref byte[] response)
- RETURN_CODE [device_sendBeep](#) (int time)
- RETURN_CODE [ctls_startTransaction](#) (double amount, double amtOther, int exponent, int type, int timeout, byte[] tags, bool isFastEMV=false)
- RETURN_CODE [device_setDateTime](#) ()
- RETURN_CODE [ctls_cancelTransaction](#) ()
- RETURN_CODE [ctls_enableL2Application](#) (bool enable)
- RETURN_CODE [device_getRKIStatus](#) (bool isProd, bool isMultiKey, ref string status, ref Dictionary< string, RKI_KEY_TYPE > keys, string ident="")
- RETURN_CODE [device_getAnyRKIStatus](#) (bool isMultiKey, ref string status, ref Dictionary< string, RKI_KEY_TYPE > keys, string ident="")
- RETURN_CODE [device_sendConfigurationFromZip](#) (byte[] zip, string filename, VC_OPERATION_TYPE type, bool matchFW, string ident="", bool isForeground=false)
- RETURN_CODE [device_sendConfiguration](#) (string filename, VC_OPERATION_TYPE type, bool matchFW, string ident="", bool isForeground=false)
- RETURN_CODE [device_readConfigurationToMemory](#) (string memo, Configs.InstallRules rules, List< Configs.DeviceCommand > cmds, string ident="", bool isForeground=false)
- RETURN_CODE [device_getConfigurationFromMemory](#) (ref string json, string ident="")
- RETURN_CODE [device_RemoteKeyInjection](#) (RKI_KEY_TYPE type, string keyName, string ident="", bool performOnForeground=false)

Static Public Member Functions

- static void [initSC](#) ()
- static int [getCommandTimeout](#) ()
- static void [setCommandTimeout](#) (int milliseconds)
- static void [setCallback](#) (Callback my_Callback)
- static void [setCallback](#) (IntPtr my_Callback, SynchronizationContext context)
- static String [SDK_Version](#) ()

Properties

- static [IDT_CM100 SharedController](#) [get]

21.4.1 Detailed Description

Class for [IDT_CM100](#) ICC reader

21.4.2 Member Function Documentation

21.4.2.1 RETURN_CODE IDTechSDK.IDT_CM100.ctls_cancelTransaction ()

Cancel CTLS Transaction

Cancels the currently executing CTLS transaction.

Returns

RETURN_CODE: Values can be parsed with `errorCode.getErrorString()`

21.4.2.2 RETURN_CODE IDTechSDK.IDT_CM100.ctls_enableL2Application (bool *enable*)

Enable L2 Application

Enables or disables the L2 Application. Disable when L1 functionality is needed

@param enable TRUE = application enabled, FALSE = application disabled

Returns

RETURN_CODE: Values can be parsed with `errorCode.getErrorString()`

21.4.2.3 RETURN_CODE IDTechSDK.IDT_CM100.ctls_startTransaction (double *amount*, double *amtOther*, int *exponent*, int *type*, int *timeout*, byte[] *tags*, bool *isFastEMV* = false)

Start EMV Transaction Request

Authorizes the CTLS transaction for a CTLS card

The tags will be returned in the callback routine.

Parameters

<i>amount</i>	Transaction amount value (tag value 9F02)
<i>amtOther</i>	Other amount value, if any (tag value 9F03)
<i>type</i>	Transaction type (tag value 9C).
<i>timeout</i>	Timeout value in seconds.
<i>tags</i>	Any other tags to be included in the request. Passed as TLV Data. Example, tag 9F02 with amount 0x000000000100 would be 0x9F0206000000000100 If tags 9F02 (amount), 9F03 (other amount), or 9C (transaction type) are included, they will take priority over these values supplied as individual parameters to this method.
<i>isFastEMV</i>	If TRUE, it will populate the IDTTransactionData.fastEMV with ASCII data similar to IDTech FastEMV KB output, after performing an auto-authenticate and auto-complete with ResultCode = Could Not Contact Host

Returns

RETURN_CODE: Values can be parsed with `errorCode.getErrorString()`

21.4.2.4 RETURN_CODE IDTechSDK.IDT_CM100.device_getAnyRKIStatus (bool *isMultiKey*, ref string *status*, ref Dictionary< string, RKI_KEY_TYPE > *keys*, string *ident* = " ")

Get RKI Status Polls the RKI servers to validate device RKI eligibility. Note: if device type is known in advance (production or demo device), it is more efficient to use `device_getRKIStatus` instead

Parameters

<i>isMultiKey</i>	True = MultiKey, False = SingleKey
<i>status</i>	RKI status returned as a string
<i>keys</i>	Returns all available keys
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with `errorCode.getErrorString()`. When no data is available, return code = RETURN_CODE_NO_DATA_AVAILABLE

21.4.2.5 RETURN_CODE IDTechSDK.IDT_CM100.device_getConfigurationFromMemory (ref string json, string ident = " ")

Read Device Configuration From Memory Retrieves the ViVOconfig configuration data from memory after a successful device_readConfigurationToMemory

Parameters

<i>json</i>	Returns the .json file data that can be saved to disk
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with `errorCode.getErrorString()`. When no data is available, return code = RETURN_CODE_NO_DATA_AVAILABLE

21.4.2.6 RETURN_CODE IDTechSDK.IDT_CM100.device_getFirmwareVersion (ref string response)

Polls device for Firmware Version

Parameters

<i>response</i>	Response returned of Firmware Version
-----------------	---------------------------------------

Returns

RETURN_CODE: Values can be parsed with `errorCode.getErrorString()`

21.4.2.7 RETURN_CODE IDTechSDK.IDT_CM100.device_getRKIStatus (bool isProd, bool isMultiKey, ref string status, ref Dictionary< string, RKI_KEY_TYPE > keys, string ident = " ")

Get RKI Status From Specified Server Polls the RKI server to validate device RKI eligibility

Parameters

<i>isProd</i>	TRUE = poll production RKI server, FALSE = poll demo RKI server
<i>isMultiKey</i>	True = MultiKey, False = SingleKey
<i>status</i>	RKI status returned as a string
<i>keys</i>	Returns dictionary of available keys
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with `errorCode.getErrorString()`. When no data is available, return code = RETURN_CODE_NO_DATA_AVAILABLE

21.4.2.8 RETURN_CODE IDTechSDK.IDT_CM100.device_getVersion (byte *version*, ref string *response*)

Get Version

Parameters

<i>version</i>	The version to return: <ul style="list-style-type: none"> • 0x01 = READER • 0x91 = L1 • 0xA1 = PAYPASS • 0xB1 = VISA • 0xC1 = AMEX • 0xD1 = DISCOVER
<i>response</i>	Response returned of Firmware Version

Returns

RETURN_CODE: Values can be parsed with `errorCode.getErrorString()`

21.4.2.9 RETURN_CODE IDTechSDK.IDT_CM100.device_readConfigurationToMemory (string *memo*, Configs.InstallRules *rules*, List< Configs.DeviceCommand > *cmds*, string *ident* = "", bool *isForeground* = false)

Read Device Configuration To Memory Executes a ViVOconfig read of device settings and stores the results in memory

Once a ViVOconfig read starts, all messages will be returned to the callback as `DeviceState.ViVOconfig`. The ViVOconfig operation is complete when the return code from a `DeviceState.ViVOconfig` message is one of the following values:

- RETURN_CODE_SDK_VIVOCONFIG_READ_CONFIG_SUCCESS = Read process completed successfully. Execute `device_getConfigurationFromMemory` to retrieve the configuration data
- RETURN_CODE_SDK_VIVOCONFIG_READ_CONFIG_FAILED = Read process FAILED.

Parameters

<i>memo</i>	Optional: populates the memo field of the captured .json file
<i>rules</i>	Optional: populates the .json file with available install rules
<i>cmds</i>	Optional: populates the .json file with commands to execute
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.
<i>isForeground</i>	If TRUE, will block program until update complete. Otherwise, FALSE performs update on background.

Returns

RETURN_CODE: Values can be parsed with `errorCode.getErrorString()`. When no data is available, return code = RETURN_CODE_NO_DATA_AVAILABLE

21.4.2.10 **RETURN_CODE** IDTechSDK.IDT_CM100.device_RemoteKeyInjection (*RKI_KEY_TYPE type*, *string keyName*, *string ident = ""*, *bool performOnForeground = false*)

Remote Key Injection Performs a remote key injection for the device

Parameters

<i>type</i>	Remote Key Injection Type
<i>keyName</i>	Name of key (optional)
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.
<i>performOnForeground</i>	TRUE = block during process (default is false)

Returns

RETURN_CODE: Values can be parsed with `errorCode.getErrorString()`. When no data is available, return code = **RETURN_CODE_NO_DATA_AVAILABLE**

21.4.2.11 **RETURN_CODE** IDTechSDK.IDT_CM100.device_sendBeep (*int time*)

Beep Device

Parameters

<i>Time</i>	to beep in .1 second increments. Example: 16 = 1.6 seconds
-------------	--

Returns

RETURN_CODE: Values can be parsed with `errorCode.getErrorString()`

21.4.2.12 **RETURN_CODE** IDTechSDK.IDT_CM100.device_sendConfiguration (*string filename*, *VC_OPERATION_TYPE type*, *bool matchFW*, *string ident = ""*, *bool isForeground = false*)

Send Configuration Executes a ViVOconfig update or verify to a device

Once a ViVOconfig write or verify starts, all messages will be returned to the callback as `DeviceState.ViVOconfig`. The ViVOconfig operation is complete when the return code from a `DeviceState.ViVOconfig` message is one of the following values:

- **RETURN_CODE_SDK_VIVOCONFIG_VERIFY_SUCCESS** = Verification process completed successfully. No differences found.
- **RETURN_CODE_SDK_VIVOCONFIG_VERIFY_SUCCESS_WITH_WARNING** = Verification process completed with warnings.
- **RETURN_CODE_SDK_VIVOCONFIG_VERIFY_FAILED** = Verification process FAILED
- **RETURN_CODE_SDK_VIVOCONFIG_WRITE_SUCCESS** = Write process completed successfully.
- **RETURN_CODE_SDK_VIVOCONFIG_WRITE_SUCCESS_WITH_WARNING** = Write process completed with warnings
- **RETURN_CODE_SDK_VIVOCONFIG_WRITE_FAILED** = Write process FAILED

Parameters

<i>filename</i>	The .json configuration file to update the device with NOTE: You can also use this parameter to pass the path of where the .json file is located at on local storage instead
<i>filename</i>	The name of the .json configuration file
<i>type</i>	VC_OPERATION_TYPE

- VC_OPERATION_TYPE_WRITE = Write File To Device, Hash must validate
- VC_OPERATION_TYPE_VERIFY = Verify Device With File
- VC_OPERATION_TYPE_WRITE_IGNORE_HASH = Write File To Device, Ignore Hash
- VC_OPERATION_TYPE_WRITE_FIX_HASH = Write File To Device, Fix Hash If Necessary

Parameters

<i>matchFW</i>	TRUE = Device FW must match file FW, FALSE = Don't validate device FW against file
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.
<i>isForeground</i>	If TRUE, will block program until update complete. Otherwise, FALSE performs update on background.

Returns

RETURN_CODE: Values can be parsed with `errorCode.getErrorString()`. When no data is available, return code = RETURN_CODE_NO_DATA_AVAILABLE

21.4.2.13 RETURN_CODE IDTechSDK.IDT_CM100.device_sendConfigurationFromZip (byte[] zip, string filename, VC_OPERATION_TYPE type, bool matchFW, string ident = "", bool isForeground = false)

Send Configuration From Zip Executes a ViVOconfig update or verify to a device using files from a .zip archive

Once a ViVOconfig write or update starts, all messages will be returned to the callback as DeviceState.ViVOconfig. The ViVOconfig operation is complete when the return code from a DeviceState.ViVOconfig message is one of the following values:

- RETURN_CODE_SDK_VIVOCONFIG_VERIFY_SUCCESS = Verification process completed successfully. No differences found.
- RETURN_CODE_SDK_VIVOCONFIG_VERIFY_SUCCESS_WITH_WARNING = Verification process completed with warnings.
- RETURN_CODE_SDK_VIVOCONFIG_VERIFY_FAILED = Verification process FAILED
- RETURN_CODE_SDK_VIVOCONFIG_WRITE_SUCCESS = Write process completed successfully.
- RETURN_CODE_SDK_VIVOCONFIG_WRITE_SUCCESS_WITH_WARNING = Write process completed with warnings
- RETURN_CODE_SDK_VIVOCONFIG_WRITE_FAILED = Write process FAILED

Parameters

<i>zip</i>	.Zip archive containing the .json configuration file and also any associated image assets NOTE: You can also use this parameter to pass the path of where the .zip file is located at on local storage instead
<i>filename</i>	The name of the .json configuration file
<i>type</i>	VC_OPERATION_TYPE

- VC_OPERATION_TYPE_WRITE = Write File To Device, Hash must validate
- VC_OPERATION_TYPE_VERIFY = Verify Device With File
- VC_OPERATION_TYPE_WRITE_IGNORE_HASH = Write File To Device, Ignore Hash
- VC_OPERATION_TYPE_WRITE_FIX_HASH = Write File To Device, Fix Hash If Necessary

Parameters

<i>matchFW</i>	TRUE = Device FW must match file FW, FALSE = Don't validate device FW against file
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.
<i>isForeground</i>	If TRUE, will block program until update complete. Otherwise, FALSE performs update on background.

Returns

RETURN_CODE: Values can be parsed with `errorCode.getErrorString()`. When no data is available, return code = RETURN_CODE_NO_DATA_AVAILABLE

21.4.2.14 RETURN_CODE IDTechSDK.IDT_CM100.device_sendDataCommand (string *cmd*, bool *calcLRC*, ref byte[] *response*)

Send a NSData object to device

Sends a command represented by the provide NSData object to the device through the accessory protocol.

Parameters

<i>cmd</i>	NSData representation of command to execute
<i>calcLRC</i>	If TRUE, this will wrap command with start/length/lrc/sum/end: '{STX}{Len_Low}{Len_High} data {CheckLRC} {CheckSUM} {ETX}'
<i>response</i>	Response data

Returns

RETURN_CODE: Values can be parsed with `errorCode.getErrorString()`

21.4.2.15 RETURN_CODE IDTechSDK.IDT_CM100.device_sendPAE (string *command*, ref string *response*, int *timeout*, string *ip* = " ")

Send Payment Application Engine Command

Executes a PAE command

Parameters

<i>command</i>	ASCII command string, should start with "*PAE"
<i>response</i>	command response
<i>timeout</i>	timeout waiting for PAE response
<i>ip</i>	Optional IP address when connected via TCP/IP

Returns

RETURN_CODE: Values can be parsed with `errorCode.getErrorString()`

21.4.2.16 `RETURN_CODE IDTechSDK.IDT_CM100.device_setDateTime ()`**Set DateTime**

Sets the CM100 date/time to the current date/time of the host operating system

Returns

RETURN_CODE: Values can be parsed with `errorCode.getErrorString()`

21.4.2.17 `static int IDTechSDK.IDT_CM100.getCommandTimeout () [static]`**Get Command Timeout**

Gets the default timeout (in milliseconds) waiting for a blocking command response

Return values

<i>time</i>	Time
-------------	------

21.4.2.18 `static String IDTechSDK.IDT_CM100.SDK_Version () [static]`**SDK Version**

- All Devices

Returns the current version of SDK

Returns

Framework version

21.4.2.19 `static void IDTechSDK.IDT_CM100.setCallback (Callback my_Callback) [static]`**Set Callback**

Sets the class callback

21.4.2.20 `static void IDTechSDK.IDT_CM100.setCallback (IntPtr my_Callback, SynchronizationContext context) [static]`**Set Callback**

Sets the class callback

Parameters

<i>my_Callback</i>	The callback function to receive the response message from device. defined as follows. <code>public unsafe delegate void MFCCallBack(Parameters parameters);</code>
<i>context</i>	The context of the UI thread

21.4.2.21 `static void IDTechSDK.IDT_CM100.setCommandTimeout (int milliseconds) [static]`

Set Command Timeout

Sets the default timeout (in milliseconds) waiting for a blocking command response

Parameters

<i>milliseconds</i>	Time
---------------------	------

21.4.3 Property Documentation

21.4.3.1 `IDT_CM100 IDTechSDK.IDT_CM100.SharedController [static],[get]`

Singleton Instance

Establishes an singleton instance of [IDT_CM100](#) class.

Returns

Instance of [IDT_CM100](#)

The documentation for this class was generated from the following file:

- `/Users/randy/Repo/universal-sdk/IDTechSDK_STD/DeviceHeaders/IDT_CM100.cs`

21.5 IDTechSDK.IDT_K100 Class Reference

Public Member Functions

- RETURN_CODE [device_getFirmwareVersion](#) (ref string response)
- RETURN_CODE [device_StartRKI](#) (int type)
- RETURN_CODE [device_sendDataCommand](#) (string cmd, bool calcLRC, ref byte[] response)
- RETURN_CODE [device_sendDataCommand_ext](#) (string cmd, bool calcLRC, ref byte[] response, int timeout, bool noResponse)
- RETURN_CODE [config_getModelNumber](#) (ref string response)
- bool [config_setCmdTimeOutDuration](#) (int newTimeOut)
- RETURN_CODE [pin_getEncryptedPIN](#) (int keyType, string PAN, string message, int timeout, bool isAES=false, string ident="")
- RETURN_CODE [pin_getFunctionKey](#) ()
- RETURN_CODE [pin_sendBeep](#) (int frequency, int duration)
- RETURN_CODE [device_rebootDevice](#) ()
- RETURN_CODE [pin_cancelPINEntry](#) ()
- RETURN_CODE [device_getKeyStatus](#) (ref byte[] status)
- RETURN_CODE [pin_enableKeypad](#) ()
- RETURN_CODE [device_startRKI](#) (bool isTest)
- RETURN_CODE [device_getDateTimeString](#) (ref string str)
- RETURN_CODE [config_setBaudRate](#) (int baud)
- RETURN_CODE [config_getBaudRate](#) (ref int baud)
- RETURN_CODE [device_getRKIStatus](#) (bool isProd, bool isMultiKey, ref string status, ref Dictionary< string, RKI_KEY_TYPE > keys, string ident="")
- RETURN_CODE [device_getAnyRKIStatus](#) (bool isMultiKey, ref string status, ref Dictionary< string, RKI_KEY_TYPE > keys, string ident="")
- RETURN_CODE [device_updateFirmwareFromZip](#) (byte[] zipfile, string ident="", bool isForeground=false)

- RETURN_CODE [device_sendConfigurationFromZip](#) (byte[] zip, string filename, VC_OPERATION_TYPE type, bool matchFW, string ident="", bool isForeground=false)
- RETURN_CODE [device_sendConfiguration](#) (string filename, VC_OPERATION_TYPE type, bool matchFW, string ident="", bool isForeground=false)
- RETURN_CODE [device_readConfigurationToMemory](#) (string memo, Configs.InstallRules rules, List< Configs.DeviceCommand > cmds, string ident="", bool isForeground=false)
- RETURN_CODE [device_getConfigurationFromMemory](#) (ref string json, string ident="")
- RETURN_CODE [device_RemoteKeyInjection](#) (RKL_KEY_TYPE type, string keyName, string ident="", bool performOnForeground=false)

Static Public Member Functions

- static bool [useSerialPort](#) (int port)
- static int [getCommandTimeout](#) ()
- static void [setCommandTimeout](#) (int milliseconds)
- static bool [useSerialPort](#) (int port, int baud)
- static bool [useSerialPortLinux](#) (string path)
- static bool [useSerialPortLinux](#) (string path, int baud)
- static bool [closeSerialPort](#) ()
- static bool [closeUSB](#) ()
- static void [initSC](#) ()
- static void [setCallback](#) (Callback my_Callback)
- static void [setCallback](#) (IntPtr my_Callback, SynchronizationContext context)
- static String [SDK_Version](#) ()

Properties

- static [IDT_K100 SharedController](#) [get]

21.5.1 Member Function Documentation

21.5.1.1 static bool IDTechSDK.IDT_K100.closeSerialPort () [static]

Close Serial Port Interface

Instructs SDK to close the Serial Port if connected to K100

Returns

bool TRUE=successful, FALSE=failure

21.5.1.2 static bool IDTechSDK.IDT_K100.closeUSB () [static]

Close USB

Instructs SDK to close the USB if connected to K100

Returns

bool TRUE=successful, FALSE=failure

21.5.1.3 RETURN_CODE IDTechSDK.IDT_K100.config_getBaudRate (ref int baud)

Get Baud Rate

Gets the buad rate for RS-232 communication.

Parameters

<i>baud</i>	<ul style="list-style-type: none">• 2 = 2400• 3 = 4800• 4 = 9600• 6 = 19200• 7 = 38400• 9 = 115200
-------------	---

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString

21.5.1.4 RETURN_CODE IDTechSDK.IDT_K100.config_getModelNumber (ref string *response*)

Polls device for Model Number

Parameters

<i>response</i>	Returns Model Number
-----------------	----------------------

Returns

RETURN_CODE: Values can be parsed with errorCode.getErrorString()

21.5.1.5 RETURN_CODE IDTechSDK.IDT_K100.config_setBaudRate (int *baud*)

Set Baud Rate

Sets the buad rate for RS-232 communication.

Parameters

<i>baud</i>	<ul style="list-style-type: none">• 2 = 2400• 3 = 4800• 4 = 9600• 6 = 19200• 7 = 38400• 9 = 115200
-------------	---

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString

21.5.1.6 `bool IDTechSDK.IDT_K100.config_setCmdTimeOutDuration (int newTimeOut)`

Command Acknowledgement Timeout

Sets the amount of seconds to wait for an {ACK} to a command before a timeout. Responses should normally be received under one second. Default is 3 seconds

Parameters

<i>newTimeOut</i>	Timeout value. Valid range 1 - 60 seconds
-------------------	---

Returns

Success flag. Determines if value was set and in range.

21.5.1.7 `RETURN_CODE IDTechSDK.IDT_K100.device_getAnyRKIStatus (bool isMultiKey, ref string status, ref Dictionary< string, RKI_KEY_TYPE > keys, string ident = " ")`

Get RKI Status Polls the RKI servers to validate device RKI eligibility. Note: if device type is known in advance (production or demo device), it is more efficient to use `device_getRKIStatus` instead

Parameters

<i>isMultiKey</i>	True = MultiKey, False = SingleKey
<i>status</i>	RKI status returned as a string
<i>keys</i>	Returns all available keys
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with `errorCode.getErrorString()`. When no data is available, return code = RETURN_CODE_NO_DATA_AVAILABLE

21.5.1.8 `RETURN_CODE IDTechSDK.IDT_K100.device_getConfigurationFromMemory (ref string json, string ident = " ")`

Read Device Configuration From Memory Retrieves the ViVOconfig configuration data from memory after a successful `device_readConfigurationToMemory`

Parameters

<i>json</i>	Returns the .json file data that can be saved to disk
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with `errorCode.getErrorString()`. When no data is available, return code = RETURN_CODE_NO_DATA_AVAILABLE

21.5.1.9 `RETURN_CODE IDTechSDK.IDT_K100.device_getDateTimeString (ref string str)`

Get Date Time String

Gets current system date and time of the device.

Parameters

<i>str</i>	The Date/Time string
------------	----------------------

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString

21.5.1.10 RETURN_CODE IDTechSDK.IDT_K100.device_getFirmwareVersion (ref string *response*)

Polls device for Firmware Version

Parameters

<i>response</i>	Response returned of Firmware Version
-----------------	---------------------------------------

Returns

RETURN_CODE: Values can be parsed with errorCode.getErrorString()

21.5.1.11 RETURN_CODE IDTechSDK.IDT_K100.device_getKeyStatus (ref byte[] *status*)

Get Key Status

Gets the status of loaded keys

Parameters

<i>status</i>	<p>Blocks of 4 bytes, format is {Key Index and Key Name} {key slot} {key status}: {Key Index and Key Name} is 1 byte.</p> <ul style="list-style-type: none"> • 0x14 = LCL Key Encryption Key (Master or KEK) • 0x01 = PIN DUKPT Key • 0x0C = RKI-KEK DUKPT Key • 0x08 = PIN Master Key • 0x0D = Pairing BDK Key (PINPAD) {keyslot is 2 bytes.Range is 0–9999 {key status} is 1byte. 0 – Not Exist, 1 – Exist, 0xFF – (Stop. Only Valid for DUKPT Key)
---------------	--

Returns

RETURN_CODE: Values can be parsed with errorCode.getErrorString()

21.5.1.12 RETURN_CODE IDTechSDK.IDT_K100.device_getRKIStatus (bool *isProd*, bool *isMultiKey*, ref string *status*, ref Dictionary< string, RKI_KEY_TYPE > *keys*, string *ident* = " ")

Get RKI Status From Specified Server Polls the RKI server to validate device RKI eligibility

Parameters

<i>isProd</i>	TRUE = poll production RKI server, FALSE = poll demo RKI server
<i>isMultiKey</i>	True = MultiKey, False = SingleKey
<i>status</i>	RKI status returned as a string
<i>keys</i>	Returns dictionary of available keys
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with `errorCode.getErrorString()`. When no data is available, return code = RETURN_CODE_NO_DATA_AVAILABLE

21.5.1.13 RETURN_CODE IDTechSDK.IDT_K100.device_readConfigurationToMemory (string *memo*, Configs.InstallRules *rules*, List< Configs.DeviceCommand > *cmds*, string *ident* = " ", bool *isForeground* = false)

Read Device Configuration To Memory Executes a ViVOconfig read of device settings and stores the results in memory

Once a ViVOconfig read starts, all messages will be returned to the callback as DeviceState.ViVOconfig. The ViVOconfig operation is complete when the return code from a DeviceState.ViVOconfig message is one of the following values:

- RETURN_CODE_SDK_VIVOCONFIG_READ_CONFIG_SUCCESS = Read process completed successfully. Execute `device_getConfigurationFromMemory` to retrieve the configuration data
- RETURN_CODE_SDK_VIVOCONFIG_READ_CONFIG_FAILED = Read process FAILED.

Parameters

<i>memo</i>	Optional: populates the memo field of the captured .json file
<i>rules</i>	Optional: populates the .json file with available install rules
<i>cmds</i>	Optional: populates the .json file with commands to execute
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.
<i>isForeground</i>	If TRUE, will block program until update complete. Otherwise, FALSE performs update on background.

Returns

RETURN_CODE: Values can be parsed with `errorCode.getErrorString()`. When no data is available, return code = RETURN_CODE_NO_DATA_AVAILABLE

21.5.1.14 RETURN_CODE IDTechSDK.IDT_K100.device_rebootDevice ()

Reboot Device

Executes a command to restart the device.

- Card data is cleared, resetting card status bits.
- Response data of the previous command is cleared.
- Resetting firmware.

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString

21.5.1.15 RETURN_CODE IDTechSDK.IDT_K100.device_RemoteKeyInjection (RKI_KEY_TYPE type, string keyName, string ident = " ", bool performOnForeground = false)

Remote Key Injection Performs a remote key injection for the device

Parameters

<i>type</i>	Remote Key Injection Type
<i>keyName</i>	Name of key (optional)
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.
<i>performOnForeground</i>	TRUE = block during process (default is false)

Returns

RETURN_CODE: Values can be parsed with errorCode.getErrorString(). When no data is available, return code = RETURN_CODE_NO_DATA_AVAILABLE

21.5.1.16 RETURN_CODE IDTechSDK.IDT_K100.device_sendConfiguration (string filename, VC_OPERATION_TYPE type, bool matchFW, string ident = " ", bool isForeground = false)

Send Configuration Executes a ViVOconfig update or verify to a device

Once a ViVOconfig write or verify starts, all messages will be returned to the callback as DeviceState.ViVOconfig. The ViVOconfig operation is complete when the return code from a DeviceState.ViVOconfig message is one of the following values:

- RETURN_CODE_SDK_VIVOCONFIG_VERIFY_SUCCESS = Verification process completed successfully. No differences found.
- RETURN_CODE_SDK_VIVOCONFIG_VERIFY_SUCCESS_WITH_WARNING = Verification process completed with warnings.
- RETURN_CODE_SDK_VIVOCONFIG_VERIFY_FAILED = Verification process FAILED
- RETURN_CODE_SDK_VIVOCONFIG_WRITE_SUCCESS = Write process completed successfully.
- RETURN_CODE_SDK_VIVOCONFIG_WRITE_SUCCESS_WITH_WARNING = Write process completed with warnings
- RETURN_CODE_SDK_VIVOCONFIG_WRITE_FAILED = Write process FAILED

Parameters

<i>filename</i>	The .json configuration file to update the device with NOTE: You can also use this parameter to pass the path of where the .json file is located at on local storage instead
<i>filename</i>	The name of the .json configuration file
<i>type</i>	VC_OPERATION_TYPE

- VC_OPERATION_TYPE_WRITE = Write File To Device, Hash must validate
- VC_OPERATION_TYPE_VERIFY = Verify Device With File

- VC_OPERATION_TYPE_WRITE_IGNORE_HASH = Write File To Device, Ignore Hash
- VC_OPERATION_TYPE_WRITE_FIX_HASH = Write File To Device, Fix Hash If Necessary

Parameters

<i>matchFW</i>	TRUE = Device FW must match file FW, FALSE = Don't validate device FW against file
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.
<i>isForeground</i>	If TRUE, will block program until update complete. Otherwise, FALSE performs update on background.

Returns

RETURN_CODE: Values can be parsed with `errorCode.getErrorString()`. When no data is available, return code = RETURN_CODE_NO_DATA_AVAILABLE

21.5.1.17 RETURN_CODE IDTechSDK.IDT_K100.device_sendConfigurationFromZip (byte[] zip, string filename, VC_OPERATION_TYPE type, bool matchFW, string ident = "", bool isForeground = false)

Send Configuration From Zip Executes a ViVOconfig update or verify to a device using files from a .zip archive

Once a ViVOconfig write or update starts, all messages will be returned to the callback as DeviceState.ViVOconfig. The ViVOconfig operation is complete when the return code from a DeviceState.ViVOconfig message is one of the following values:

- RETURN_CODE_SDK_VIVOCONFIG_VERIFY_SUCCESS = Verification process completed successfully. No differences found.
- RETURN_CODE_SDK_VIVOCONFIG_VERIFY_SUCCESS_WITH_WARNING = Verification process completed with warnings.
- RETURN_CODE_SDK_VIVOCONFIG_VERIFY_FAILED = Verification process FAILED
- RETURN_CODE_SDK_VIVOCONFIG_WRITE_SUCCESS = Write process completed successfully.
- RETURN_CODE_SDK_VIVOCONFIG_WRITE_SUCCESS_WITH_WARNING = Write process completed with warnings
- RETURN_CODE_SDK_VIVOCONFIG_WRITE_FAILED = Write process FAILED

Parameters

<i>zip</i>	.Zip archive containing the .json configuration file and also any associated image assets NOTE: You can also use this parameter to pass the path of where the .zip file is located at on local storage instead
<i>filename</i>	The name of the .json configuration file
<i>type</i>	VC_OPERATION_TYPE

- VC_OPERATION_TYPE_WRITE = Write File To Device, Hash must validate
- VC_OPERATION_TYPE_VERIFY = Verify Device With File
- VC_OPERATION_TYPE_WRITE_IGNORE_HASH = Write File To Device, Ignore Hash
- VC_OPERATION_TYPE_WRITE_FIX_HASH = Write File To Device, Fix Hash If Necessary

Parameters

<i>matchFW</i>	TRUE = Device FW must match file FW, FALSE = Don't validate device FW against file
----------------	--

Parameters

<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.
<i>isForeground</i>	If TRUE, will block program until update complete. Otherwise, FALSE performs update on background.

Returns

RETURN_CODE: Values can be parsed with `errorCode.getErrorString()`. When no data is available, return code = RETURN_CODE_NO_DATA_AVAILABLE

21.5.1.18 RETURN_CODE IDTechSDK.IDT_K100.device_sendDataCommand (string cmd, bool calcLRC, ref byte[] response)

Send a data command to the device

Sends a command to the device.

Parameters

<i>cmd</i>	String representation of command to execute
<i>calcLRC</i>	If TRUE, this will wrap command with start/length/lrc/sum/end: '{STX}{Len_Low}{Len_High} data {CheckLRC} {CheckSUM} {ETX}'
<i>response</i>	Response data

Returns

RETURN_CODE: Values can be parsed with `device_getResponseCodeString`

21.5.1.19 RETURN_CODE IDTechSDK.IDT_K100.device_sendDataCommand_ext (string cmd, bool calcLRC, ref byte[] response, int timeout, bool noResponse)

Send a data command to the device - extended

Sends a command to the device.

Parameters

<i>cmd</i>	String representation of command to execute
<i>calcLRC</i>	If TRUE, this will wrap command as NGA with start/length/lrc/sum/end: '{STX}{Len_Low}{Len_High} data {CheckLRC} {CheckSUM} {ETX}'
<i>response</i>	Response data
<i>timeout</i>	Timeout value waiting for response, in milliseconds (1000 = 1 second)
<i>noResponse</i>	if TRUE, this will not wait for a response and immediately return SUCCESS
<i>calcITP</i>	If TRUE, this will wrap command as ITP with start/end/lrc: '{STX} data {ETX}{CheckLRC}'

Returns

RETURN_CODE: Values can be parsed with `device_getResponseCodeString`

21.5.1.20 RETURN_CODE IDTechSDK.IDT_K100.device_StartRKI (int type)

Start Remote Key Injection

Starts a remote key injection request with IDTech RKI servers. Set/Get RKI url with IDT_Device.RKI_URL.

Parameters

<i>type</i>	0 = Symmetric RKI Demo Unit 1 = Symmetric RKI Production Unit 2 = PKI-RKI Demo Unit 3 = PKI-RKI Production Unit
-------------	---

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString

21.5.1.21 RETURN_CODE IDTechSDK.IDT_K100.device_startRKI (bool *isTest*)

Start Remote Key Injection

Starts a remote key injection request with IDTech RKI servers.

Parameters

<i>isTest</i>	TRUE = Demo Device, FALSE = Production Device
---------------	---

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString

21.5.1.22 RETURN_CODE IDTechSDK.IDT_K100.device_updateFirmwareFromZip (byte[] *zipfile*, string *ident* = "", bool *isForeground* = false)

Update Firmware From Zip

Performs one or more device firmware updates from firmware files passed as a compressed archive with a valid configuration file (.json format)

Parameters

<i>zipfile</i>	Zip file containing on or more firmware update files and a .json configuration file
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.
<i>isForeground</i>	If TRUE, will block program until firmware update complete. Otherwise, FALSE performs FW update on background.

Returns

RETURN_CODE: Values can be parsed with errorCode.getErrorString(string ident = "")

21.5.1.23 static int IDTechSDK.IDT_K100.getCommandTimeout () [static]

Get Command Timeout

Gets the default timeout (in milliseconds) waiting for a blocking command response

Return values

<i>time</i>	Time
-------------	------

21.5.1.24 RETURN_CODE IDTechSDK.IDT_K100.pin_cancelPINEntry ()

Cancel PIN Entry

Cancel “Get Function Key” & “Get Encrypted PIN” & “Get Numeric” & “Get Amount”

Returns

RETURN_CODE: Values can be parsed with `errorCode.getErrorString()`

21.5.1.25 RETURN_CODE IDTechSDK.IDT_K100.pin_enableKeypad ()

Enable Keypad

Captures keypad button presses until Timeout reached, or `pin_cancelPINEntry` is executed, whichever comes first

Timeout is hardwired to 3 minutes

Returns

RETURN_CODE: Values can be parsed with `errorCode.getErrorString()`

21.5.1.26 RETURN_CODE IDTechSDK.IDT_K100.pin_getEncryptedPIN (int keyType, string PAN, string message, int timeout, bool isAES = false, string ident = " ")

Get Encrypted PIN

Requests PIN Entry

Parameters

<i>keyType</i>	<ul style="list-style-type: none"> • 0x00- MKSK-TDES: External Plaintext PAN • 0x01- DUKPT-TDES: External Plaintext PAN • 0x10 MKSK-TDES: External Ciphertext PAN • 0x11 DUKPT-TDES: External Ciphertext PAN
<i>PAN</i>	Account Number
<i>message</i>	Message to display

Timeout is hard wired 3 minutes

Parameters

<i>isAES</i>	set to TRUE if PEK is AES, not TDES
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with `errorCode.getErrorString()`

21.5.1.27 RETURN_CODE IDTechSDK.IDT_K100.pin_getFunctionKey ()

Get Function Key

Captures a function key

- Backspace = B
- Cancel = C
- Enter = E
- * = *
- # = #
- Help = ?
- Function Key 1 = F1
- Function Key 2 = F2
- Function Key 3 = F3

Timeout hard wired 3 minutes

Returns

RETURN_CODE: Values can be parsed with `errorCode.getErrorString()`

21.5.1.28 RETURN_CODE IDTechSDK.IDT_K100.pin_sendBeep (int *frequency*, int *duration*)

Send Beep

Executes a beep request.

Parameters

<i>frequency</i>	Frequency, range 200-20000Hz
<i>duration</i>	Duration, range 16-65535ms

Returns

RETURN_CODE: Values can be parsed with `device.getResponseCodeString`

21.5.1.29 static String IDTechSDK.IDT_K100.SDK_Version () [static]

SDK Version

- All Devices

Returns the current version of SDK

Returns

Framework version

21.5.1.30 `static void IDTechSDK.IDT_K100.setCallback (Callback my_Callback) [static]`

Set Callback

Sets the class callback

21.5.1.31 `static void IDTechSDK.IDT_K100.setCallback (IntPtr my_Callback, SynchronizationContext context) [static]`

Set Callback

Sets the class callback

Parameters

<i>my_Callback</i>	The callback function to receive the response message from device. defined as follows. public unsafe delegate void MFCCallBack(Parameters parameters);
<i>context</i>	The context of the UI thread

21.5.1.32 `static void IDTechSDK.IDT_K100.setCommandTimeout (int milliseconds) [static]`

Set Command Timeout

Sets the default timeout (in milliseconds) waiting for a blocking command response

Parameters

<i>milliseconds</i>	Time
---------------------	------

21.5.1.33 `static bool IDTechSDK.IDT_K100.useSerialPort (int port) [static]`

Use Serial Port Interface

Instructs SDK to attempt to use the Serial Port for communication with K100 using default baud rate

Parameters

<i>port</i>	Serial Port to use. Example COM1 = 1.
-------------	---------------------------------------

Returns

bool TRUE=successful, FALSE=failure

21.5.1.34 `static bool IDTechSDK.IDT_K100.useSerialPort (int port, int baud) [static]`

Use Serial Port Interface with baud rate K100

Parameters

<i>port</i>	Serial Port to use. Example COM1 = 1.
<i>baud</i>	Baud rate to override default. Example 115200;

Returns

bool TRUE=successful, FALSE=failure

21.5.1.35 static bool IDTechSDK.IDT_K100.useSerialPortLinux (string *path*) [static]

Use Serial Port Interface on Linux

Instructs SDK to attempt to use the Serial Port for communication with BTMag using default baud rate on Linux implementations

Parameters

<i>path</i>	Path to use. Example /dev/ttyUSB*
-------------	-----------------------------------

Returns

bool TRUE=successful, FALSE=failure

21.5.1.36 static bool IDTechSDK.IDT_K100.useSerialPortLinux (string *path*, int *baud*) [static]

Use Serial Port Interface on Linux with baud rate

Instructs SDK to attempt to use the Serial Port for communication with BTPay on Linux implementations

Parameters

<i>path</i>	Path to use. Example /dev/ttyUSB*
<i>baud</i>	Baud rate to override default. Example 115200;

Returns

bool TRUE=successful, FALSE=failure

21.5.2 Property Documentation**21.5.2.1 IDT_K100 IDTechSDK.IDT_K100.SharedController [static], [get]**

Singleton Instance

Establishes an singleton instance of [IDT_K100](#) class.

Returns

Instance of [IDT_K100](#)

The documentation for this class was generated from the following file:

- /Users/randy/Repo/universal-sdk/IDTechSDK_STD/DeviceHeaders/IDT_K100.cs

21.6 IDTechSDK.IDT_KioskIII Class Reference**Public Member Functions**

- RETURN_CODE [config_getSerialNumber](#) (ref string response, string ident="")

- RETURN_CODE [config_setBaudRate](#) (int baud, string ident="")
- RETURN_CODE [device_retrieveAIDList](#) (ref byte[] response, string ident="")
- RETURN_CODE [device_sendDataCommand](#) (string cmd, bool calcLRC, ref byte[] response, string ident="")
- RETURN_CODE [device_sendDataCommand_ext](#) (string cmd, bool calcLRC, ref byte[] response, int timeout, bool noResponse, string ident="")
- RETURN_CODE [device_getFirmwareVersion](#) (ref string response, string ident="")
- RETURN_CODE [device_getMerchantRecord](#) (int index, ref byte[] record, string ident="")
- RETURN_CODE [device_getTransactionResults](#) (ref IDTTransactionData results, string ident="")
- RETURN_CODE [device_pingDevice](#) (string ident="")
- RETURN_CODE [device_controlUserInterface](#) (byte[] values, string ident="")
- RETURN_CODE [device_rebootDevice](#) (string ident="")
- RETURN_CODE [device_controlLED](#) (byte indexLED, byte control, string ident="")
- RETURN_CODE [device_sendVivoCommandP2](#) (byte command, byte subCommand, byte[] data, ref byte[] response, string ident="")
- RETURN_CODE [device_cancelTransaction](#) (string ident="")
- RETURN_CODE [device_sendVivoCommandP2_ext](#) (byte command, byte subCommand, byte[] data, ref byte[] response, int timeout, bool noResponse, string ident="")
- RETURN_CODE [ctls_retrieveAIDList](#) (ref byte[] response, string ident="")
- RETURN_CODE [ctls_getAllConfigurationGroups](#) (ref byte[] response, string ident="")
- RETURN_CODE [ctls_retrieveApplicationData](#) (byte[] AID, ref byte[] tlv, string ident="")
- RETURN_CODE [ctls_removeApplicationData](#) (byte[] AID, string ident="")
- RETURN_CODE [ctls_setApplicationData](#) (byte[] tlv, string ident="")
- RETURN_CODE [ctls_setDefaultConfiguration](#) (string ident="")
- RETURN_CODE [ctls_setConfigurationGroup](#) (byte[] tlv, string ident="")
- RETURN_CODE [ctls_retrieveTerminalData](#) (ref byte[] tlv, string ident="")
- RETURN_CODE [ctls_setTerminalData](#) (byte[] tlv, string ident="")
- RETURN_CODE [ctls_trySetTerminalData](#) (byte[] tlv, ref byte[] rejectedTLV, ref byte[] convertedTLV, string ident="")
- RETURN_CODE [ctls_getConfigurationGroup](#) (int group, ref byte[] tlv, string ident="")
- RETURN_CODE [ctls_removeConfigurationGroup](#) (int group, string ident="")
- RETURN_CODE [ctls_setCAPK](#) (byte[] key, string ident="")
- RETURN_CODE [ctls_retrieveCAPK](#) (byte[] capk, ref byte[] key, string ident="")
- RETURN_CODE [ctls_removeCAPK](#) (byte[] capk, string ident="")
- RETURN_CODE [ctls_removeAllCAPK](#) (string ident="")
- RETURN_CODE [ctls_retrieveCAPKList](#) (ref byte[] keys, string ident="")
- RETURN_CODE [device_startTransaction](#) (double amount, double amtOther, int exponent, int type, int timeout, byte[] tags, bool isFastEMV=false, string ident="")
- RETURN_CODE [device_activateTransaction](#) (int timeout, byte[] tags, bool isFastEMV=false, string ident="")
- RETURN_CODE [ctls_startTransaction](#) (double amount, double amtOther, int exponent, int type, int timeout, byte[] tags, bool isFastEMV=false, string ident="")
- RETURN_CODE [ctls_updateBalance](#) (byte statusCode, byte[] authCode, byte[] date, byte[] time, string ident="")
- RETURN_CODE [ctls_activateTransaction](#) (int timeout, byte[] tags, bool isFastEMV=false, string ident="")
- RETURN_CODE [ctls_cancelTransaction](#) (string ident="")
- RETURN_CODE [device_setBurstMode](#) (byte mode, string ident="")
- RETURN_CODE [device_setMerchantRecord](#) (int index, bool enabled, string merchantID, string merchantURL, string ident="")
- RETURN_CODE [device_pollForToken](#) (byte seconds, ref byte card, ref byte[] serialNumber, string ident="")
- RETURN_CODE [device_sendPAE](#) (string command, ref string response, int timeout, string ident="")
- RETURN_CODE [device_setPollMode](#) (byte mode, string ident="")
- RETURN_CODE [device_startRKI](#) (string ident="")
- RETURN_CODE [device_StartRKI](#) (int type, string ident="")
- RETURN_CODE [device_updateDeviceFirmware](#) (byte[] firmwareData, string ident="")
- RETURN_CODE [device_enablePassThrough](#) (bool enablePassThrough, string ident="")
- RETURN_CODE [felica_authentication](#) (byte[] key, string ident="")

- RETURN_CODE [felica_SendCommand](#) (byte[] command, ref byte[] response, string ident="")
- RETURN_CODE [felica_readWithMac](#) (int numBlocks, byte[] blockList, ref byte[] blocks, string ident="")
- RETURN_CODE [felica_writeWithMac](#) (int blockNumber, byte[] data, string ident="")
- RETURN_CODE [felica_read](#) (byte[] serviceCode, int numBlocks, byte[] blockList, ref byte[] blocks, string ident="")
- RETURN_CODE [felica_write](#) (byte[] serviceCode, int blockCount, byte[] blockList, byte[] data, ref byte[] statusFlag, string ident="")
- RETURN_CODE [ctls_nfcCommand](#) (byte[] nfcCmdPkt, ref byte[] response, string ident="")
- RETURN_CODE [felica_requestService](#) (byte[] nodeCode, ref byte[] response, string ident="")
- RETURN_CODE [device_getRKIStatus](#) (bool isProd, bool isMultiKey, ref string status, ref Dictionary< string, RKI_KEY_TYPE > keys, string ident="")
- RETURN_CODE [device_getAnyRKIStatus](#) (bool isMultiKey, ref string status, ref Dictionary< string, RKI_KEY_TYPE > keys, string ident="")
- RETURN_CODE [device_updateFirmwareFromZip](#) (byte[] zipfile, string ident="", bool isForeground=false)
- RETURN_CODE [device_sendConfigurationFromZip](#) (byte[] zip, string filename, VC_OPERATION_TYPE type, bool matchFW, string ident="", bool isForeground=false)
- RETURN_CODE [device_sendConfiguration](#) (string filename, VC_OPERATION_TYPE type, bool matchFW, string ident="", bool isForeground=false)
- RETURN_CODE [device_readConfigurationToMemory](#) (string memo, Configs.InstallRules rules, List< Configs.DeviceCommand > cmds, string ident="", bool isForeground=false)
- RETURN_CODE [device_getConfigurationFromMemory](#) (ref string json, string ident="")
- RETURN_CODE [device_RemoteKeyInjection](#) (RKI_KEY_TYPE type, string keyName, string ident="", bool performOnForeground=false)
- RETURN_CODE [config_setDEKVariantType](#) (byte type, string ident="")
- RETURN_CODE [config_getDEKVariantType](#) (ref byte type, string ident="")
- RETURN_CODE [config_setDUKPTEncryptionType](#) (byte type, string ident="")
- RETURN_CODE [config_getDUKPTEncryptionType](#) (ref byte type, string ident="")
- RETURN_CODE [config_getDUKPT_KSN](#) (ref byte[] ksn, string ident="")
- RETURN_CODE [config_getSalt_KCV](#) (ref byte[] kcv, string ident="")
- RETURN_CODE [config_checkDUKPTKey](#) (byte keyindex, ref byte[] val, string ident="")
- RETURN_CODE [config_setDUKPT_DEK_Attribution_TDES](#) (byte keyslot, byte outputMode, byte variant, string ident="")
- RETURN_CODE [config_setDUKPT_DEK_Attribution_AES](#) (byte keyslot, byte workingKey, byte keyUsage, string ident="")
- RETURN_CODE [config_getDUKPT_DEK_Attribution](#) (byte keyslot, ref byte mode, ref byte outputMode, ref byte workingKey, ref byte variant_keyUsage, string ident="")
- RETURN_CODE [config_setKeyslot_PEK_DEK](#) (bool isPEK, byte keyslot, string ident="")
- RETURN_CODE [config_getKeyslot_PEK_DEK](#) (ref byte keyslotPEK, ref byte keyslotDEK, string ident="")
- RETURN_CODE [config_setRKLKeys](#) (short keyNumber, byte[] tr31, byte[] nonce, byte[] hmac, ref byte[] kv, ref byte[] nonce_device, ref byte[] hmac_device, string ident="")

Static Public Member Functions

- static bool [useSerialPort](#) (int port)
- static bool [useSerialPortLinux](#) (string path)
- static bool [useSerialPortLinux](#) (string path, int baud)
- static int [getCommandTimeout](#) (string ident="")
- static void [setCommandTimeout](#) (int milliseconds, string ident="")
- static bool [useSerialPort](#) (int port, int baud, string ident="")
- static void [initSC](#) ()
- static void [setCallback](#) (Callback my_Callback)
- static void [setCallback](#) (IntPtr my_Callback, SynchronizationContext context)
- static String [SDK_Version](#) ()
- static void [lcd_retrieveMessage](#) (DisplayMessages.DISPLAY_MESSAGE_LANGUAGE lang, DisplayMessages.DISPLAY_MESSAGE_IDENTIFIER id, ref string line1, ref string line2)

Properties

- static [IDT_KioskIII SharedController](#) [get]

21.6.1 Detailed Description

Class for KioskIII and KioskIV ICC reader

21.6.2 Member Function Documentation

21.6.2.1 RETURN_CODE IDTechSDK.IDT_KioskIII.config_checkDUKPTKey (byte *keyindex*, ref byte[] *val*, string *ident* = " ")

Check DUKPT Key

This command checks whether a valid DUKPT key is stored at the specified slot and if a valid key is found then some basic information related to the type of key is returned. The actual Key data is never returned. This command can be used to check whether a key is already present before injecting a key in a KeyIndex to prevent overwriting an existing DUKPT key.

Parameters

<i>keyindex</i>	Data Encryption Key Index (usually 5)
<i>val</i>	<ul style="list-style-type: none"> • Byte 0 = Key State: 00h = Unused, 01h = Valid, 02h = End Of Life, FFh = Not Available • Byte 1-2 = Key Usage (ASCII). "D0" = Use to encrypt data • Byte 3 = Algorithm (ASCII). "T" = Triple DES • Byte 4 = Mode ofUse (ASCII). "E" = Encryption Only • Byte 5-6 = Key Version (ASCII). "00" = key version not used
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with `device_getResponseCodeString`

21.6.2.2 RETURN_CODE IDTechSDK.IDT_KioskIII.config_getDEKVariantType (ref byte *type*, string *ident* = " ")

Get Data Encryption Key Variant Type

Returns the Data Encryption Key Type

Parameters

<i>type</i>	<ul style="list-style-type: none"> • 0 = Data Variant • 1 = Pin Variant
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString

21.6.2.3 RETURN_CODE IDTechSDK.IDT_KioskIII.config_getDUKPT_DEK_Attribution (byte *keyslot*, ref byte *mode*, ref byte *outputMode_workingKey*, ref byte *variant_keyUsage*, string *ident* = " ")

Get DUKPT DEK Attribution based on KeySlot

Command settings can only be changed for each key one time.

Parameters

<i>keyslot</i>	Key Slot
<i>mode</i>	<ul style="list-style-type: none"> • 0 = TDES • 1 = AES
<i>outputMode_workingKey</i>	Output Mode is when TDES, Working Key is when AES <ul style="list-style-type: none"> • 0 = TDES / TDES • 1 = AES-128 / TDES3 • 2 = TransArmor / AES-128
<i>variant_keyUsage</i>	Variant is when TDES, Key Usage is when AES <ul style="list-style-type: none"> • 0 = Data Key / Encrypt/Decrypt • 1 = PIN Key / Encrypt • 2 = N/A / Decrypt
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString

21.6.2.4 RETURN_CODE IDTechSDK.IDT_KioskIII.config_getDUKPT_KSN (ref byte[] *ksn*, string *ident* = " ")

Get DUKPT Key Serial Number (KSN)

Host can use this command to retrieve the KSN of the DUKPT key.

Parameters

<i>ksn</i>	Key Serial Number
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString

21.6.2.5 RETURN_CODE IDTechSDK.IDT_Kioskill.config_getDUKPTEncryptionType (ref byte *type*, string *ident* = " ")

Get DUKPT Key Encryption Type

Returns DUKPT Key Encryption Type.

Parameters

<i>type</i>	<ul style="list-style-type: none"> • 0 = TDES • 1 = AES • 2 = TransArmor
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString

21.6.2.6 RETURN_CODE IDTechSDK.IDT_Kioskill.config_getKeySlot_PEK_DEK (ref byte *keyslotPEK*, ref byte *keyslotDEK*, string *ident* = " ")

Get KeySlot for DUKPT PEK and DUKPT DEK

- Returns the KeySlot for PEK and DEK (when exists).

Parameters

<i>keyslotPEK</i>	PEK KeySlot (0-9). Value of 255 (0xff) = does not exist
<i>keyslotDEK</i>	DEK KeySlot (0-9). Value of 255 (0xff) = does not exist
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString

21.6.2.7 RETURN_CODE IDTechSDK.IDT_Kioskill.config_getSalt_KCV (ref byte[] *kcv*, string *ident* = " ")

Get Salt KCV

Host can use this command to retrieve the Salt KCV.

Parameters

<i>kcv</i>	KCV
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString

21.6.2.8 RETURN_CODE IDTechSDK.IDT_KioskIII.config_getSerialNumber (ref string *response*, string *ident* = " ")

Polls device for Serial Number

Parameters

<i>response</i>	Returns Serial Number
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString

21.6.2.9 RETURN_CODE IDTechSDK.IDT_KioskIII.config_setBaudRate (int *baud*, string *ident* = " ")

Set Baud Rate

Sets the buad rate for RS-232 communication.

Parameters

<i>baud</i>	<ul style="list-style-type: none"> • 4 = 9600 • 6 = 19200 • 7 = 38400 • 8 = 57600 • 9 = 115200
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString

21.6.2.10 RETURN_CODE IDTechSDK.IDT_KioskIII.config_setDEKVariantType (byte *type*, string *ident* = " ")

Set Data Encryption Key Variant Type

This command exists to specify the key variant type of Data encryption Key (Slot = 0), and MUST be used before the initial loading of the Data encryption Key into the device. The key variant type CANNOT be changed once the Data encryption Key is present. It must remain either Data Variant or PIN Variant.

Parameters

<i>type</i>	<ul style="list-style-type: none"> • 0 = Data Variant • 1 = Pin Variant
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString

21.6.2.11 RETURN_CODE IDTechSDK.IDT_KioskIII.config_setDUKPT_DEK_Attribution_AES (byte *keyslot*, byte *workingKey*, byte *keyUsage*, string *ident* = " ")

Set DUKPT DEK Attribution based on KeySlot AES

Command settings can only be changed for each key one time.

Parameters

<i>keyslot</i>	Key Slot
<i>workingKey</i>	<ul style="list-style-type: none"> • 0 = TDES • 1 = TDES3 • 2 = AES-128
<i>keyUsage</i>	<ul style="list-style-type: none"> • 0 = Encrypt/Decrypt • 1 = Encrypt • 2 = Decrypt
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString

21.6.2.12 RETURN_CODE IDTechSDK.IDT_KioskIII.config_setDUKPT_DEK_Attribution_TDES (byte *keyslot*, byte *outputMode*, byte *variant*, string *ident* = " ")

Set DUKPT DEK Attribution based on KeySlot TDES

Command settings can only be changed for each key one time. However switching the Output mode (TDES and TransArmor TDES) is always valid.

Parameters

<i>keyslot</i>	Key Slot
<i>outputMode</i>	<ul style="list-style-type: none"> • 0 = TDES • 1 = AES-128 • 2 = TransArmor TDES
<i>variant</i>	<ul style="list-style-type: none"> • 0 = Data Variant • 1 = Pin Variant
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString

21.6.2.13 RETURN_CODE IDTechSDK.IDT_KioskIII.config_setDUKPTEncryptionType (byte type, string ident = " ")

Set DUKPT Key Encryption Type

This command exists to specify the encryption type of Data encryption Key(Slot = 0), and MUST be used before the initial loading of the Data encryption Key into the device. The encryption type CANNOT be changed once the Data encryption Key is present. It must remain either TDES or AES.

Parameters

<i>type</i>	<ul style="list-style-type: none"> • 0 = TDES • 1 = AES • 2 = TransArmor
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString

21.6.2.14 RETURN_CODE IDTechSDK.IDT_KioskIII.config_setKeySlot_PEK_DEK (bool isPEK, byte keyslot, string ident = " ")

Set KeySlot for DUKPT PEK and DUKPT DEK

• Selects one of the available PEKs for use in subsequent PIN encryption operations. • Selects one of the available DEKs for use in subsequent Data encryption operations.

Parameters

<i>isPEK</i>	TRUE = Set PEK, FALSE = Set DEK
<i>keyslot</i>	The KeySlot number specifies the key this command will select
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString

21.6.2.15 RETURN_CODE IDTechSDK.IDT_KioskIII.config_setRKLKeys (short keyNumber, byte[] tr31, byte[] nonce, byte[] hmac, ref byte[] kv, ref byte[] nonce_device, ref byte[] hmac_device, string ident = " ")

Set RKL Keys

• Sets the RKL Keys.

Parameters

<i>keyNumber</i>	Number of keys remaining to load
<i>tr31</i>	ASN.1 structure of encrypted key(s)

Parameters

<i>nonce</i>	Nonce generated by RKMS to generate an HMAC used for auth.
<i>hmac</i>	HMAC-SHA256 generated from RKMS.
<i>kv</i>	ASN.1 Key Verification Structure returned from device.
<i>nonce_device</i>	Nonce generated by the device to generate a MAC used for auth.
<i>hmac_device</i>	HMAC-SHA256 generated from device terminal.
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with `device_getResponseCodeString`

21.6.2.16 `RETURN_CODE IDTechSDK.IDT_KioskIII.ctls_activateTransaction (int timeout, byte[] tags, bool isFastEMV = false, string ident = " ")`

Start CTLS Transaction Request

Authorizes the CTLS transaction

The tags will be returned in the callback routine.

Parameters

<i>timeout</i>	Timeout value in seconds.
<i>tags</i>	The tags to be included in the request. Passed as TLV Data. Example, tag 9F02 with amount 0x000000000100 would be 0x9F0206000000000100
<i>isFastEMV</i>	If TRUE, it will populate the <code>IDTTransactionData.fastEMV</code> with ASCII data similar to IDTech FastEMV KB output, after performing an auto-authenticate and auto-complete with <code>ResultCode = Could Not Contact Host</code>
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with `errorCode.getErrorString()`

NOTE ON APPLEPAY VAS: To enable ApplePay VAS, first a merchant record must be defined in one of the six available index positions (1-6) using `device_setMerchantRecord`, then container tag FFEE06 must be sent as part of the additional tags parameter of `ctls_startTransaction`. Tag FFEE06 must contain tag 9F26 and 9F22, and can optionally contain tags 9F2B and DFO1. Example FFEE06189F220201009F2604000000009F2B050100000000DF010101 9F22 = two bytes = ApplePay Terminal Application Version Number. Hard defined as 0100 for now. (required) 9F26 = four bytes = ApplePay Terminal Capabilities Information (required)

- Byte 1 = RFU
- Byte 2 = Terminal Type
- - Bit 8 = VAS Support (1=on, 0 = off)
- - Bit 7 = Touch ID Required (1=on, 0 = off)
- - Bit 6 = RFU
- - Bit 5 = RFU
- - Bit 1,2,3,4

- - - 0 = Payment Terminal
- - - 1 = Transit Terminal
- - - 2 = Access Terminal
- - - 3 = Wireless Handoff Terminal
- - - 4 = App Handoff Terminal
- - - 15 = Other Terminal
- Byte 3 = RFU
- Byte 4 = Terminal Mode
- - 0 = ApplePay VAS OR ApplePay
- - 1 = ApplePay VAS AND ApplePay
- - 2 = ApplePay VAS ONLY
- - 3 = ApplePay ONLY 9F2B = 5 bytes = ApplePay VAS Filter. Each byte filters for that specific merchant index (optional) DF01 = 1 byte = ApplePay VAS Protocol. (optional)
- - Bit 1 : 1 = URL VAS, 0 = Full VAS
- - Bit 2 : 1 = VAS Beeps, 0 = No VAS Beeps
- - Bit 3 : 1 = Silent Comm Error, 2 = EMEA Comm Error
- - Bit 4-8 : RFU

21.6.2.17 RETURN_CODE IDTechSDK.IDT_KioskIII.ctls_cancelTransaction (string *ident* = " ")

Cancel EMV Transaction

Cancels the currently executing EMV transaction.

Parameters

<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.
--------------	--

Returns

RETURN_CODE: Values can be parsed with `errorCode.getErrorString()`

21.6.2.18 RETURN_CODE IDTechSDK.IDT_KioskIII.ctls_getAllConfigurationGroups (ref byte *response*[], string *ident* = " ")

Retrieve All Configuration Groups

Returns all the Configuration Groups installed on the terminal for CTLS

Parameters

<i>response</i>	array of CTLS groups as TLV bytes
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with `errorCode.getErrorString()`

21.6.2.19 RETURN_CODE IDTechSDK.IDT_KioskIII.ctls_getConfigurationGroup (int *group*, ref byte[] *tlv*, string *ident* = " ")

Get Configuration Group

Retrieves the Configuration for the specified Group.

Parameters

<i>group</i>	Configuration Group
<i>tlv</i>	return data
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with `errorCode.getErrorString()`

21.6.2.20 RETURN_CODE IDTechSDK.IDT_KioskIII.ctls_nfcCommand (byte[] *nfcCmdPkt*, ref byte[] *response*, string *ident* = " ")

NFC Command

This command uses `nfcCmdPkt[0]` in command data field to implement different functions. This command should be used in Pass-Through mode and command with "Poll for a NFC Tag" data should be used first. Command with other data can only be used once the "Poll for a NFC Tag" command has indicated that a NFC tag is present.

Parameters

<i>nfcCmdPkt</i>	<p>System Code</p> <ul style="list-style-type: none"> • Poll for NFC Tag: nfcCmdPkt[0] = 0xff, nfcCmdPkt[1] = timeout value (in seconds) • Tag1 Static Get All Data: nfcCmdPkt[0] = 0x11 • Tag1 Static Read a Byte: nfcCmdPkt[0] = 0x12, nfcCmdPkt[1] = Address of Data • Tag1 Static Write a Byte: nfcCmdPkt[0] = 0x13 nfcCmdPkt[1] = Address of Data, nfcCmdPkt[2] = Data to be written • Tag1 Static Write a Byte NE: nfcCmdPkt[0] = 0x14, nfcCmdPkt[1] = Address of Data, nfcCmdPkt[2] = Data to be written • Tag1 Dynamic Read a Segment: nfcCmdPkt[0] = 0x15, nfcCmdPkt[1] = Address of Segment • Tag1 Dynamic Read 8 Bytes: nfcCmdPkt[0] = 0x16, nfcCmdPkt[1] = Address of Data • Tag1 Dynamic Write 8 Bytes: nfcCmdPkt[0] = 0x17, nfcCmdPkt[1] = Address of Data, nfcCmdPkt[2]~nfcCmdPkt[9] = Data to be written • Tag1 Dynamic Write 8 Bytes NE: nfcCmdPkt[0] = 0x18, nfcCmdPkt[1] = Address of Data, nfcCmdPkt[2]~nfcCmdPkt[9] = Data to be written • Tag2 Read Data (16 bytes): nfcCmdPkt[0] = 0x21, nfcCmdPkt[1] = Address of Data • Tag2 Write Data (4 bytes): nfcCmdPkt[0] = 0x22, nfcCmdPkt[1] = Address of Data, nfcCmdPkt[2]~nfcCmdPkt[5] = Data to be written • Tag2 Select Sect: nfcCmdPkt[0] = 0x23, nfcCmdPkt[1] = Sect number • Tag3 Read Data: – nfcCmdPkt[0] = 0x41, – nfcCmdPkt[1] = Number of services, value n – nfcCmdPkt[2]~nfcCmdPkt[2n+1]: Service code list – nfcCmdPkt[2n+2]: Number of blocks, value m. – nfcCmdPkt[2n+3....]: Block list, length is 2m~3m • Tag3 Write Data: – nfcCmdPkt[0] = 0x41, – nfcCmdPkt[1] = Number of services, value n – nfcCmdPkt[2]~nfcCmdPkt[2n+1]: Service code list – nfcCmdPkt[2n+2]: Number of blocks, value m. – nfcCmdPkt[2n+3....]: Block list, length is 2m~3m – nfcCmdPkt[...]: Block data, length is 16m • Tag4 Command: nfcCmdPkt[0] = 0x81, nfcCmdPkt[1]~nfcCmdPkt[n]:data
<i>response</i>	Response as explained in FeliCA Lite-S User's Manual
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with `errorCode.getErrorString()`

21.6.2.21 RETURN_CODE IDTechSDK.IDT_KioskIII.ctls_removeAllCAPK (string ident = " ")

Remove All Certificate Authority Public Key

Removes all the CAPK for CTLS

Parameters

<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.
--------------	--

Returns

RETURN_CODE: Values can be parsed with `errorCode.getErrorString()`

21.6.2.22 RETURN_CODE IDTechSDK.IDT_KioskIII.ctls_removeApplicationData (byte[] AID, string ident = " ")

Remove Application Data by AID

Removes the Application Data for CTLS as specified by the AID name passed as a parameter

Parameters

<i>AID</i>	Name of ApplicationID Must be between 5 and 16 bytes
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with `errorCode.getErrorString()`

21.6.2.23 RETURN_CODE IDTechSDK.IDT_KioskIII.ctls_removeCAPK (byte[] capk, string ident = " ")

Remove Certificate Authority Public Key

Removes the CAPK as specified by the RID/Index

Parameters

<i>capk</i>	6 byte CAPK = 5 bytes RID + 1 byte INDEX
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with `errorCode.getErrorString()`

21.6.2.24 RETURN_CODE IDTechSDK.IDT_KioskIII.ctls_removeConfigurationGroup (int group, string ident = " ")

Remove Configuration Group

Removes the Configuration as specified by the Group. Must not by group 0

Parameters

<i>group</i>	Configuration Group
--------------	---------------------

Return values

<i>RETURN_CODE</i>	Return codes listed as typedef enum in IDTCommon:RETURN_CODE. Values can be parsed with <code>IDT_BTPay::device_getResponseCodeString():()</code>
--------------------	---

21.6.2.25 RETURN_CODE IDTechSDK.IDT_KioskIII.ctls_retrieveAIDList (ref byte *response*[[[]], string *ident* = " ")

Retrieve AID list

Returns all the AID names and their assigned groups installed on the terminal for CTLS.

Parameters

<i>response</i>	array of 2-tag TLV data objects: FFE4 (group name) followed by 9F06 (AID)
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with `errorCode.getErrorString()`

21.6.2.26 RETURN_CODE IDTechSDK.IDT_KioskIII.ctls_retrieveApplicationData (byte[] *AID*, ref byte[] *tlv*, string *ident* = " ")

Retrieve Application Data by AID

Retrieves the CTLS Application Data as specified by the AID name passed as a parameter.

Parameters

<i>AID</i>	Name of ApplicationID. Must be between 5 and 16 bytes
<i>tlv</i>	The TLV elements of the requested AID
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with `errorCode.getErrorString()`

21.6.2.27 RETURN_CODE IDTechSDK.IDT_KioskIII.ctls_retrieveCAPK (byte[] *capk*, ref byte[] *key*, string *ident* = " ")

Retrieve Certificate Authority Public Key

Retrieves the CAPK for CTLS as specified by the RID/Index passed as a parameter.

Parameters

<i>capk</i>	6 bytes CAPK = 5 bytes RID + 1 byte Index
-------------	---

Parameters

<i>key</i>	<p>Response returned as a CAKey format: [5 bytes RID][1 byte Index][1 byte Hash Algorithm][1 byte Encryption Algorithm][20 bytes HashValue][4 bytes Public Key Exponent][2 bytes Modulus Length][Variable bytes Modulus] Where:</p> <ul style="list-style-type: none"> • Hash Algorithm: The only algorithm supported is SHA-1. The value is set to 0x01 • Encryption Algorithm: The encryption algorithm in which this key is used. Currently support only one type: RSA. The value is set to 0x01. • HashValue: Which is calculated using SHA-1 over the following fields: RID & Index & Modulus & Exponent • Public Key Exponent: Actually, the real length of the exponent is either one byte or 3 bytes. It can have two values: 3 (Format is 0x00 00 00 03), or 65537 (Format is 0x00 01 00 01) • Modulus Length: LenL LenH Indicated the length of the next field. • Modulus: This is the modulus field of the public key. Its length is specified in the field above.
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with `errorCode.getErrorString()`

21.6.2.28 RETURN_CODE IDTechSDK.IDT_KioskIII.ctls_retrieveCAPKList (ref byte[] keys, string ident = " ")

Retrieve the Certificate Authority Public Key list

Returns all the CAPK RID and Index installed on the terminal for CTLS.

Parameters

<i>keys</i>	[key1][key2]...[keyn], each key 6 bytes where key = 5 bytes RID + 1 byte index
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with `errorCode.getErrorString()`

21.6.2.29 RETURN_CODE IDTechSDK.IDT_KioskIII.ctls_retrieveTerminalData (ref byte[] tlv, string ident = " ")

Retrieve Terminal Data

Retrieves the Terminal Data for CTLS. This is configuration group 0 (Tag FFEE - > FFEE0100). The terminal data can also be retrieved by `ctls_getConfigurationGroup(0)`.

Parameters

<i>tlv</i>	Response returned as a TLV
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with `errorCode.getErrorString()`

21.6.2.30 RETURN_CODE IDTechSDK.IDT_KioskIII.ctls_setApplicationData (byte[] tlv, string ident = " ")

Set Application Data by AID

Sets the Application Data for CTLS as specified by TLV data

Parameters

<i>tlv</i>	Application data in TLV format The first tag of the TLV data must be the group number (FFE4). The second tag of the TLV data must be the AID (9F06)
------------	---

Example valid TLV, for Group #2, AID a0000000035010: "ffe401029f0607a0000000051010ffe10101ffe50110ffe30114ffe20106"

Parameters

<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.
--------------	--

Returns

RETURN_CODE: Values can be parsed with `errorCode.getErrorString()`

21.6.2.31 RETURN_CODE IDTechSDK.IDT_KioskIII.ctls_setCAPK (byte[] key, string ident = " ")

Set Certificate Authority Public Key

Sets the CAPK for CTLS as specified by the CAKey structure

Parameters

<i>key</i>	CAKey format: [5 bytes RID][1 byte Index][1 byte Hash Algorithm][1 byte Encryption Algorithm][20 bytes HashValue][4 bytes Public Key Exponent][2 bytes Modulus Length][Variable bytes Modulus] Where: <ul style="list-style-type: none"> • Hash Algorithm: The only algorithm supported is SHA-1. The value is set to 0x01 • Encryption Algorithm: The encryption algorithm in which this key is used. Currently support only one type: RSA. The value is set to 0x01. • HashValue: Which is calculated using SHA-1 over the following fields: RID & Index & Modulus & Exponent • Public Key Exponent: Actually, the real length of the exponent is either one byte or 3 bytes. It can have two values: 3 (Format is 0x00 00 00 03), or 65537 (Format is 0x00 01 00 01) • Modulus Length: LenL LenH Indicated the length of the next field. • Modulus: This is the modulus field of the public key. Its length is specified in the field above.
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with `errorCode.getErrorString()`

21.6.2.32 RETURN_CODE IDTechSDK.IDT_KioskIII.ctls_setConfigurationGroup (byte[] *tlv*, string *ident* = " ")

Set Configuration Group

Sets the Configuration Group for CTLS as specified by the TLV data

Parameters

<i>tlv</i>	Configuration Group Data in TLV format The first tag of the TLV data must be the group number (FFE4). A second tag must exist
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with `errorCode.getErrorString()`

21.6.2.33 RETURN_CODE IDTechSDK.IDT_KioskIII.ctls_setDefaultConfiguration (string *ident* = " ")

Set Default Configuration Group

Resets the device to default CTLS configuration group settings

Parameters

<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.
--------------	--

Returns

RETURN_CODE: Values can be parsed with `errorCode.getErrorString()`

21.6.2.34 RETURN_CODE IDTechSDK.IDT_KioskIII.ctls_setTerminalData (byte[] *tlv*, string *ident* = " ")

Set Terminal Data

Sets the Terminal Data for CTLS as specified by the TLV. The first TLV must be Configuration Group Number (Tag FFE4). The terminal global data is group 0, so the first TLV would be FFE40100. Other groups can be defined using this method (1 or greater), and those can be retrieved with `emv_getConfigurationGroup(int group)`, and deleted with `emv_removeConfigurationGroup(int group)`. You cannot delete group 0.

Parameters

<i>tlv</i>	TerminalData configuration data
------------	---------------------------------

Return values

<i>RETURN_CODE</i>	Return codes listed as typedef enum in IDTCommon:RETURN_CODE. Values can be parsed with <code>IDT_BTPay::device_getResponseCodeString():()</code>
--------------------	---

21.6.2.35 RETURN_CODE IDTechSDK.IDT_KioskIII.ctls_startTransaction (double *amount*, double *amtOther*, int *exponent*, int *type*, int *timeout*, byte[] *tags*, bool *isFastEMV* = false, string *ident* = " ")

Start CTLS Transaction Request

The tags will be returned in the callback routine.

Parameters

<i>amount</i>	Transaction amount value (tag value 9F02)
<i>amtOther</i>	Other amount value, if any (tag value 9F03)
<i>type</i>	Transaction type (tag value 9C).
<i>timeout</i>	Timeout value in seconds.
<i>tags</i>	Any other tags to be included in the request. Passed as TLV data. Example, tag 9F02 with amount 0x000000000100 would be 0x9F0206000000000100 If tags 9F02 (amount), 9F03 (other amount), or 9C (transaction type) are included, they will take priority over these values supplied as individual parameters to this method.
<i>isFastEMV</i>	If TRUE, it will populate the IDTTransactionData.fastEMV with ASCII data similar to IDTech FastEMV KB output, after performing an auto-authenticate and auto-complete with ResultCode = Could Not Contact Host
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with `errorCode.getErrorString()`

NOTE ON APPLEPAY VAS: To enable ApplePay VAS, first a merchant record must be defined in one of the six available index positions (1-6) using `device_setMerchantRecord`, then container tag FFEE06 must be sent as part of the additional tags parameter of `ctls_startTransaction`. Tag FFEE06 must contain tag 9F26 and 9F22, and can optionally contain tags 9F2B and DFO1. Example FFEE06189F220201009F2604000000009F2B050100000000↵DF010101 9F22 = two bytes = ApplePay Terminal Application Version Number. Hard defined as 0100 for now. (required) 9F26 = four bytes = ApplePay Terminal Capabilities Information (required)

- Byte 1 = RFU
- Byte 2 = Terminal Type
 - - Bit 8 = VAS Support (1=on, 0 = off)
 - - Bit 7 = Touch ID Required (1=on, 0 = off)
 - - Bit 6 = RFU
 - - Bit 5 = RFU
 - - Bit 1,2,3,4
 - - - 0 = Payment Terminal
 - - - 1 = Transit Terminal
 - - - 2 = Access Terminal
 - - - 3 = Wireless Handoff Terminal
 - - - 4 = App Handoff Terminal
 - - - 15 = Other Terminal
- Byte 3 = RFU
- Byte 4 = Terminal Mode
 - - 0 = ApplePay VAS OR ApplePay
 - - 1 = ApplePay VAS AND ApplePay

- - 2 = ApplePay VAS ONLY
- - 3 = ApplePay ONLY 9F2B = 5 bytes = ApplePay VAS Filter. Each byte filters for that specific merchant index (optional) DF01 = 1 byte = ApplePay VAS Protocol. (optional)
- - Bit 1 : 1 = URL VAS, 0 = Full VAS
- - Bit 2 : 1 = VAS Beeps, 0 = No VAS Beeps
- - Bit 3 : 1 = Silent Comm Error, 2 = EMEA Comm Error
- - Bit 4-8 : RFU

21.6.2.36 RETURN_CODE IDTechSDK.IDT_KioskIII.ctls_trySetTerminalData (byte[] *tlv*, ref byte[] *rejectedTLV*, ref byte[] *convertedTLV*, string *ident* = " ")

Try to Set CTLS Terminal Data

Attempts to set the CTLS Terminal Data. Use this functions to attempt to set the terminal data to the device. This function will allow the cross-device tag compatible writing of IDTech Legacy/New tag definitions

Parameters

<i>tlv</i>	TerminalData TLV Data
<i>rejectedTLV</i>	Contains the tags (if any) that were rejected by the firmware
<i>convertedTLV</i>	Contains the tags (if any) that were converted and accepted by the firmware
<i>ident</i>	Optional identifier

Return values

RETURN_CODE	Return codes listed as typedef enum in IDTCommon:RETURN_CODE. Values can be parsed with IDT_BTPay::device_getResponseCodeString:()
--------------------	--

21.6.2.37 RETURN_CODE IDTechSDK.IDT_KioskIII.ctls_updateBalance (byte *statusCode*, byte[] *authCode*, byte[] *date*, byte[] *time*, string *ident* = " ")

Update Balance

This command is the authorization response sent by the issuer to the terminal including the Authorization Status (OK or NOT OK).

This command is also being used in some implementations (i.e. EMEA) to communicate the results of Issuer Authentication to the reader in order to display the correct LCD messages. With this command, the POS passes the authorization result (OK/NOT OK), and possibly the Authorization Code (Auth_Code)/Date/Time to the terminal.

Parameters

<i>statusCode</i>	00: OK, 01: NOT OK, 02: (ARC response 89 for Interac)
<i>authCode</i>	Authorization code from host. Six bytes. Optional
<i>date</i>	Transaction date. If null, uses current terminal date. 3 bytes compressed numeric YYMMDD (tag value 9A).
<i>time</i>	Transaction time. If null, uses current terminal time. 3 bytes compressed numeric HHMMSS (tag value 9F21).
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString

21.6.2.38 RETURN_CODE IDTechSDK.IDT_KioskIII.device_activateTransaction (int *timeout*, byte[] *tags*, bool *isFastEMV* = false, string *ident* = " ")

Start CTLS Transaction Request

Authorizes the CTLS transaction

The tags will be returned in the callback routine.

Parameters

<i>timeout</i>	Timeout value in seconds.
<i>tags</i>	The tags to be included in the request. Passed as TLV Data. Example, tag 9F02 with amount 0x0000000000100 would be 0x9F0206000000000100
<i>isFastEMV</i>	If TRUE, it will populate the IDTTransactionData.fastEMV with ASCII data similar to IDTech FastEMV KB output, after performing an auto-authenticate and auto-complete with ResultCode = Could Not Contact Host
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with errorCode.getErrorString()

NOTE ON APPLEPAY VAS: To enable ApplePay VAS, first a merchant record must be defined in one of the six available index positions (1-6) using device_setMerchantRecord, then container tag FFEE06 must be sent as part of the additional tags parameter of ctls_startTransaction. Tag FFEE06 must contain tag 9F26 and 9F22, and can optionally contain tags 9F2B and DFO1. Example FFEE06189F220201009F2604000000009F2B050100000000↵DF010101 9F22 = two bytes = ApplePay Terminal Applicaiton Version Number. Hard defined as 0100 for now. (required) 9F26 = four bytes = ApplePay Terminal Capabilities Information (required)

- Byte 1 = RFU
- Byte 2 = Terminal Type
- - Bit 8 = VAS Support (1=on, 0 = off)
- - Bit 7 = Touch ID Required (1=on, 0 = off)
- - Bit 6 = RFU
- - Bit 5 = RFU
- - Bit 1,2,3,4
- - - 0 = Payment Terminal
- - - 1 = Transit Terminal
- - - 2 = Access Terminal
- - - 3 = Wireless Handoff Terminal
- - - 4 = App Handoff Terminal
- - - 15 = Other Terminal
- Byte 3 = RFU

- Byte 4 = Terminal Mode
- - 0 = ApplePay VAS OR ApplePay
- - 1 = ApplePay VAS AND ApplePay
- - 2 = ApplePay VAS ONLY
- - 3 = ApplePay ONLY 9F2B = 5 bytes = ApplePay VAS Filter. Each byte filters for that specific merchant index (optional) DF01 = 1 byte = ApplePay VAS Protocol. (optional)
- - Bit 1 : 1 = URL VAS, 0 = Full VAS
- - Bit 2 : 1 = VAS Beeps, 0 = No VAS Beeps
- - Bit 3 : 1 = Silent Comm Error, 2 = EMEA Comm Error
- - Bit 4-8 : RFU

21.6.2.39 RETURN_CODE IDTechSDK.IDT_KioskIII.device_cancelTransaction (string *ident* = " ")

Cancel Transaction

Cancels the currently executing device transaction.

Parameters

<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.
--------------	--

Returns

RETURN_CODE: Values can be parsed with `errorCode.getErrorString()`

21.6.2.40 RETURN_CODE IDTechSDK.IDT_KioskIII.device_controlLED (byte *indexLED*, byte *control*, string *ident* = " ")

Control LED

Controls the LED for the reader. This command will only operate in pass-through mode

Parameters

<i>indexLED</i>	For LED <ul style="list-style-type: none"> • 00: LED 0 (Power LED, leftmost LED) • 01: LED 1 • 02: LED 2 • 03: LED 3 • FF: All LED's
<i>control</i>	LED Status: <ul style="list-style-type: none"> • 00: LED Off • 01: LED On
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString

21.6.2.41 RETURN_CODE IDTechSDK.IDT_KioskIII.device_controlUserInterface (byte[] values, string ident = " ")

Control User Interface

Controls the User Interface: Display, Beep, LED

```
@param values Four bytes to control the user interface
Byte[0] = LCD Message
Messages 00-07 are normally controlled by the reader.
- 00h: Idle Message (Welcome)
- 01h: Present card (Please Present Card)
- 02h: Time Out or Transaction cancel (No Card)
- 03h: Transaction between reader and card is in the middle (Processing...)
- 04h: Transaction Pass (Thank You)
- 05h: Transaction Fail (Fail)
- 06h: Amount (Amount $ 0.00 Tap Card)
- 07h: Balance or Offline Available funds (Balance $ 0.00) Messages 08-0B are controlled by the terminal
- 08h: Insert or Swipe card (Use Chip & PIN)
- 09h: Try Again(Tap Again)
- 0Ah: Tells the customer to present only one card (Present 1 card only)
- 0Bh: Tells the customer to wait for authentication/authorization (Wait)
- FFh: indicates the command is setting the LED/Buzzer only.
Byte[1] = Beep Indicator
- 00h: No beep
- 01h: Single beep
- 02h: Double beep
- 03h: Three short beeps
- 04h: Four short beeps
- 05h: One long beep of 200 ms
- 06h: One long beep of 400 ms
- 07h: One long beep of 600 ms
- 08h: One long beep of 800 ms
Byte[2] = LED Number
- 00h: LED 0 (Power LED) 01h: LED 1
- 02h: LED 2
- 03h: LED 3
- FFh: All LEDs
Byte[3] = LED Status
- 00h: LED Off
- 01h: LED On
```

Parameters

<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.
--------------	--

Returns

RETURN_CODE: Values can be parsed with errorCode.getErrorString()

21.6.2.42 RETURN_CODE IDTechSDK.IDT_KioskIII.device_enablePassThrough (bool enablePassThrough, string ident = " ")

Enable Pass Through

Enables Pass Through Mode for direct communication with L1 interface (power on icc, send apdu, etc).

Parameters

<i>enablePassThrough</i>	true = pass through ON, false = pass through OFF
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with `errorCode.getErrorString()`

21.6.2.43 RETURN_CODE IDTechSDK.IDT_KioskIII.device_getAnyRKIStatus (bool *isMultiKey*, ref string *status*, ref Dictionary< string, RKI_KEY_TYPE > *keys*, string *ident* = " ")

Get RKI Status Polls the RKI servers to validate device RKI eligibility. Note: if device type is known in advance (production or demo device), it is more efficient to use `device_getRKIStatus` instead

Parameters

<i>isMultiKey</i>	True = MultiKey, False = SingleKey
<i>status</i>	RKI status returned as a string
<i>keys</i>	Returns all available keys
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with `errorCode.getErrorString()`. When no data is available, return code = RETURN_CODE_NO_DATA_AVAILABLE

21.6.2.44 RETURN_CODE IDTechSDK.IDT_KioskIII.device_getConfigurationFromMemory (ref string *json*, string *ident* = " ")

Read Device Configuration From Memory Retrieves the ViVOconfig configuration data from memory after a successful `device_readConfigurationToMemory`

Parameters

<i>json</i>	Returns the .json file data that can be saved to disk
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with `errorCode.getErrorString()`. When no data is available, return code = RETURN_CODE_NO_DATA_AVAILABLE

21.6.2.45 RETURN_CODE IDTechSDK.IDT_KioskIII.device_getFirmwareVersion (ref string *response*, string *ident* = " ")

Polls device for Firmware Version

Parameters

<i>response</i>	Response returned of Firmware Version
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with `errorCode.getErrorString()`

21.6.2.46 RETURN_CODE IDTechSDK.IDT_KioskIII.device_getMerchantRecord (int *index*, ref byte[] *record*, string *ident* = " ")

Get Merchant Record Gets the merchant record for ApplePay VAS

Parameters

<i>index</i>	Merchant Record index, valid values 1-6
<i>record</i>	Data returned containing 99 bytes: Byte 0 = Merchant Index Byte 1 = Merchant Enabled (1 = enabled) Byte 2 - 33 = Merchant Protocol Hash-256 value Byte 34 = Length of Merchant URL Bytes 35 - 99 = URL
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with `errorCode.getErrorString()`

21.6.2.47 RETURN_CODE IDTechSDK.IDT_KioskIII.device_getRKIStatus (bool *isProd*, bool *isMultiKey*, ref string *status*, ref Dictionary< string, RKI_KEY_TYPE > *keys*, string *ident* = " ")

Get RKI Status From Specified Server Polls the RKI server to validate device RKI eligibility

Parameters

<i>isProd</i>	TRUE = poll production RKI server, FALSE = poll demo RKI server
<i>isMultiKey</i>	True = MultiKey, False = SingleKey
<i>status</i>	RKI status returned as a string
<i>keys</i>	Returns dictionary of available keys
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with `errorCode.getErrorString()`. When no data is available, return code = RETURN_CODE_NO_DATA_AVAILABLE

21.6.2.48 RETURN_CODE IDTechSDK.IDT_KioskIII.device_getTransactionResults (ref IDTTransactionData *results*, string *ident* = " ")

Get Transaction Results Gets the transaction results when the reader is functioning in "Auto Poll" mode

Parameters

<i>results</i>	The transaction results
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with `errorCode.getErrorString()`. When no data is available, return code = RETURN_CODE_NO_DATA_AVAILABLE

21.6.2.49 RETURN_CODE IDTechSDK.IDT_KioskIII.device_pingDevice (string *ident* = " ")

Ping Device

Pings the reader. If connected, returns success. Otherwise, returns timeout.

Parameters

<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.
--------------	--

Returns

RETURN_CODE: Values can be parsed with `errorCode.getErrorString()`

21.6.2.50 RETURN_CODE IDTechSDK.IDT_KioskIII.device_pollForToken (byte *seconds*, ref byte *card*, ref byte[] *serialNumber*, string *ident* = " ")

Poll for Token

Once Pass-Through Mode is started, ViVOpay will not poll for any cards until the "Poll for Token" command is received. This command tells ViVOpay to start polling for a Type A or Type B PICC until a PICC is detected or a timeout occurs.

This command automatically turns the RF Antenna on.

If a PICC is detected within the specified time limit, ViVOpay activates it and responds back to the terminal with card related data such as the Serial Number. If no PICC is detected within the specified time limit, ViVOpay stops polling and responds back indicating that no card was found. No card related data is returned in this case

Parameters

<i>timeout</i>	Timeout, in seconds to wait for card to be detected
<i>card</i>	Card Type: <ul style="list-style-type: none"> • 00h None (Card Not Detected or Could not Activate) • 01h ISO 14443 Type A (Supports ISO 14443-4 Protocol) • 02h ISO 14443 Type B (Supports ISO 14443-4 Protocol) • 03h Mifare Type A (Standard) • 04h Mifare Type A (Ultralight) • 05h ISO 14443 Type A (Does not support ISO 14443-4 Protocol) • 06h ISO 14443 Type B (Does not support ISO 14443-4 Protocol) • 07h ISO 14443 Type A and Mifare (NFC phone)
<i>serialNumber</i>	Serial Number or the UID of the PICC
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with `device_getResponseCodeString`

21.6.2.51 **RETURN_CODE** IDTechSDK.IDT_KioskIII.device_readConfigurationToMemory (string *memo*, Configs.InstallRules *rules*, List< Configs.DeviceCommand > *cmds*, string *ident* = " ", bool *isForeground* = false)

Read Device Configuration To Memory Executes a ViVOconfig read of device settings and stores the results in memory

Once a ViVOconfig read starts, all messages will be returned to the callback as DeviceState.ViVOconfig. The ViVOconfig operation is complete when the return code from a DeviceState.ViVOconfig message is one of the following values:

- RETURN_CODE_SDK_VIVOCONFIG_READ_CONFIG_SUCCESS = Read process completed successfully. Execute device_getConfigurationFromMemory to retrieve the configuration data
- RETURN_CODE_SDK_VIVOCONFIG_READ_CONFIG_FAILED = Read process FAILED.

Parameters

<i>memo</i>	Optional: populates the memo field of the captured .json file
<i>rules</i>	Optional: populates the .json file with available install rules
<i>cmds</i>	Optional: populates the .json file with commands to execute
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.
<i>isForeground</i>	If TRUE, will block program until update complete. Otherwise, FALSE performs update on background.

Returns

RETURN_CODE: Values can be parsed with errorCode.getErrorString(). When no data is available, return code = RETURN_CODE_NO_DATA_AVAILABLE

21.6.2.52 **RETURN_CODE** IDTechSDK.IDT_KioskIII.device_rebootDevice (string *ident* = " ")

Reboot Device

Performs a reboot of the device

Parameters

<i>ip</i>	Optional: The callback function will only be applicable to the provided IP.
-----------	---

21.6.2.53 **RETURN_CODE** IDTechSDK.IDT_KioskIII.device_RemoteKeyInjection (RKI_KEY_TYPE *type*, string *keyName*, string *ident* = " ", bool *performOnForeground* = false)

Remote Key Injection Performs a remote key injection for the device

Parameters

<i>type</i>	Remote Key Injection Type
<i>keyName</i>	Name of key (optional)
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.
<i>performOnForeground</i>	TRUE = block during process (default is false)

Returns

RETURN_CODE: Values can be parsed with `errorCode.getErrorString()`. When no data is available, return code = RETURN_CODE_NO_DATA_AVAILABLE

21.6.2.54 RETURN_CODE IDTechSDK.IDT_KioskIII.device_retrieveAIDList (ref byte response[][], string ident = " ")

Retrieve AID list

Returns all the AID names and their assigned groups installed on the terminal for CTLS/CONTACT.

Parameters

<i>response</i>	array of TLV data objects: FFE4 (group name) followed by 9F06 (AID), and DFEE4F (Interface 01 = CTLS, 02 = CONTACT)
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with `errorCode.getErrorString()`

21.6.2.55 RETURN_CODE IDTechSDK.IDT_KioskIII.device_sendConfiguration (string filename, VC_OPERATION_TYPE type, bool matchFW, string ident = " ", bool isForeground = false)

Send Configuration Executes a ViVOconfig update or verify to a device

Once a ViVOconfig write or verify starts, all messages will be returned to the callback as DeviceState.ViVOconfig. The ViVOconfig operation is complete when the return code from a DeviceState.ViVOconfig message is one of the following values:

- RETURN_CODE_SDK_VIVOCONFIG_VERIFY_SUCCESS = Verification process completed successfully. No differences found.
- RETURN_CODE_SDK_VIVOCONFIG_VERIFY_SUCCESS_WITH_WARNING = Verification process completed with warnings.
- RETURN_CODE_SDK_VIVOCONFIG_VERIFY_FAILED = Verification process FAILED
- RETURN_CODE_SDK_VIVOCONFIG_WRITE_SUCCESS = Write process completed successfully.
- RETURN_CODE_SDK_VIVOCONFIG_WRITE_SUCCESS_WITH_WARNING = Write process completed with warnings
- RETURN_CODE_SDK_VIVOCONFIG_WRITE_FAILED = Write process FAILED

Parameters

<i>filename</i>	The .json configuration file to update the device with NOTE: You can also use this parameter to pass the path of where the .json file is located at on local storage instead
<i>filename</i>	The name of the .json configuration file
<i>type</i>	VC_OPERATION_TYPE

- VC_OPERATION_TYPE_WRITE = Write File To Device, Hash must validate
- VC_OPERATION_TYPE_VERIFY = Verify Device With File
- VC_OPERATION_TYPE_WRITE_IGNORE_HASH = Write File To Device, Ignore Hash

- VC_OPERATION_TYPE_WRITE_FIX_HASH = Write File To Device, Fix Hash If Necessary

Parameters

<i>matchFW</i>	TRUE = Device FW must match file FW, FALSE = Don't validate device FW against file
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.
<i>isForeground</i>	If TRUE, will block program until update complete. Otherwise, FALSE performs update on background.

Returns

RETURN_CODE: Values can be parsed with `errorCode.getErrorString()`. When no data is available, return code = RETURN_CODE_NO_DATA_AVAILABLE

21.6.2.56 RETURN_CODE IDTechSDK.IDT_KioskIII.device_sendConfigurationFromZip (byte[] zip, string filename, VC_OPERATION_TYPE type, bool matchFW, string ident = "", bool isForeground = false)

Send Configuration From Zip Executes a ViVOconfig update or verify to a device using files from a .zip archive

Once a ViVOconfig write or update starts, all messages will be returned to the callback as DeviceState.ViVOconfig. The ViVOconfig operation is complete when the return code from a DeviceState.ViVOconfig message is one of the following values:

- RETURN_CODE_SDK_VIVOCONFIG_VERIFY_SUCCESS = Verification process completed successfully. No differences found.
- RETURN_CODE_SDK_VIVOCONFIG_VERIFY_SUCCESS_WITH_WARNING = Verification process completed with warnings.
- RETURN_CODE_SDK_VIVOCONFIG_VERIFY_FAILED = Verification process FAILED
- RETURN_CODE_SDK_VIVOCONFIG_WRITE_SUCCESS = Write process completed successfully.
- RETURN_CODE_SDK_VIVOCONFIG_WRITE_SUCCESS_WITH_WARNING = Write process completed with warnings
- RETURN_CODE_SDK_VIVOCONFIG_WRITE_FAILED = Write process FAILED

Parameters

<i>zip</i>	.Zip archive containing the .json configuration file and also any associated image assets NOTE: You can also use this parameter to pass the path of where the .zip file is located at on local storage instead
<i>filename</i>	The name of the .json configuration file
<i>type</i>	VC_OPERATION_TYPE

- VC_OPERATION_TYPE_WRITE = Write File To Device, Hash must validate
- VC_OPERATION_TYPE_VERIFY = Verify Device With File
- VC_OPERATION_TYPE_WRITE_IGNORE_HASH = Write File To Device, Ignore Hash
- VC_OPERATION_TYPE_WRITE_FIX_HASH = Write File To Device, Fix Hash If Necessary

Parameters

<i>matchFW</i>	TRUE = Device FW must match file FW, FALSE = Don't validate device FW against file
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Parameters

<i>isForeground</i>	If TRUE, will block program until update complete. Otherwise, FALSE performs update on background.
---------------------	--

Returns

RETURN_CODE: Values can be parsed with `errorCode.getErrorString()`. When no data is available, return code = RETURN_CODE_NO_DATA_AVAILABLE

21.6.2.57 RETURN_CODE IDTechSDK.IDT_KioskIII.device_sendDataCommand (string *cmd*, bool *calcLRC*, ref byte[] *response*, string *ident* = " ")

Send a data command to the device

Sends a command to the device.

Parameters

<i>cmd</i>	String representation of command to execute
<i>calcLRC</i>	If TRUE, this will wrap command with start/length/lrc/sum/end: '{STX}{Len_Low}{Len_High} data {CheckLRC} {CheckSUM} {ETX}'
<i>response</i>	Response data
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with `device_getResponseCodeString`

21.6.2.58 RETURN_CODE IDTechSDK.IDT_KioskIII.device_sendDataCommand_ext (string *cmd*, bool *calcLRC*, ref byte[] *response*, int *timeout*, bool *noResponse*, string *ident* = " ")

Send a data command to the device - extended

Sends a command to the device.

Parameters

<i>cmd</i>	String representation of command to execute
<i>calcLRC</i>	If TRUE, this will wrap command as NGA with start/length/lrc/sum/end: '{STX}{Len_Low}{Len_High} data {CheckLRC} {CheckSUM} {ETX}'
<i>response</i>	Response data
<i>timeout</i>	Timeout value waiting for response, in milliseconds (1000 = 1 second)
<i>noResponse</i>	if TRUE, this will not wait for a response and immediately return SUCCESS
<i>calcITP</i>	If TRUE, this will wrap command as ITP with start/end/lrc: '{STX} data {ETX}{CheckLRC}'
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with `device_getResponseCodeString`

21.6.2.59 `RETURN_CODE IDTechSDK.IDT_KioskIII.device_sendPAE (string command, ref string response, int timeout, string ident = " ")`

Send Payment Application Engine Command

Executes a PAE command

Parameters

<i>command</i>	ASCII command string, should start with "*PAE"
<i>response</i>	command response
<i>timeout</i>	timeout waiting for PAE response
<i>ip</i>	Optional IP address when connected via TCP/IP
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with `errorCode.getErrorString()`

21.6.2.60 `RETURN_CODE IDTechSDK.IDT_KioskIII.device_sendVivoCommandP2 (byte command, byte subCommand, byte[] data, ref byte[] response, string ident = " ")`

Send Vivo Command Protocol 2

Sends a protocol 2 command to Vivo readers (IDG/NEO)

Parameters

<i>command</i>	Command
<i>subCommand</i>	Sub-Command
<i>data</i>	Data. May be null
<i>response</i>	Response
<i>ip</i>	Optional IP
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with `errorCode.getErrorString()`

21.6.2.61 `RETURN_CODE IDTechSDK.IDT_KioskIII.device_sendVivoCommandP2_ext (byte command, byte subCommand, byte[] data, ref byte[] response, int timeout, bool noResponse, string ident = " ")`

Send Vivo Command Protocol 2 Extended

Sends a protocol 2 command to Vivo readers (IDG/NEO)

Parameters

<i>command</i>	Command
<i>subCommand</i>	Sub-Command
<i>data</i>	Data. May be null
<i>response</i>	Response
<i>timeout</i>	Timeout, in milliseconds (3000 = 3 seconds)

Parameters

<i>noResponse</i>	TRUE = don't wait for response, FALSE = wait for response defined by timeout
<i>ip</i>	Optional IP
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with `errorCode.getErrorString()`

21.6.2.62 RETURN_CODE IDTechSDK.IDT_KioskIII.device_setBurstMode (byte *mode*, string *ident* = " ")

Send Burst Mode

Sets the burst mode for the device.

Parameters

<i>mode</i>	0 = OFF, 1 = Always On, 2 = Auto Exit
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with `errorCode.getErrorString()`

21.6.2.63 RETURN_CODE IDTechSDK.IDT_KioskIII.device_setMerchantRecord (int *index*, bool *enabled*, string *merchantID*, string *merchantURL*, string *ident* = " ")

Set Merchant Record Sets the merchant record for ApplePay VAS

Parameters

<i>index</i>	Merchant Record index, valid values 1-6
<i>enabled</i>	Merchant Enabled/Valid flag
<i>merchantID</i>	Merchant unique identifier registered with Apple. Example com.idtechproducts.applePay
<i>merchantURL</i>	Merchant URL, when applicable
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with `errorCode.getErrorString()`

21.6.2.64 RETURN_CODE IDTechSDK.IDT_KioskIII.device_setPollMode (byte *mode*, string *ident* = " ")

Send Poll Mode

Sets the poll mode for the device. Auto Poll keeps reader active, Poll On Demand only polls when requested by terminal

Parameters

<i>mode</i>	0 = Auto Poll, 1 = Poll On Demand
-------------	-----------------------------------

Parameters

<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.
--------------	--

Returns

RETURN_CODE: Values can be parsed with `errorCode.getErrorString()`

21.6.2.65 RETURN_CODE IDTechSDK.IDT_KioskIII.device_startRKI (string *ident* = " ")

Start Legacy Remote Key Injection

Starts a remote key injection request with IDTech RKI Legacy servers.

Parameters

<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.
--------------	--

Returns

RETURN_CODE: Values can be parsed with `device_getResponseCodeString`

21.6.2.66 RETURN_CODE IDTechSDK.IDT_KioskIII.device_StartRKI (int *type*, string *ident* = " ")

Start Remote Key Injection

Starts a remote key injection request with IDTech RKI servers. Set/Get RKI url with `IDT_Device.RKI_URL`.

Parameters

<i>type</i>	0 = Symmetric RKI Demo Unit 1 = Symmetric RKI Production Unit 2 = PKI-RKI Demo Unit 3 = PKI-RKI Production Unit
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with `device_getResponseCodeString`

21.6.2.67 RETURN_CODE IDTechSDK.IDT_KioskIII.device_startTransaction (double *amount*, double *amtOther*, int *exponent*, int *type*, int *timeout*, byte[] *tags*, bool *isFastEMV* = false, string *ident* = " ")

Start CTLS Transaction Request

The tags will be returned in the callback routine.

Parameters

<i>amount</i>	Transaction amount value (tag value 9F02)
<i>amtOther</i>	Other amount value, if any (tag value 9F03)
<i>type</i>	Transaction type (tag value 9C).
<i>timeout</i>	Timeout value in seconds.

Parameters

<i>tags</i>	Any other tags to be included in the request. Passed as TLV data. Example, tag 9F02 with amount 0x000000000100 would be 0x9F0206000000000100 If tags 9F02 (amount), 9F03 (other amount), or 9C (transaction type) are included, they will take priority over these values supplied as individual parameters to this method.
<i>isFastEMV</i>	If TRUE, it will populate the IDTTransactionData.fastEMV with ASCII data similar to IDTech FastEMV KB output, after performing an auto-authenticate and auto-complete with ResultCode = Could Not Contact Host
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with `errorCode.getErrorString()`

NOTE ON APPLEPAY VAS: To enable ApplePay VAS, first a merchant record must be defined in one of the six available index positions (1-6) using `device_setMerchantRecord`, then container tag FFEE06 must be sent as part of the additional tags parameter of `ctls_startTransaction`. Tag FFEE06 must contain tag 9F26 and 9F22, and can optionally contain tags 9F2B and DF01. Example FFEE06189F220201009F2604000000009F2B050100000000DF010101 9F22 = two bytes = ApplePay Terminal Application Version Number. Hard defined as 0100 for now. (required) 9F26 = four bytes = ApplePay Terminal Capabilities Information (required)

- Byte 1 = RFU
- Byte 2 = Terminal Type
 - - Bit 8 = VAS Support (1=on, 0 = off)
 - - Bit 7 = Touch ID Required (1=on, 0 = off)
 - - Bit 6 = RFU
 - - Bit 5 = RFU
 - - Bit 1,2,3,4
 - - - 0 = Payment Terminal
 - - - 1 = Transit Terminal
 - - - 2 = Access Terminal
 - - - 3 = Wireless Handoff Terminal
 - - - 4 = App Handoff Terminal
 - - - 15 = Other Terminal
- Byte 3 = RFU
- Byte 4 = Terminal Mode
 - - 0 = ApplePay VAS OR ApplePay
 - - 1 = ApplePay VAS AND ApplePay
 - - 2 = ApplePay VAS ONLY
 - - 3 = ApplePay ONLY 9F2B = 5 bytes = ApplePay VAS Filter. Each byte filters for that specific merchant index (optional) DF01 = 1 byte = ApplePay VAS Protocol. (optional)
 - - Bit 1 : 1 = URL VAS, 0 = Full VAS
 - - Bit 2 : 1 = VAS Beeps, 0 = No VAS Beeps
 - - Bit 3 : 1 = Silent Comm Error, 2 = EMEA Comm Error
 - - Bit 4-8 : RFU

21.6.2.68 `RETURN_CODE IDTechSDK.IDT_KioskIII.device_updateDeviceFirmware (byte[] firmwareData, string ident = " ")`

Update Firmware

Updates the firmware .

Parameters

<i>firmwareData</i>	Signed binary data of a firmware file provided by IDTech
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

`RETURN_CODE`: Values can be parsed with `errorCode.getErrorString()`

After you pass the `firmwareData` file, a new thread will start to execute the firmware download. You will receive status of the progress through callbacks to the `IDTechSDK.Callback()` delegate. The following parameters will be passed back:

- `sender = IDT_DEVICE_Types.IDT_DEVICE_AUGUSTA`
- `state = DeviceState.FirmwareUpdate`
- `transactionResultCode = status of the firmware update (starting, entering bootloader, applying update, block success, firmware success)`
- `data = File Progress`. Four bytes, with bytes `[0][1]` = current block, and bytes `[2][3]` = total blocks. `0x00030010` = block 3 of 16

Example code starting a firmware update

```
OpenFileDialog diag = new OpenFileDialog(ident);
diag.Filter = "NGA FW Files|*.fm";
if (diag.ShowDialog() == DialogResult.OK)
{
    byte[] file = File.ReadAllBytes(diag.FileName, ident);
    RETURN_CODE rt = IDT_Device.SharedController.device_updateDeviceFirmware(file, ident);
    if (rt == RETURN_CODE.RETURN_CODE_DO_SUCCESS)
    {
        //Was a success
    }
    else
    {
        //Error starting firmware download
    }
}
```

Example monitoring firmware update status / success

```
private void MessageCallBack(IDTechSDK.IDT_DEVICE_Types type, DeviceState state, byte[] data,
    IDTTransactionData cardData, EMV_Callback emvCallback, RETURN_CODE transactionResultCode, string ident = "")
{
    switch (state)
    {
        case DeviceState.FirmwareUpdate:
            switch (transactionResultCode)
            {
                case RETURN_CODE.RETURN_CODE_FW_STARTING_UPDATE:
                    SetOutputText("Starting Firmware Update\n", ident);
                    break;
                case RETURN_CODE.RETURN_CODE_DO_SUCCESS:
                    SetOutputText("Firmware Update Successful\n", ident);
                    break;
                case RETURN_CODE.RETURN_CODE_APPLYING_FIRMWARE_UPDATE:
```

```

        SetOutputText("Applying Firmware Update...\n", ident);
        break;
    case RETURN_CODE.RETURN_CODE_ENTERING_BOOTLOADER_MODE:
        SetOutputText("Entering Bootloader Mode...\n", ident);
        break;
    case RETURN_CODE.RETURN_CODE_BLOCK_TRANSFER_SUCCESS:

        int start = data[0] * 0x100 + data[1];
        int end = data[2] * 0x100 + data[3];

        SetOutputText("Sent block " + start.ToString() + " of " + end.ToString() + "\n", ident)
;
        break;
    default:
        SetOutputText("Firmware Update Error Code: " + "0x" + String.Format("{0:X}", (ushort)
transactionResultCode) + ": " + IDTechSDK.errorCode.getErrorString(transactionResultCode) + "\r\n",
ident);
        break;
    }
    break;
}
}
}

```

21.6.2.69 RETURN_CODE IDTechSDK.IDT_KioskIII.device_updateFirmwareFromZip (byte[] *zipfile*, string *ident* = "", bool *isForeground* = false)

Update Firmware From Zip

Performs one or more device firmware updates from firmware files passed as a compressed archive with a valid configuration file (.json format)

Parameters

<i>zipfile</i>	Zip file containing on or more firmware update files and a .json configuration file
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.
<i>isForeground</i>	If TRUE, will block program until firmware update complete. Otherwise, FALSE performs FW update on background.

Returns

RETURN_CODE: Values can be parsed with `errorCode.getErrorString(string ident = "")`

21.6.2.70 RETURN_CODE IDTechSDK.IDT_KioskIII.felica_authentication (byte[] *key*, string *ident* = "")

FeliCa Authentication

Provides a key to be used in a follow up FeliCa Read with MAC (3 blocks max) or Write with MAC (1 block max). This command must be executed before each Read w/MAC or Write w/MAC command

NOTE: The reader must be in Pass Through Mode for FeliCa commands to work.

Parameters

<i>key</i>	16 byte key used for MAC generation of Read or Write with MAC
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with `errorCode.getErrorString()`

21.6.2.71 **RETURN_CODE** IDTechSDK.IDT_KioskIII.felica_read (byte[] *serviceCode*, int *numBlocks*, byte[] *blockList*, ref byte[] *blocks*, string *ident* = " ")

FeliCa Read

Reads up to 4 blocks.

NOTE: The reader must be in Pass Through Mode for FeliCa commands to work.

Parameters

<i>serviceCode</i>	Service Code List. Each service code in Service Code List = 2 bytes of data
<i>numBlocks</i>	Number of blocks
<i>blockList</i>	Blocks to read. Maximum 4 block requests
<i>blocks</i>	Blocks read. Each block 16 bytes.
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with `errorCode.getErrorString()`

21.6.2.72 **RETURN_CODE** IDTechSDK.IDT_KioskIII.felica_readWithMac (int *numBlocks*, byte[] *blockList*, ref byte[] *blocks*, string *ident* = " ")

FeliCa Read with MAC Generation

Reads up to 3 blocks with MAC Generation. FeliCa Authentication must be performed first

NOTE: The reader must be in Pass Through Mode for FeliCa commands to work.

Parameters

<i>numBlocks</i>	Number of blocks
<i>blockList</i>	Block to read. Each block in blockList Maximum 3 block requests
<i>blocks</i>	Blocks read. Each block 16 bytes.
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with `errorCode.getErrorString()`

21.6.2.73 **RETURN_CODE** IDTechSDK.IDT_KioskIII.felica_requestService (byte[] *nodeCode*, ref byte[] *response*, string *ident* = " ")

FeliCa Request Service

Perform functions a Felica Request Service

NOTE: The reader must be in Pass Through Mode for FeliCa commands to work.

Parameters

<i>nodeCode</i>	Node Code
<i>response</i>	Response as explained in FeliCA Lite-S User's Manual
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with `errorCode.getErrorString()`

21.6.2.74 RETURN_CODE IDTechSDK.IDT_KioskIII.felica_SendCommand (byte[] *command*, ref byte[] *response*, string *ident* = " ")

FeliCa Send Command

Send a Felica Command

Parameters

<i>command</i>	Command data from settlement center to be sent to felica card
<i>response</i>	Response data from felica card.

Returns

RETURN_CODE: Values can be parsed with `device_getIDGStatusCodeString()`

21.6.2.75 RETURN_CODE IDTechSDK.IDT_KioskIII.felica_write (byte[] *serviceCode*, int *blockCount*, byte[] *blockList*, byte[] *data*, ref byte[] *statusFlag*, string *ident* = " ")

FeliCa Write

Writes a block

NOTE: The reader must be in Pass Through Mode for FeliCa commands to work.

Parameters

<i>serviceCode</i>	Service Code list. Each service code must be be 2 bytes
<i>blockCount</i>	Block Count
<i>blockList</i>	Block list.
<i>data</i>	Block to write. Must be 16 bytes.
<i>statusFlag</i>	Status flag response as explained in FeliCA Lite-S User's Manual, Section 4.5
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with `errorCode.getErrorString()`

21.6.2.76 RETURN_CODE IDTechSDK.IDT_KioskIII.felica_writeWithMac (int *blockNumber*, byte[] *data*, string *ident* = " ")

FeliCa Write with MAC Generation

Writes a block with MAC Generation. FeliCa Authentication must be performed first

NOTE: The reader must be in Pass Through Mode for FeliCa commands to work.

Parameters

<i>blockNumber</i>	Number of block
<i>data</i>	Block to write. Must be 16 bytes.
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with `errorCode.getErrorString()`

21.6.2.77 `static int IDTechSDK.IDT_KioskIII.getCommandTimeout (string ident = " ") [static]`

Get Command Timeout

Gets the default timeout (in milliseconds) waiting for a blocking command response

Return values

<i>time</i>	Time
-------------	------

21.6.2.78 `static void IDTechSDK.IDT_KioskIII.lcd_retrieveMessage (DisplayMessages.DISPLAY_MESSAGE_LANGUAGE lang, DisplayMessages.DISPLAY_MESSAGE_IDENTIFIER id, ref string line1, ref string line2) [static]`

Retrieve LCD Message

Returns the string value for a message ID returned for LCD messaging

Parameters

<i>lang</i>	Language.
<i>id</i>	Message ID
<i>line1</i>	Line 1 string value
<i>line2</i>	Line 2 string value

Returns

RETURN_CODE: Values can be parsed with `device_getResponseCodeString`

21.6.2.79 `static String IDTechSDK.IDT_KioskIII.SDK_Version () [static]`

SDK Version

- All Devices

Returns the current version of SDK

Returns

Framework version

21.6.2.80 `static void IDTechSDK.IDT_KioskIII.setCallback (CallBack my_Callback) [static]`

Set Callback

Sets the class callback

21.6.2.81 `static void IDTechSDK.IDT_KioskIII.setCallback (IntPtr my_Callback, SynchronizationContext context)`
`[static]`

Set Callback

Sets the class callback

Parameters

<i>my_Callback</i>	The callback function to receive the response message from device. defined as follows. public unsafe delegate void MFCCallBack(Parameters parameters, ident);
<i>context</i>	The context of the UI thread

21.6.2.82 `static void IDTechSDK.IDT_KioskIII.setCommandTimeout (int milliseconds, string ident = " ")` `[static]`

Set Command Timeout

Sets the default timeout (in milliseconds) waiting for a blocking command response

Parameters

<i>milliseconds</i>	Time
---------------------	------

21.6.2.83 `static bool IDTechSDK.IDT_KioskIII.useSerialPort (int port)` `[static]`

Use Serial Port Interface

Instructs SDK to attempt to use the Serial Port for communication with Kiosk III/IV using default baud rate

Parameters

<i>port</i>	Serial Port to use. Example COM1 = 1.
-------------	---------------------------------------

Returns

bool TRUE=successful, FALSE=failure

21.6.2.84 `static bool IDTechSDK.IDT_KioskIII.useSerialPort (int port, int baud, string ident = " ")` `[static]`

Use Serial Port Interface with baud rate

Instructs SDK to attempt to use the Serial Port for communication with Kiosk III/IV

Parameters

<i>port</i>	Serial Port to use. Example COM1 = 1.
<i>baud</i>	Baud rate to override default. Example 115200;

Returns

bool TRUE=successful, FALSE=failure

21.6.2.85 `static bool IDTechSDK.IDT_KioskIII.useSerialPortLinux (string path) [static]`

Use Serial Port Interface on Linux

Instructs SDK to attempt to use the Serial Port for communication with BTMag using default baud rate on Linux implementations

Parameters

<i>path</i>	Path to use. Example /dev/ttyUSB*
-------------	-----------------------------------

Returns

bool TRUE=successful, FALSE=failure

21.6.2.86 `static bool IDTechSDK.IDT_KioskIII.useSerialPortLinux (string path, int baud) [static]`

Use Serial Port Interface on Linux with baud rate

Instructs SDK to attempt to use the Serial Port for communication with BTPay on Linux implementations

Parameters

<i>path</i>	Path to use. Example /dev/ttyUSB*
<i>baud</i>	Baud rate to override default. Example 115200;

Returns

bool TRUE=successful, FALSE=failure

21.6.3 Property Documentation

21.6.3.1 `IDT_KioskIII IDTechSDK.IDT_KioskIII.SharedController [static],[get]`

Singleton Instance

Establishes an singleton instance of [IDT_KioskIII](#) class.

Returns

Instance of [IDT_KioskIII](#)

The documentation for this class was generated from the following file:

- /Users/randy/Repo/universal-sdk/IDTechSDK_STD/DeviceHeaders/IDT_KioskIII.cs

21.7 IDTechSDK.IDT_L100 Class Reference

Public Member Functions

- RETURN_CODE [device_getFirmwareVersion](#) (ref string response, string ident="")
- RETURN_CODE [device_StartRKI](#) (int type, string ident="")
- RETURN_CODE [device_updateDeviceFirmware](#) (byte[] firmwareData, string ident="")
- RETURN_CODE [device_sendPAE](#) (string command, ref string response, int timeout, string ident="")

- RETURN_CODE [device_sendDataCommand](#) (string cmd, bool calcLRC, ref byte[] response, string ident="")
- RETURN_CODE [device_sendDataCommand_ext](#) (string cmd, bool calcLRC, ref byte[] response, int timeout, bool noResponse, string ident="")
- RETURN_CODE [config_getModelNumber](#) (ref string response, string ident="")
- RETURN_CODE [config_getSerialNumber](#) (ref string response, string ident="")
- bool [config_setCmdTimeOutDuration](#) (int newTimeOut, string ident="")
- RETURN_CODE [pin_getEncryptedPIN](#) (int keyType, string PAN, string message, int timeout, bool isAES=false, string ident="")
- RETURN_CODE [pin_getManualPanEntry](#) (bool csc, bool ADR, bool ZIP, string ident="")
- RETURN_CODE [pin_getFunctionKey](#) (string ident="")
- RETURN_CODE [pin_sendBeep](#) (int frequency, int duration, string ident="")
- RETURN_CODE [device_rebootDevice](#) (string ident="")
- RETURN_CODE [pin_setKeyPressCapture](#) (bool showKeyValue, string ident="")
- RETURN_CODE [pin_cancelPINEntry](#) (string ident="")
- RETURN_CODE [device_getKeyStatus](#) (ref byte[] status, string ident="")
- RETURN_CODE [pin_promptForKeyInput](#) (string message, byte[] signature, bool maskInput, int minLen, int maxLen, string ident="")
- RETURN_CODE [pin_promptForAmountInput](#) (string message, byte[] signature, bool maskInput, int minLen, int maxLen, string ident="")
- RETURN_CODE [pin_promptForKeyInput](#) (int messageID, int languageID, bool maskInput, int minLen, int maxLen)
- RETURN_CODE [pin_promptForAmountInput](#) (int messageID, int languageID, int minLen, int maxLen)
- RETURN_CODE [device_setSleepModeTime](#) (int time, string ident="")
- RETURN_CODE [device_startRKI](#) (string ident="")
- RETURN_CODE [device_enterStopMode](#) (string ident="")
- RETURN_CODE [device_setDateTime](#) (string ident="")
- RETURN_CODE [device_getDateTime](#) (ref byte[] dateTime, string ident="")
- RETURN_CODE [lcd_clearDisplay](#) (int lineNumber, string ident="")
- RETURN_CODE [lcd_clearAllLines](#) (string ident="")
- RETURN_CODE [lcd_savePrompt](#) (int promptNumber, string prompt, string ident="")
- RETURN_CODE [lcd_displayPrompt](#) (int promptNumber, int lineNumber, string ident="")
- RETURN_CODE [lcd_displayMessage](#) (int lineNumber, string message, string ident="")
- RETURN_CODE [config_setEthernetMACAddress](#) (byte[] address, string ident="")
- RETURN_CODE [config_getEthernetMACAddress](#) (ref byte[] address, string ident="")
- RETURN_CODE [config_getNetworkConfiguration](#) (ref bool isStatic, ref string address, ref string subnet, ref string gateway, ref string dns, string ident="")
- RETURN_CODE [config_setNetworkConfiguration](#) (bool isStatic, string address, string subnet, string gateway, string dns, string ident="")
- RETURN_CODE [lcd_enableBacklight](#) (bool enable, string ident="")
- RETURN_CODE [lcd_getBacklightStatus](#) (ref bool enabled, string ident="")
- RETURN_CODE [config_setBaudRate](#) (int baud, string ident="")
- RETURN_CODE [config_getBaudRate](#) (ref int baud, string ident="")
- RETURN_CODE [device_getRKIStatus](#) (bool isProd, bool isMultiKey, ref string status, ref Dictionary< string, RKI_KEY_TYPE > keys, string ident="")
- RETURN_CODE [device_getAnyRKIStatus](#) (bool isMultiKey, ref string status, ref Dictionary< string, RKI_KEY_TYPE > keys, string ident="")
- RETURN_CODE [device_updateFirmwareFromZip](#) (byte[] zipfile, string ident="", bool isForeground=false)
- RETURN_CODE [device_sendConfigurationFromZip](#) (byte[] zip, string filename, VC_OPERATION_TYPE type, bool matchFW, string ident="", bool isForeground=false)
- RETURN_CODE [device_sendConfiguration](#) (string filename, VC_OPERATION_TYPE type, bool matchFW, string ident="", bool isForeground=false)
- RETURN_CODE [device_readConfigurationToMemory](#) (string memo, Configs.InstallRules rules, List< Configs.DeviceCommand > cmds, string ident="", bool isForeground=false)
- RETURN_CODE [device_getConfigurationFromMemory](#) (ref string json, string ident="")
- RETURN_CODE [device_RemoteKeyInjection](#) (RKI_KEY_TYPE type, string keyName, string ident="", bool performOnForeground=false)

Static Public Member Functions

- static bool [useSerialPort](#) (int port)
- static bool [useSerialPortLinux](#) (string path)
- static bool [useSerialPortLinux](#) (string path, int baud)
- static int [getCommandTimeout](#) (string ident="")
- static void [setCommandTimeout](#) (int milliseconds, string ident="")
- static bool [useSerialPort](#) (int port, int baud, string ident="")
- static bool [closeSerialPort](#) ()
- static bool [closeUSB](#) (string ident="")
- static void [initSC](#) ()
- static void [setCallback](#) (CallBack my_Callback)
- static void [setCallback](#) (IntPtr my_Callback, SynchronizationContext context)
- static String [SDK_Version](#) ()
- static void [lcd_retrieveMessage](#) (DisplayMessages.DISPLAY_MESSAGE_LANGUAGE lang, DisplayMessages.DISPLAY_MESSAGE_IDENTIFIER id, ref string line1, ref string line2)

Properties

- static [IDT_L100 SharedController](#) [get]

21.7.1 Member Function Documentation

21.7.1.1 static bool IDTechSDK.IDT_L100.closeSerialPort () [static]

Close Serial Port Interface

Instructs SDK to close the Serial Port if connected to L100

Returns

bool TRUE=successful, FALSE=failure

21.7.1.2 static bool IDTechSDK.IDT_L100.closeUSB (string ident = " ") [static]

Close USB

Instructs SDK to close the USB if connected to L100

Returns

bool TRUE=successful, FALSE=failure

21.7.1.3 RETURN_CODE IDTechSDK.IDT_L100.config_getBaudRate (ref int baud, string ident = " ")

Get Baud Rate

Gets the buad rate for RS-232 communication.

Parameters

<i>baud</i>	<ul style="list-style-type: none"> • 2 = 2400 • 3 = 4800 • 4 = 9600 • 6 = 19200 • 7 = 38400 • 9 = 115200
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString

21.7.1.4 RETURN_CODE IDTechSDK.IDT_L100.config_getEthernetMACAddress (ref byte[] *address*, string *ident* = " ")

Get Device Ethernet MAC Address

Parameters

<i>address</i>	6-byte MAC Address
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString

21.7.1.5 RETURN_CODE IDTechSDK.IDT_L100.config_getModelNumber (ref string *response*, string *ident* = " ")

Polls device for Model Number

Parameters

<i>response</i>	Returns Model Number
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with errorCode.getErrorString()

21.7.1.6 RETURN_CODE IDTechSDK.IDT_L100.config_getNetworkConfiguration (ref bool *isStatic*, ref string *address*, ref string *subnet*, ref string *gateway*, ref string *dns*, string *ident* = " ")

Get Device Network Configuration

Parameters

<i>isStatic</i>	TRUE = Static IP, FALSE = DHCP
<i>address</i>	Device IP Address as string. Example "192.168.1.15"
<i>subnet</i>	Device Subnet as string. Example "255.255.255.0"
<i>gateway</i>	Device Gateway address as a string. Example "8.8.8.8"
<i>dns</i>	Device DNS address as string. Example "192.168.1.22"
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString

21.7.1.7 RETURN_CODE IDTechSDK.IDT_L100.config_getSerialNumber (ref string response, string ident = " ")

Polls device for Serial Number

Parameters

<i>response</i>	Returns Serial Number
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with errorCode.getErrorString()

21.7.1.8 RETURN_CODE IDTechSDK.IDT_L100.config_setBaudRate (int baud, string ident = " ")

Set Baud Rate

Sets the buad rate for RS-232 communication.

Parameters

<i>baud</i>	<ul style="list-style-type: none"> • 2 = 2400 • 3 = 4800 • 4 = 9600 • 6 = 19200 • 7 = 38400 • 9 = 115200
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString

21.7.1.9 bool IDTechSDK.IDT_L100.config_setCmdTimeOutDuration (int *newTimeOut*, string *ident* = " ")

Command Acknowledgement Timeout

Sets the amount of seconds to wait for an {ACK} to a command before a timeout. Responses should normally be received under one second. Default is 3 seconds

Parameters

<i>newTimeOut</i>	Timeout value. Valid range 1 - 60 seconds
-------------------	---

Returns

Success flag. Determines if value was set and in range.

21.7.1.10 RETURN_CODE IDTechSDK.IDT_L100.config_setEthernetMACAddress (byte[] *address*, string *ident* = " ")

Set Device Ethernet MAC Address

Parameters

<i>address</i>	6-byte MAC Address
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString

21.7.1.11 RETURN_CODE IDTechSDK.IDT_L100.config_setNetworkConfiguration (bool *isStatic*, string *address*, string *subnet*, string *gateway*, string *dns*, string *ident* = " ")

Set Device Network Configuration

Parameters

<i>isStatic</i>	TRUE = Static IP, FALSE = DHCP
<i>address</i>	Device IP Address as string. Example "192.168.1.15"
<i>subnet</i>	Device Subnet as string. Example "255.255.255.0"
<i>gateway</i>	Device Gateway address as a string. Example "8.8.8.8"
<i>dns</i>	Device DNS address as string. Example "192.168.1.22"
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString

21.7.1.12 RETURN_CODE IDTechSDK.IDT_L100.device_enterStopMode (string *ident* = " ")

Enter Stop Mode

Set device enter to stio mode. In stop mode, LCD display and backlight is off. Stop mode reduces power consumption to the lowest possible level. A unit in stop mode can only be woken up by a physical key press.

Parameters

<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.
--------------	--

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString

21.7.1.13 RETURN_CODE IDTechSDK.IDT_L100.device_getAnyRKIStatus (bool *isMultiKey*, ref string *status*, ref Dictionary< string, RKI_KEY_TYPE > *keys*, string *ident* = " ")

Get RKI Status Polls the RKI servers to validate device RKI eligibility. Note: if device type is known in advance (production or demo device), it is more efficient to use device_getRKIStatus instead

Parameters

<i>isMultiKey</i>	True = MultiKey, False = SingleKey
<i>status</i>	RKI status returned as a string
<i>keys</i>	Returns all available keys
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with errorCode.getErrorString(). When no data is available, return code = RETURN_CODE_NO_DATA_AVAILABLE

21.7.1.14 RETURN_CODE IDTechSDK.IDT_L100.device_getConfigurationFromMemory (ref string *json*, string *ident* = " ")

Read Device Configuration From Memory Retrieves the ViVOconfig configuration data from memory after a successful device_readConfigurationToMemory

Parameters

<i>json</i>	Returns the .json file data that can be saved to disk
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with errorCode.getErrorString(). When no data is available, return code = RETURN_CODE_NO_DATA_AVAILABLE

21.7.1.15 RETURN_CODE IDTechSDK.IDT_L100.device_getDateTime (ref byte[] *dateTime*, string *ident* = " ")

Get Date Time

Gets current system date and time of the device.

Parameters

<i>dateTime</i>	The date time returned as follows: <ul style="list-style-type: none"> • byte 0: Year 00-99 • byte 1: Month 01-12 • byte 2: Date 01-31 • byte 3: Hour 00-23 • byte 4: Minute 00-59 • byte 5: Second 00-59
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString

21.7.1.16 RETURN_CODE IDTechSDK.IDT_L100.device_getFirmwareVersion (ref string *response*, string *ident* = " ")

Polls device for Firmware Version

Parameters

<i>response</i>	Response returned of Firmware Version
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with errorCode.getErrorString()

21.7.1.17 RETURN_CODE IDTechSDK.IDT_L100.device_getKeyStatus (ref byte[] *status*, string *ident* = " ")

Get Key Status

Gets the status of loaded keys

Parameters

<i>status</i>	byte 0: PIN DUKPT Key, 1 Exists, 0 None, 0xFF STOP byte 1: PIN Master Key, 1 Exists, 0 None byte 2: PIN Session Key, 1 Exists, 0 None byte 3: Account/MSR DUKPT Key, Does not support, always 0 byte 4: Account/ICC DUKPT Key, Does not support, always 0 byte 5: Admin DUKPT Key, 1 Exists, 0 None, 0xFF STOP
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with errorCode.getErrorString()

21.7.1.18 **RETURN_CODE** IDTechSDK.IDT_L100.device_getRKIStatus (bool *isProd*, bool *isMultiKey*, ref string *status*, ref Dictionary< string, RKI_KEY_TYPE > *keys*, string *ident* = " ")

Get RKI Status From Specified Server Polls the RKI server to validate device RKI eligibility

Parameters

<i>isProd</i>	TRUE = poll production RKI server, FALSE = poll demo RKI server
<i>isMultiKey</i>	True = MultiKey, False = SingleKey
<i>status</i>	RKI status returned as a string
<i>keys</i>	Returns dictionary of available keys
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with `errorCode.getErrorString()`. When no data is available, return code = **RETURN_CODE_NO_DATA_AVAILABLE**

21.7.1.19 **RETURN_CODE** IDTechSDK.IDT_L100.device_readConfigurationToMemory (string *memo*, Configs.InstallRules *rules*, List< Configs.DeviceCommand > *cmds*, string *ident* = " ", bool *isForeground* = false)

Read Device Configuration To Memory Executes a ViVOconfig read of device settings and stores the results in memory

Once a ViVOconfig read starts, all messages will be returned to the callback as `DeviceState.ViVOconfig`. The ViVOconfig operation is complete when the return code from a `DeviceState.ViVOconfig` message is one of the following values:

- **RETURN_CODE_SDK_VIVOCONFIG_READ_CONFIG_SUCCESS** = Read process completed successfully. Execute `device_getConfigurationFromMemory` to retrieve the configuration data
- **RETURN_CODE_SDK_VIVOCONFIG_READ_CONFIG_FAILED** = Read process FAILED.

Parameters

<i>memo</i>	Optional: populates the memo field of the captured .json file
<i>rules</i>	Optional: populates the .json file with available install rules
<i>cmds</i>	Optional: populates the .json file with commands to execute
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.
<i>isForeground</i>	If TRUE, will block program until update complete. Otherwise, FALSE performs update on background.

Returns

RETURN_CODE: Values can be parsed with `errorCode.getErrorString()`. When no data is available, return code = **RETURN_CODE_NO_DATA_AVAILABLE**

21.7.1.20 **RETURN_CODE** IDTechSDK.IDT_L100.device_rebootDevice (string *ident* = " ")

Reboot Device

Executes a command to restart the device.

- Card data is cleared, resetting card status bits.

- Response data of the previous command is cleared.
- Resetting firmware.

Parameters

<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.
--------------	--

Returns

RETURN_CODE: Values can be parsed with `device_getResponseCodeString`

21.7.1.21 `RETURN_CODE IDTechSDK.IDT_L100.device_RemoteKeyInjection (RKL_KEY_TYPE type, string keyName, string ident = " ", bool performOnForeground = false)`

Remote Key Injection Performs a remote key injection for the device

Parameters

<i>type</i>	Remote Key Injection Type
<i>keyName</i>	Name of key (optional)
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.
<i>performOnForeground</i>	TRUE = block during process (default is false)

Returns

RETURN_CODE: Values can be parsed with `errorCode.getErrorString()`. When no data is available, return code = RETURN_CODE_NO_DATA_AVAILABLE

21.7.1.22 `RETURN_CODE IDTechSDK.IDT_L100.device_sendConfiguration (string filename, VC_OPERATION_TYPE type, bool matchFW, string ident = " ", bool isForeground = false)`

Send Configuration Executes a ViVOconfig update or verify to a device

Once a ViVOconfig write or verify starts, all messages will be returned to the callback as `DeviceState.ViVOconfig`. The ViVOconfig operation is complete when the return code from a `DeviceState.ViVOconfig` message is one of the following values:

- RETURN_CODE_SDK_VIVOCONFIG_VERIFY_SUCCESS = Verification process completed successfully. No differences found.
- RETURN_CODE_SDK_VIVOCONFIG_VERIFY_SUCCESS_WITH_WARNING = Verification process completed with warnings.
- RETURN_CODE_SDK_VIVOCONFIG_VERIFY_FAILED = Verification process FAILED
- RETURN_CODE_SDK_VIVOCONFIG_WRITE_SUCCESS = Write process completed successfully.
- RETURN_CODE_SDK_VIVOCONFIG_WRITE_SUCCESS_WITH_WARNING = Write process completed with warnings
- RETURN_CODE_SDK_VIVOCONFIG_WRITE_FAILED = Write process FAILED

Parameters

<i>filename</i>	The .json configuration file to update the device with NOTE: You can also use this parameter to pass the path of where the .json file is located at on local storage instead
-----------------	--

Parameters

<i>filename</i>	The name of the .json configuration file
<i>type</i>	VC_OPERATION_TYPE

- VC_OPERATION_TYPE_WRITE = Write File To Device, Hash must validate
- VC_OPERATION_TYPE_VERIFY = Verify Device With File
- VC_OPERATION_TYPE_WRITE_IGNORE_HASH = Write File To Device, Ignore Hash
- VC_OPERATION_TYPE_WRITE_FIX_HASH = Write File To Device, Fix Hash If Necessary

Parameters

<i>matchFW</i>	TRUE = Device FW must match file FW, FALSE = Don't validate device FW against file
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.
<i>isForeground</i>	If TRUE, will block program until update complete. Otherwise, FALSE performs update on background.

Returns

RETURN_CODE: Values can be parsed with `errorCode.getErrorString()`. When no data is available, return code = RETURN_CODE_NO_DATA_AVAILABLE

21.7.1.23 `RETURN_CODE IDTechSDK.IDT_L100.device_sendConfigurationFromZip (byte[] zip, string filename, VC_OPERATION_TYPE type, bool matchFW, string ident = " ", bool isForeground = false)`

Send Configuration From Zip Executes a ViVOconfig update or verify to a device using files from a .zip archive

Once a ViVOconfig write or update starts, all messages will be returned to the callback as `DeviceState.ViVOconfig`. The ViVOconfig operation is complete when the return code from a `DeviceState.ViVOconfig` message is one of the following values:

- RETURN_CODE_SDK_VIVOCONFIG_VERIFY_SUCCESS = Verification process completed successfully. No differences found.
- RETURN_CODE_SDK_VIVOCONFIG_VERIFY_SUCCESS_WITH_WARNING = Verification process completed with warnings.
- RETURN_CODE_SDK_VIVOCONFIG_VERIFY_FAILED = Verification process FAILED
- RETURN_CODE_SDK_VIVOCONFIG_WRITE_SUCCESS = Write process completed successfully.
- RETURN_CODE_SDK_VIVOCONFIG_WRITE_SUCCESS_WITH_WARNING = Write process completed with warnings
- RETURN_CODE_SDK_VIVOCONFIG_WRITE_FAILED = Write process FAILED

Parameters

<i>zip</i>	.Zip archive containing the .json configuration file and also any associated image assets NOTE: You can also use this parameter to pass the path of where the .zip file is located at on local storage instead
<i>filename</i>	The name of the .json configuration file
<i>type</i>	VC_OPERATION_TYPE

- VC_OPERATION_TYPE_WRITE = Write File To Device, Hash must validate
- VC_OPERATION_TYPE_VERIFY = Verify Device With File
- VC_OPERATION_TYPE_WRITE_IGNORE_HASH = Write File To Device, Ignore Hash
- VC_OPERATION_TYPE_WRITE_FIX_HASH = Write File To Device, Fix Hash If Necessary

Parameters

<i>matchFW</i>	TRUE = Device FW must match file FW, FALSE = Don't validate device FW against file
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.
<i>isForeground</i>	If TRUE, will block program until update complete. Otherwise, FALSE performs update on background.

Returns

RETURN_CODE: Values can be parsed with `errorCode.getErrorString()`. When no data is available, return code = RETURN_CODE_NO_DATA_AVAILABLE

21.7.1.24 RETURN_CODE IDTechSDK.IDT_L100.device_sendDataCommand (string *cmd*, bool *calcLRC*, ref byte[] *response*, string *ident* = " ")

Send a data command to the device

Sends a command to the device.

Parameters

<i>cmd</i>	String representation of command to execute
<i>calcLRC</i>	If TRUE, this will wrap command with start/length/lrc/sum/end: '{STX}{Len_Low}{Len_High} data {CheckLRC} {CheckSUM} {ETX}'
<i>response</i>	Response data
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with `device_getResponseCodeString`

21.7.1.25 RETURN_CODE IDTechSDK.IDT_L100.device_sendDataCommand_ext (string *cmd*, bool *calcLRC*, ref byte[] *response*, int *timeout*, bool *noResponse*, string *ident* = " ")

Send a data command to the device - extended

Sends a command to the device.

Parameters

<i>cmd</i>	String representation of command to execute
<i>calcLRC</i>	If TRUE, this will wrap command as NGA with start/length/lrc/sum/end: '{STX}{Len_Low}{Len_High} data {CheckLRC} {CheckSUM} {ETX}'
<i>response</i>	Response data
<i>timeout</i>	Timeout value waiting for response, in milliseconds (1000 = 1 second)
<i>noResponse</i>	if TRUE, this will not wait for a response and immediately return SUCCESS
<i>calcITP</i>	If TRUE, this will wrap command as ITP with start/end/lrc: '{STX} data {ETX}{CheckLRC}'
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString

21.7.1.26 RETURN_CODE IDTechSDK.IDT_L100.device_sendPAE (string *command*, ref string *response*, int *timeout*, string *ident* = " ")

Send Payment Application Engine Command

Executes a PAE command

Parameters

<i>command</i>	ASCII command string, should start with "*PAE"
<i>response</i>	command response
<i>timeout</i>	timeout waiting for PAE response
<i>ip</i>	Optional IP address when connected via TCP/IP
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with errorCode.getErrorString()

21.7.1.27 RETURN_CODE IDTechSDK.IDT_L100.device_setDateTime (string *ident* = " ")

Set Date Time

Set current system date and time to the device.

Parameters

<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.
--------------	--

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString

21.7.1.28 RETURN_CODE IDTechSDK.IDT_L100.device_setSleepModeTime (int *time*, string *ident* = " ")

Set Sleep Mode Timer

Set device enter to sleep mode after the given time. In sleep mode, LCD display and backlight is off. Sleep mode reduces power consumption to the lowest possible level. A unit in Sleep mode can only be woken up by a physical key press.

Parameters

<i>time</i>	Enter sleep time value, in second.
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString

21.7.1.29 RETURN_CODE IDTechSDK.IDT_L100.device_StartRKI (int *type*, string *ident* = " ")

Start Remote Key Injection

Starts a remote key injection request with IDTech RKI servers. Set/Get RKI url with IDT_Device.RKI_URL.

Parameters

<i>type</i>	0 = Symmetric RKI Demo Unit 1 = Symmetric RKI Production Unit 2 = PKI-RKI Demo Unit 3 = PKI-RKI Production Unit
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString

21.7.1.30 RETURN_CODE IDTechSDK.IDT_L100.device_startRKI (string *ident* = " ")

Start Remote Key Injection

Starts a remote key injection request with IDTech RKI servers.

Parameters

<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.
--------------	--

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString

21.7.1.31 RETURN_CODE IDTechSDK.IDT_L100.device_updateDeviceFirmware (byte[] *firmwareData*, string *ident* = " ")

Update Firmware

Updates the firmware .

Parameters

<i>firmwareData</i>	Signed binary data of a firmware file provided by IDTech
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with errorCode.getErrorString()

After you pass the firmwareData file, a new thread will start to execute the firmware download. You will receive status of the progress through callbacks to the IDTechSDK.Callback() delegate. The following parameters will be passed back:

- sender = IDT_DEVICE_Types.IDT_DEVICE_AUGUSTA

- state = DeviceState.FirmwareUpdate
- transactionResultCode = status of the firmware update (starting, entering bootloader, applying update, block success, firmware success)
- data = File Progress. Four bytes, with bytes [0][1] = current block, and bytes [2][3] = total blocks. 0x00030010 = block 3 of 16

Example code starting a firmware update

```
OpenFileDialog diag = new OpenFileDialog(ident);

diag.Filter = "NGA FW Files|*.fm";

if (diag.ShowDialog() == DialogResult.OK)
{
    byte[] file = File.ReadAllBytes(diag.FileName, ident);
    RETURN_CODE rt = IDT_Device.SharedController.device_updateDeviceFirmware(file, ident);
    if (rt == RETURN_CODE.RETURN_CODE_DO_SUCCESS)
    {
        //Was a success
    }
    else
    {
        //Error starting firmware download
    }
}
```

Example monitoring firmware update status / success

```
private void MessageCallBack(IDTechSDK.IDT_DEVICE_Types type, DeviceState state, byte[] data,
    IDTTransactionData cardData, EMV_Callback emvCallback, RETURN_CODE transactionResultCode, string ident = "")
{
    switch (state)
    {
        case DeviceState.FirmwareUpdate:
            switch (transactionResultCode)
            {
                case RETURN_CODE.RETURN_CODE_FW_STARTING_UPDATE:
                    SetOutputText("Starting Firmware Update\n", ident);
                    break;
                case RETURN_CODE.RETURN_CODE_DO_SUCCESS:
                    SetOutputText("Firmware Update Successful\n", ident);
                    break;
                case RETURN_CODE.RETURN_CODE_APPLYING_FIRMWARE_UPDATE:
                    SetOutputText("Applying Firmware Update...\n", ident);
                    break;
                case RETURN_CODE.RETURN_CODE_ENTERING_BOOTLOADER_MODE:
                    SetOutputText("Entering Bootloader Mode...\n", ident);
                    break;
                case RETURN_CODE.RETURN_CODE_BLOCK_TRANSFER_SUCCESS:
                    int start = data[0] * 0x100 + data[1];
                    int end = data[2] * 0x100 + data[3];

                    SetOutputText("Sent block " + start.ToString() + " of " + end.ToString() + "\n", ident);
                    break;
                default:
                    SetOutputText("Firmware Update Error Code: " + "0x" + String.Format("{0:X}", (ushort)
transactionResultCode) + ": " + IDTechSDK.errorCode.getErrorString(transactionResultCode) + "\r\n",
ident);
                    break;
            }
        break;
    }
}
```

21.7.1.32 RETURN_CODE IDTechSDK.IDT_L100.device_updateFirmwareFromZip (byte[] zipfile, string ident = "", bool isForeground = false)

Update Firmware From Zip

Performs one or more device firmware updates from firmware files passed as a compressed archive with a valid configuration file (.json format)

Parameters

<i>zipfile</i>	Zip file containing on or more firmware update files and a .json configuration file
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.
<i>isForeground</i>	If TRUE, will block program until firmware update complete. Otherwise, FALSE performs FW update on background.

Returns

RETURN_CODE: Values can be parsed with `errorCode.getErrorString(string ident = "")`

21.7.1.33 `static int IDTechSDK.IDT_L100.getCommandTimeout (string ident = " ") [static]`

Get Command Timeout

Gets the default timeout (in milliseconds) waiting for a blocking command response

Return values

<i>time</i>	Time
-------------	------

21.7.1.34 `RETURN_CODE IDTechSDK.IDT_L100.lcd_clearAllLines (string ident = " ")`

Clear LCD Display

Clears all lines of the LCD Display.

Parameters

<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.
--------------	--

Returns

RETURN_CODE: Values can be parsed with `device_getResponseCodeString`

21.7.1.35 `RETURN_CODE IDTechSDK.IDT_L100.lcd_clearDisplay (int lineNumber, string ident = " ")`

Clear LCD Display Line

Clears the line number of the LCD Display.

Parameters

<i>lineNumber</i>	Line number to clear (1-4)
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with `device_getResponseCodeString`

21.7.1.36 RETURN_CODE IDTechSDK.IDT_L100.lcd_displayMessage (int *lineNumber*, string *message*, string *ident* = " ")

Display Message on Line

Displays a message on a display line.

Parameters

<i>lineNumber</i>	Line number to display message on (1-4)
<i>message</i>	Message to display
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString

21.7.1.37 RETURN_CODE IDTechSDK.IDT_L100.lcd_displayPrompt (int *promptNumber*, int *lineNumber*, string *ident* = " ")

Display Prompt on Line

Displays a message prompt from L100 memory.

Parameters

<i>promptNumber</i>	Prompt number (0-9)
<i>lineNumber</i>	Line number to display message prompt (1-4)
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString

21.7.1.38 RETURN_CODE IDTechSDK.IDT_L100.lcd_enableBacklight (bool *enable*, string *ident* = " ")

Enable/Disable LCD Backlight

Turns on/off the LCD back lighting.

Parameters

<i>enable</i>	TRUE = turn ON backlight, FALSE = turn OFF backlight
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString

21.7.1.39 RETURN_CODE IDTechSDK.IDT_L100.lcd_getBacklightStatus (ref bool *enabled*, string *ident* = " ")

Get Backlight Status

Returns the status of the LCD back lighting.

Parameters

<i>enabled</i>	TRUE = Backlight is ON, FALSE = Backlight is OFF
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString

21.7.1.40 static void IDTechSDK.IDT_L100.lcd_retrieveMessage (DisplayMessages.DISPLAY_MESSAGE_LANGUAGE *lang*, DisplayMessages.DISPLAY_MESSAGE_IDENTIFIER *id*, ref string *line1*, ref string *line2*) [static]

Retrieve LCD Message

Returns the string value for a message ID returned for LCD messaging

Parameters

<i>lang</i>	Language.
<i>id</i>	Message ID
<i>line1</i>	Line 1 string value
<i>line2</i>	Line 2 string value
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString

21.7.1.41 RETURN_CODE IDTechSDK.IDT_L100.lcd_savePrompt (int *promptNumber*, string *prompt*, string *ident* = " ")

Save Prompt

Saves a message prompt to L100 memory.

Parameters

<i>promptNumber</i>	Prompt number (0-9)
<i>prompt</i>	Prompt string (up to 20 characters)
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString

21.7.1.42 RETURN_CODE IDTechSDK.IDT_L100.pin_cancelPINEntry (string *ident* = " ")

Cancel PIN Entry

Cancel "Get Function Key" & "Get Encrypted PIN" & "Get Numeric" & "Get Amount"

Parameters

<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.
--------------	--

Returns

RETURN_CODE: Values can be parsed with `errorCode.getErrorString()`

21.7.1.43 RETURN_CODE IDTechSDK.IDT_L100.pin_getEncryptedPIN (int *keyType*, string *PAN*, string *message*, int *timeout*, bool *isAES* = false, string *ident* = " ")

Get Encrypted PIN

Requests PIN Entry

Parameters

<i>keyType</i>	<ul style="list-style-type: none"> • 0x00- MKSK-TDES: External Plaintext PAN • 0x01- DUKPT-TDES: External Plaintext PAN • 0x10 MKSK-TDES: External Ciphertext PAN • 0x11 DUKPT-TDES: External Ciphertext PAN
<i>PAN</i>	Account Number
<i>message</i>	Message to display = timeout
<i>isAES</i>	set to TRUE if PEK is AES, not TDES
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with `errorCode.getErrorString()`

21.7.1.44 RETURN_CODE IDTechSDK.IDT_L100.pin_getFunctionKey (string *ident* = " ")

Get Function Key

Captures a function key

- Backspace = B
- Cancel = C
- Enter = E
- * = *
- # = #
- Help = ?
- Function Key 1 = F1
- Function Key 2 = F2
- Function Key 3 = F3

Timeout hard wired 3 minutes

Parameters

<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.
--------------	--

Returns

RETURN_CODE: Values can be parsed with `errorCode.getErrorString()`

21.7.1.45 RETURN_CODE IDTechSDK.IDT_L100.pin_getManualPanEntry (bool *csc*, bool *ADR*, bool *ZIP*, string *ident* = " ")

Get Manual Pan Entry

prompt the user to manually enter a card PAN and Expiry Date (and optionally CSC) from the keypad and return it to the POS.

Parameters

<i>csc</i>	Request CSS
<i>ADR</i>	Request Address
<i>ZIP</i>	Request Zip
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with `device_getResponseCodeString`

21.7.1.46 RETURN_CODE IDTechSDK.IDT_L100.pin_promptForAmountInput (string *message*, byte[] *signature*, bool *maskInput*, int *minLen*, int *maxLen*, string *ident* = " ")

Prompt for Amount Input from Encrypted Message

Prompts for amount input using the secure message data

```
@param message    Message
@param signature   Signature, 256 bytes
@param maskInput  Mask Input Entry
@param minLen     Minimum input length.  Cannot be less than 1
@param maxLen     Maximum input length.  Cannot be greater than 15
```

Timeout is hardwired to 3 minutes

Parameters

<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.
--------------	--

Returns

RETURN_CODE: Values can be parsed with `errorCode.getErrorString()`

21.7.1.47 RETURN_CODE IDTechSDK.IDT_L100.pin_promptForAmountInput (int *messageID*, int *languageID*, int *minLen*, int *maxLen*)

Prompt for Amount Input

Prompts for amount input using the secure message according to the following table

Msg Id	English Prompt	Portuguese Prompt	Spanish Prompt	French Prompt
1	ENTER	ENTER	INGRESE	ENTREZ
2	REENTER	RE-INTRODUZIR	REINGRESE	RE-ENTREZ
3	ENTER YOUR	INTRODUZIR O SEU	INGRESE SU	ENTREZ VOTRE
4	REENTER YOUR	RE-INTRODUZIR O SEU	REINGRESE SU	RE-ENTREZ VOTRE
5	PLEASE ENTER	POR FAVOR DIGITE	POR FAVOR INGRESE	SVP ENTREZ
6	PLEASE REENTER	POR FAVO REENTRAR	POR FAVO REINGRESE	SVP RE-ENTREZ
7	PO NUMBER	NÚMERO PO	NUMERO PO	No COMMANDE
8	DRIVER ID	LICENÇA	LICENCIA	ID CONDUCTEUR
9	ODOMETER	ODOMETER	ODOMETRO	ODOMETRE
10	ID NUMBER	NÚMERO ID	NUMERO ID	No IDENT
11	EQUIP CODE	EQUIP CODE	CODIGO EQUIP	CODE EQUIPEMENT
12	DRIVERS ID	DRIVER ID	ID CONDUCTOR	ID CONDUCTEUR
13	JOB NUMBER	EMP NÚMERO	NUMERO EMP	No TRAVAIL
14	WORK ORDER	TRABALHO ORDEM	ORDEN TRABAJO	FICHE TRAVAIL
15	VEHICLE ID	ID VEÍCULO	ID VEHICULO	ID VEHICULE
16	ENTER DRIVER	ENTER DRIVER	INGRESE CONDUCTOR	ENTR CONDUCTEUR
17	ENTER DEPT	ENTER DEPT	INGRESE DEPT	ENTR DEPARTEMNT
18	ENTER PHONE	ADICIONAR PHONE	INGRESE TELEFONO	ENTR No TELEPH
19	ENTER ROUTE	ROUTE ADD	INGRESE RUTA	ENTREZ ROUTE
20	ENTER FLEET	ENTER FROTA	INGRESE FLOTA	ENTREZ PARC AUTO
21	ENTER JOB ID	ENTER JOB ID	INGRESE ID TRABAJO	ENTR ID TRAVAIL
22	ROUTE NUMBER	NÚMERO PATH	RUTA NUMERO	No ROUTE
23	ENTER USER ID	ENTER USER ID	INGRESE ID USUARIO	ID UTILISATEUR
24	FLEET NUMBER	NÚMERO DE FROTA	FLOTA NUMERO	No PARC AUTO
25	ENTER PRODUCT	ADICIONAR PRODUCTO	INGRESE PRODUCTO	ENTREZ PRODUIT
26	DRIVER NUMBER	NÚMERO DRIVER	CONDUCTOR NUMERO	No CONDUCTEUR
27	ENTER LICENSE	ENTER LICENÇA	INGRESE LICENCIA	ENTREZ PERMIS
28	ENTER FLEET NO	ENTER NRO FROTA	INGRESE NRO FLOTA	ENT No PARC AUTO
29	ENTER CAR WASH	WASH ENTER	INGRESE LAVADO	ENTREZ LAVE-AUTO
30	ENTER VEHICLE	ENTER VEÍCULO	INGRESE VEHICULO	ENTREZ VEHICULE
31	ENTER TRAILER	TRAILER ENTER	INGRESE TRAILER	ENTREZ REMORQUE
32	ENTER ODOMETER	ENTER ODOMETER	INGRESE ODOMETRO	ENTREZ ODOMETRE
33	DRIVER LICENSE	CARTEIRA DE MOTORISTA	LICENCIA CONDUCTOR	PERMIS CONDUIRE
34	ENTER CUSTOMER	ENTER CLIENTE	INGRESE CLIENTE	ENTREZ CLIENT
35	VEHICLE NUMBER	NÚMERO DO VEÍCULO	VEHICULO NUMERO	No VEHICULE
36	ENTER CUST DATA	ENTER CLIENTE INFO	INGRESE INFO CLIENTE	INFO CLIENT
37	REENTER DRIVID	REENTRAR DRIVER ID	REINGRESE ID CHOFER	RE-ENTR ID COND

Msg Id	English Prompt	Portuguese Prompt	Spanish Prompt	French Prompt
38	ENTER USER DATA	ENTER INFO USUÁRIO	INGRESE INFO USUARIO	INFO UTILISATEUR
39	ENTER CUST CODE	ENTER CODE. CLIENTE	INGRESE COD. CLIENTE	ENTR CODE CLIENT
40	ENTER EMPLOYEE	ENTER FUNCIONÁRIO	INGRESE EMPLEADO	ENTREZ EMPLOYE
41	ENTER ID NUMBER	ENTER NÚMERO ID	INGRESE NUMERO ID	ENTREZ No ID
42	ENTER DRIVER ID	ENTER ID DRIVER	INGRESE ID CONDUCTOR	No CONDUCTEUR
43	ENTER FLEET PIN	ENTER PIN FROTA	INGRESE PIN DE FLOTA	NIP PARC AUTO
44	ODOMETER NUMBER	NÚMERO ODOMETER	ODOMETRO NUMERO	No ODOMETRE
45	ENTER DRIVER LIC	ENTER DRIVER LIC	INGRESE LIC CONDUCTOR	PERMIS CONDUIRE
46	ENTER TRAILER NO	NRO TRAILER ENTER	INGRESE NRO TRAILER	ENT No REMORQUE
47	REENTER VEHICLE	REENTRAR VEÍCULO	REINGRESE VEHICULO	RE-ENTR VEHICULE
48	ENTER VEHICLE ID	ENTER VEÍCULO ID	INGRESE ID VEHICULO	ENTR ID VEHICULE
49	ENTER BIRTH DATE	INSERIR DATA NAC	INGRESE FECHA NAC	ENT DT NAISSANCE
50	ENTER DOB MMDDYY	ENTER FDN MMDDYY	INGRESE FDN MMDDAA	NAISSANCE MMJJAA
51	ENTER FLEET DATA	ENTER FROTA INFO	INGRESE INFO DE FLOTA	INFO PARC AUTO
52	ENTER REFERENCE	ENTER REFERÊNCIA	INGRESE REFERENCIA	ENTREZ REFERENCE
53	ENTER AUTH NUMBER	ENTER NÚMERO AUT	INGRESE NUMERO AUT	No AUTORISATION
54	ENTER HUB NUMBER	ENTER HUB NRO	INGRESE NRO HUB	ENTREZ No NOYAU
55	ENTER HUBOMETER	MEDIDA PARA ENTRAR HUB	INGRESE MEDIDO DE HUB	COMPTEUR NOYAU
56	ENTER TRAILER ID	TRAILER ENTER ID	INGRESE ID TRAILER	ENT ID REMORQUE
57	ODOMETER READING	QUILOMETRAGEM	LECTURA ODOMETRO	LECTURE ODOMETRE
58	REENTER ODOMETER	REENTRAR ODOMETER	REINGRESE ODOMETRO	RE-ENT ODOMETRE
59	REENTER DRIV. ID	REENTRAR DRIVER ID	REINGRESE ID CHOFER	RE-ENT ID CONDUC
60	ENTER CUSTOMER ID	ENTER CLIENTE ID	INGRESE ID CLIENTE	ENTREZ ID CLIENT
61	ENTER CUST. ID	ENTER CLIENTE ID	INGRESE ID CLIENTE	ENTREZ ID CLIENT
62	ENTER ROUTE NUM	ENTER NUM ROUTE	INGRESE NUM RUTA	ENT No ROUTE
63	ENTER FLEET NUM	FROTA ENTER NUM	INGRESE NUM FLOTA	ENT No PARC AUTO
64	FLEET PIN	FROTA PIN	PIN DE FLOTA	NIP PARC AUTO
65	DRIVER #	DRIVER #	CONDUCTOR #	CONDUCTEUR
66	ENTER DRIVER #	ENTER DRIVER #	INGRESE CONDUCTOR #	ENT # CONDUCTEUR

Msg Id	English Prompt	Portuguese Prompt	Spanish Prompt	French Prompt
67	VEHICLE #	VEÍCULO #	VEHICULO #	# VEHICULE
68	ENTER VEHICLE #	ENTER VEÍCULO #	INGRESE VEHICULO #	ENT # VEHICULE
69	JOB #	TRABALHO #	TRABAJO #	# TRAVAIL
70	ENTER JOB #	ENTER JOB #	INGRESE TRABAJO #	ENTREZ # TRAVAIL
71	DEPT NUMBER	NÚMERO DEPT	NUMERO DEPTO	No DEPARTEMENT
72	DEPARTMENT #	DEPARTAMENTO #	DEPARTAMENTO #	DEPARTEMENT
73	ENTER DEPT #	ENTER DEPT #	INGRESE DEPTO #	ENT# DEPARTEMENT
74	LICENSE NUMBER	NÚMERO DE LICENÇA	NUMERO LICENCIA	No PERMIS
75	LICENSE #	LICENÇA #	LICENCIA #	# PERMIS
76	ENTER LICENSE #	ENTER LICENÇA #	INGRESE LICENCIA #	ENTREZ # PERMIS
77	DATA	INFO	INFO	INFO
78	ENTER DATA	ENTER INFO	INGRESE INFO	ENTREZ INFO
79	CUSTOMER DATA	CLIENTE INFO	INFO CLIENTE	INFO CLIENT
80	ID #	ID #	ID #	# ID
81	ENTER ID #	ENTER ID #	INGRESE ID #	ENTREZ # ID
82	USER ID	USER ID	ID USUARIO	ID UTILISATEUR
83	ROUTE #	ROUTE #	RUTA #	# ROUTE
84	ENTER ROUTE #	ADD ROUTE #	INGRESE RUTA #	ENTREZ # ROUTE
85	ENTER CARD NUM	ENTER NÚMERO DE CARTÃO	INGRESE NUM TARJETA	ENTREZ NO CARTE
86	EXP DATE(YMMM)	VALIDADE VAL (AAMM)	FECHA EXP (AAMM)	DATE EXPIR(AAMM)
87	PHONE NUMBER	TELEFONE	NUMERO TELEFONO	NO TEL
88	CVV START DATE	CVV DATA DE INÍCIO	CVV FECHA INICIO	CVV DATE DE DEBUT
89	ISSUE NUMBER	NÚMERO DE EMISSÃO	NUMERO DE EMISION	NO DEMISSION
90	START DATE (MMYY)	DATA DE INÍCIO (AAMM)	FECHA INICIO (AAMM)	DATE DE DEBUT-AAMM

```
@param messageID Message (1-90)
@param languageID 0=English Prompt, 1=Portuguese Prompt, 2=Spanish Prompt, 3=French Prompt
@param minLen Minimum input length. Cannot be less than 1
@param maxLen Maximum input length. Cannot be greater than 15
```

```
Timeout is hardwired to 3 minutes
```

```
@return RETURN_CODE: Values can be parsed with errorCode.getErrorString()
```

21.7.1.48 RETURN_CODE IDTechSDK.IDT_L100.pin_promptForKeyInput (string message, byte[] signature, bool maskInput, int minLen, int maxLen, string ident = " ")

Prompt for Key Input from Encrypted Message

Prompts for a numeric key using the secure encrypted message data

Parameters

<i>message</i>	Message
<i>signature</i>	Signature, 256 bytes

Parameters

<i>maskInput</i>	TRUE = entry is masked with '*', FALSE = entry is displayed on keypad
<i>minLen</i>	Minimum input length. Cannot be less than 1
<i>maxLen</i>	Maximum input length. Cannot be greater than 16

Integrators that desire to have custom messages for their application must submit the message request to IDTech with the following information (all fields required)

- Demo or Production Message Needed:
- Minimum PIN Requirement:
- Maximum PIN Requirement:
- Masking On or Off:
- Message Text:

The following data can be used as test data

- message: "Enter Ticket"
- minLen: 1
- maxLen: 16
- maskInput: False
- signature (demo unit): 80774ED82C8037776108A5E9E6B8F00070E7ADE1A3E6143B59B94EBC↵
B15712DB99ECA2A2193903F8FAEC83A516B2436F0C83EA143BB133620E6F94C7D47272C37657↵
A46D9B49783F5D3DDA46CAECAE168A28D9540D526F8A1D8EC50F1D051FFFC6A22622C99E46↵
A4A6178DA030D25F613119F1C8800A08933696E411742528415B5889E9ABD24BDF2E807A0518↵
C78E87B085218A3A18E291862CDDDA4B066EB1D4C0747D4800FD35D0DA47C250F7DB44146699↵
B853354081619C8B5FB3B66C293186588DA69CFD654202734408106FAD639EAF2932D8D8611EB↵
A2262728B4047AF1617370D29CDE9DD81B2B7934699F765E50017029A3739813686B194D12E2
- signature (prod unit): 8CE8E76C978871CCE1001010159F88E4968D8628213E28D9899C4103642608↵
E98AFBEC9BC29746B246E02A2C27566478E1F1C5FB9D0DB328D16A9A2F87216B93AA590D1261255↵
AAC9197007F2F4D18AF2E1DBECC5603339FCD31C12771138687E3608F7DCF835CE2DE8709E317↵
C7E12CD139792E2343683B9FB61819AB698D58A831D90596B3BD8DC1C0F5868E88C328D87C4489↵
E1AD378515B474C5DAEA876762F8D707610190175260BF6292AE910A7CB85C7386AEE30155038↵
F506BDC40D255F4ED269F8A2BA0D764BAC1146C35F7C18347204B74354FDDBC0A83DC2BF2BC34↵
E1A8161C209345122DC56F7963ACDD1183807CAA49D72B1F3CA79AB2CD12C2

Timeout is hardwired to 3 minutes

Parameters

<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.
--------------	--

Returns

RETURN_CODE: Values can be parsed with `errorCode.getErrorString()`

21.7.1.49 **RETURN_CODE** IDTechSDK.IDT_L100.pin_promptForKeyInput (int *messageID*, int *languageID*, bool *maskInput*, int *minLen*, int *maxLen*)

Prompt for Key Input

Prompts for a numeric key using the secure message according to the following table

Msg Id	English Prompt	Portuguese Prompt	Spanish Prompt	French Prompt
1	ENTER	ENTER	INGRESE	ENTREZ
2	REENTER	RE-INTRODUZIR	REINGRESE	RE-ENTREZ
3	ENTER YOUR	INTRODUZIR O SEU	INGRESE SU	ENTREZ VOTRE
4	REENTER YOUR	RE-INTRODUZIR O SEU	REINGRESE SU	RE-ENTREZ VOTRE
5	PLEASE ENTER	POR FAVOR DIGITE	POR FAVOR INGRESE	SVP ENTREZ
6	PLEASE REENTER	POR FAVO REENTRAR	POR FAVO REINGRESE	SVP RE-ENTREZ
7	PO NUMBER	NÚMERO PO	NUMERO PO	No COMMANDE
8	DRIVER ID	LICENÇA	LICENCIA	ID CONDUCTEUR
9	ODOMETER	ODOMETER	ODOMETRO	ODOMETRE
10	ID NUMBER	NÚMERO ID	NUMERO ID	No IDENT
11	EQUIP CODE	EQUIP CODE	CODIGO EQUIP	CODE EQUIPEMENT
12	DRIVERS ID	DRIVER ID	ID CONDUCTOR	ID CONDUCTEUR
13	JOB NUMBER	EMP NÚMERO	NUMERO EMP	No TRAVAIL
14	WORK ORDER	TRABALHO ORDEM	ORDEN TRABAJO	FICHE TRAVAIL
15	VEHICLE ID	ID VEÍCULO	ID VEHICULO	ID VEHICULE
16	ENTER DRIVER	ENTER DRIVER	INGRESE CONDUCTOR	ENTR CONDUCTEUR
17	ENTER DEPT	ENTER DEPT	INGRESE DEPT	ENTR DEPARTEMNT
18	ENTER PHONE	ADICIONAR PHONE	INGRESE TELEFONO	ENTR No TELEPH
19	ENTER ROUTE	ROUTE ADD	INGRESE RUTA	ENTREZ ROUTE
20	ENTER FLEET	ENTER FROTA	INGRESE FLOTA	ENTREZ PARC AUTO
21	ENTER JOB ID	ENTER JOB ID	INGRESE ID TRABAJO	ENTR ID TRAVAIL
22	ROUTE NUMBER	NÚMERO PATH	RUTA NUMERO	No ROUTE
23	ENTER USER ID	ENTER USER ID	INGRESE ID USUARIO	ID UTILISATEUR
24	FLEET NUMBER	NÚMERO DE FROTA	FLOTA NUMERO	No PARC AUTO
25	ENTER PRODUCT	ADICIONAR PRODUTO	INGRESE PRODUCTO	ENTREZ PRODUIT
26	DRIVER NUMBER	NÚMERO DRIVER	CONDUCTOR NUMERO	No CONDUCTEUR
27	ENTER LICENSE	ENTER LICENÇA	INGRESE LICENCIA	ENTREZ PERMIS
28	ENTER FLEET NO	ENTER NRO FROTA	INGRESE NRO FLOTA	ENT No PARC AUTO
29	ENTER CAR WASH	WASH ENTER	INGRESE LAVADO	ENTREZ LAVE-AUTO
30	ENTER VEHICLE	ENTER VEÍCULO	INGRESE VEHICULO	ENTREZ VEHICULE
31	ENTER TRAILER	TRAILER ENTER	INGRESE TRAILER	ENTREZ REMORQUE
32	ENTER ODOMETER	ENTER ODOMETER	INGRESE ODOMETRO	ENTREZ ODOMETRE

Msg Id	English Prompt	Portuguese Prompt	Spanish Prompt	French Prompt
33	DRIVER LICENSE	CARTEIRA DE MOTORISTA	LICENCIA CONDUTOR	PERMIS CONDUIRE
34	ENTER CUSTOMER	ENTER CLIENTE	INGRESE CLIENTE	ENTREZ CLIENT
35	VEHICLE NUMBER	NÚMERO DO VEÍCULO	VEHICULO NUMERO	No VEHICULE
36	ENTER CUST DATA	ENTER CLIENTE INFO	INGRESE INFO CLIENTE	INFO CLIENT
37	REENTER DRIVID	REENTRAR DRIVER ID	REINGRESE ID CHOFER	RE-ENTR ID COND
38	ENTER USER DATA	ENTER INFO USUÁRIO	INGRESE INFO USUARIO	INFO UTILISATEUR
39	ENTER CUST CODE	ENTER CODE. CLIENTE	INGRESE COD. CLIENTE	ENTR CODE CLIENT
40	ENTER EMPLOYEE	ENTER FUNCIONÁRIO	INGRESE EMPLEADO	ENTREZ EMPLOYE
41	ENTER ID NUMBER	ENTER NÚMERO ID	INGRESE NUMERO ID	ENTREZ No ID
42	ENTER DRIVER ID	ENTER ID DRIVER	INGRESE ID CONDUCTOR	No CONDUCTEUR
43	ENTER FLEET PIN	ENTER PIN FROTA	INGRESE PIN DE FLOTA	NIP PARC AUTO
44	ODOMETER NUMBER	NÚMERO ODOMETER	ODOMETRO NUMERO	No ODOMETRE
45	ENTER DRIVER LIC	ENTER DRIVER LIC	INGRESE LIC CONDUCTOR	PERMIS CONDUIRE
46	ENTER TRAILER NO	NRO TRAILER ENTER	INGRESE NRO TRAILER	ENT No REMORQUE
47	REENTER VEHICLE	REENTRAR VEÍCULO	REINGRESE VEHICULO	RE-ENTR VEHICULE
48	ENTER VEHICLE ID	ENTER VEÍCULO ID	INGRESE ID VEHICULO	ENTR ID VEHICULE
49	ENTER BIRTH DATE	INSERIR DATA NAC	INGRESE FECHA NAC	ENT DT NAISSANCE
50	ENTER DOB MMDDYY	ENTER FDN MMDDYY	INGRESE FDN MMDDAA	NAISSANCE MMJJAA
51	ENTER FLEET DATA	ENTER FROTA INFO	INGRESE INFO DE FLOTA	INFO PARC AUTO
52	ENTER REFERENCE	ENTER REFERÊNCIA	INGRESE REFERENCIA	ENTREZ REFERENCE
53	ENTER AUTH NUMBER	ENTER NÚMERO AUT	INGRESE NUMERO AUT	No AUTORISATION
54	ENTER HUB NUMBER	ENTER HUB NRO	INGRESE NRO HUB	ENTREZ No NOYAU
55	ENTER HUBOMETER	MEDIDA PARA ENTRAR HUB	INGRESE MEDIDO DE HUB	COMPTEUR NOYAU
56	ENTER TRAILER ID	TRAILER ENTER ID	INGRESE ID TRAILER	ENT ID REMORQUE
57	ODOMETER READING	QUILOMETRAGEM	LECTURA ODOMETRO	LECTURE ODOMETRE
58	REENTER ODOMETER	REENTRAR ODOMETER	REINGRESE ODOMETRO	RE-ENT ODOMETRE
59	REENTER DRIV. ID	REENTRAR DRIVER ID	REINGRESE ID CHOFER	RE-ENT ID CONDUC
60	ENTER CUSTOMER ID	ENTER CLIENTE ID	INGRESE ID CLIENTE	ENTREZ ID CLIENT

Msg Id	English Prompt	Portuguese Prompt	Spanish Prompt	French Prompt
61	ENTER CUST. ID	ENTER CLIENTE ID	INGRESE ID CLIEN↵ TE	ENTREZ ID CLIENT
62	ENTER ROUTE NUM	ENTER NUM ROUTE	INGRESE NUM RUTA	ENT No ROUTE
63	ENTER FLEET NUM	FROTA ENTER NUM	INGRESE NUM FLO↵ TA	ENT No PARC AUTO
64	FLEET PIN	FROTA PIN	PIN DE FLOTA	NIP PARC AUTO
65	DRIVER #	DRIVER #	CONDUCTOR #	CONDUCTEUR
66	ENTER DRIVER #	ENTER DRIVER #	INGRESE CONDUCT↵ TOR #	ENT # CONDUCTEUR
67	VEHICLE #	VEÍCULO #	VEHICULO #	# VEHICULE
68	ENTER VEHICLE #	ENTER VEÍCULO #	INGRESE VEHICULO #	ENT # VEHICULE
69	JOB #	TRABALHO #	TRABAJO #	# TRAVAIL
70	ENTER JOB #	ENTER JOB #	INGRESE TRABAJO #	ENTREZ # TRAVAIL
71	DEPT NUMBER	NÚMERO DEPT	NUMERO DEPTO	No DEPARTEMENT
72	DEPARTMENT #	DEPARTAMENTO #	DEPARTAMENTO #	DEPARTEMENT
73	ENTER DEPT #	ENTER DEPT #	INGRESE DEPTO #	ENT# DEPARTEME↵ NT
74	LICENSE NUMBER	NÚMERO DE LICE↵ NÇA	NUMERO LICENCIA	No PERMIS
75	LICENSE #	LICENÇA #	LICENCIA #	# PERMIS
76	ENTER LICENSE #	ENTER LICENÇA #	INGRESE LICENCIA #	ENTREZ # PERMIS
77	DATA	INFO	INFO	INFO
78	ENTER DATA	ENTER INFO	INGRESE INFO	ENTREZ INFO
79	CUSTOMER DATA	CLIENTE INFO	INFO CLIENTE	INFO CLIENT
80	ID #	ID #	ID #	# ID
81	ENTER ID #	ENTER ID #	INGRESE ID #	ENTREZ # ID
82	USER ID	USER ID	ID USUARIO	ID UTILISATEUR
83	ROUTE #	ROUTE #	RUTA #	# ROUTE
84	ENTER ROUTE #	ADD ROUTE #	INGRESE RUTA #	ENTREZ # ROUTE
85	ENTER CARD NUM	ENTER NÚMERO DE CARTÃO	INGRESE NUM TAR↵ JETA	ENTREZ NO CARTE
86	EXP DATE(Yymm)	VALIDADE VAL (AA↵ MM)	FECHA EXP (AAMM)	DATE EXPIR(AAMM)
87	PHONE NUMBER	TELEFONE	NUMERO TELEFONO	NO TEL
88	CVV START DATE	CVV DATA DE INÍCIO	CVV FECHA INICIO	CVV DATE DE DEB↵ UT
89	ISSUE NUMBER	NÚMERO DE EMIS↵ SÃO	NUMERO DE EMISI↵ ON	NO DEMISSION
90	START DATE (MmYy)	DATA DE INÍCIO (A↵ AMM)	FECHA INICIO (AA↵ MM)	DATE DE DEBUT-A↵ AMM

@param messageID Message (1-90)

@param languageID 0=English Prompt, 1=Portuguese Prompt, 2=Spanish Prompt, 3=French Prompt

@param maskInput TRUE = entry is masked with '*', FALSE = entry is displayed on keypad

@param minLen Minimum input length. Cannot be less than 1

@param maxLen Maximum input length. Cannot be greater than 16

Timeout is hardwired to 3 minutes

@return RETURN_CODE: Values can be parsed with errorCode.getErrorString()

21.7.1.50 `RETURN_CODE IDTechSDK.IDT_L100.pin_sendBeep (int frequency, int duration, string ident = " ")`

Send Beep

Executes a beep request.

Parameters

<i>frequency</i>	Frequency, range 200-20000Hz
<i>duration</i>	Duration, range 16-65535ms
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with `device_getResponseCodeString`

21.7.1.51 `RETURN_CODE IDTechSDK.IDT_L100.pin_setKeypressCapture (bool showKeyValue, string ident = " ")`

Set Keypress Capture Mode

If TRUE, each keypress will be broadcast with its value 0-9, B, C, E. Function completes after timeout, C, or E.
If FALSE, each keypress will generate a generic keypress notification, with final results being returned after timeout.

Parameters

<i>showKeyValue</i>	TRUE = broadcast each keypress value, FALSE = only broadcast a key was pressed, not it's value.
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with `errorCode.getErrorString()`

21.7.1.52 `static String IDTechSDK.IDT_L100.SDK_Version () [static]`

SDK Version

- All Devices

Returns the current version of SDK

Returns

Framework version

21.7.1.53 `static void IDTechSDK.IDT_L100.setCallback (Callback my_Callback) [static]`

Set Callback

Sets the class callback

21.7.1.54 `static void IDTechSDK.IDT_L100.setCallback (IntPtr my_Callback, SynchronizationContext context)` [static]

Set Callback

Sets the class callback

Parameters

<i>my_Callback</i>	The callback function to receive the response message from device. defined as follows. public unsafe delegate void MFCCallBack(Parameters parameters, ident);
<i>context</i>	The context of the UI thread

21.7.1.55 `static void IDTechSDK.IDT_L100.setCommandTimeout (int milliseconds, string ident = " ")` [static]

Set Command Timeout

Sets the default timeout (in milliseconds) waiting for a blocking command response

Parameters

<i>milliseconds</i>	Time
---------------------	------

21.7.1.56 `static bool IDTechSDK.IDT_L100.useSerialPort (int port)` [static]

Use Serial Port Interface

Instructs SDK to attempt to use the Serial Port for communication with L100 using default baud rate

Parameters

<i>port</i>	Serial Port to use. Example COM1 = 1.
-------------	---------------------------------------

Returns

bool TRUE=successful, FALSE=failure

21.7.1.57 `static bool IDTechSDK.IDT_L100.useSerialPort (int port, int baud, string ident = " ")` [static]

Use Serial Port Interface with baud rate L100

Parameters

<i>port</i>	Serial Port to use. Example COM1 = 1.
<i>baud</i>	Baud rate to override default. Example 115200;

Returns

bool TRUE=successful, FALSE=failure

21.7.1.58 `static bool IDTechSDK.IDT_L100.useSerialPortLinux (string path)` [static]

Use Serial Port Interface on Linux

Instructs SDK to attempt to use the Serial Port for communication with BTMag using default baud rate on Linux implementations

Parameters

<i>path</i>	Path to use. Example /dev/ttyUSB*
-------------	-----------------------------------

Returns

bool TRUE=successful, FALSE=failure

21.7.1.59 `static bool IDTechSDK.IDT_L100.useSerialPortLinux (string path, int baud) [static]`

Use Serial Port Interface on Linux with baud rate

Instructs SDK to attempt to use the Serial Port for communication with BTPay on Linux implementations

Parameters

<i>path</i>	Path to use. Example /dev/ttyUSB*
<i>baud</i>	Baud rate to override default. Example 115200;

Returns

bool TRUE=successful, FALSE=failure

21.7.2 Property Documentation

21.7.2.1 `IDT_L100 IDTechSDK.IDT_L100.SharedController [static], [get]`

Singleton Instance

Establishes an singleton instance of [IDT_L100](#) class.

Returns

Instance of [IDT_L100](#)

The documentation for this class was generated from the following file:

- /Users/randy/Repo/universal-sdk/IDTechSDK_STD/DeviceHeaders/IDT_L100.cs

21.8 IDTechSDK.IDT_L80 Class Reference

Public Member Functions

- RETURN_CODE [device_getFirmwareVersion](#) (ref string response, string ident="")
- RETURN_CODE [device_PKI_RKI](#) (bool isProduction, string ident="", bool performOnForeground=false)
- RETURN_CODE [device_updateDeviceFirmware](#) (byte[] firmwareData, string ident="")
- RETURN_CODE [device_sendPAE](#) (string command, ref string response, int timeout, string ident="")
- RETURN_CODE [device_sendDataCommand](#) (string cmd, bool calcLRC, ref byte[] response, string ident="")
- RETURN_CODE [device_sendDataCommand_ext](#) (string cmd, bool calcLRC, ref byte[] response, int timeout, bool noResponse, string ident="")

- RETURN_CODE [config_getModelNumber](#) (ref string response, string ident="")
- RETURN_CODE [config_getSerialNumber](#) (ref string response, string ident="")
- bool [config_setCmdTimeOutDuration](#) (int newTimeOut, string ident="")
- RETURN_CODE [pin_getEncryptedPIN](#) (int keyType, string PAN, string message, int timeout, bool isAES=false, string ident="")
- RETURN_CODE [pin_getManualPanEntry](#) (bool csc, bool ADR, bool ZIP, string ident="")
- RETURN_CODE [pin_getFunctionKey](#) (string ident="")
- RETURN_CODE [pin_sendBeep](#) (int frequency, int duration, string ident="")
- RETURN_CODE [device_rebootDevice](#) (string ident="")
- RETURN_CODE [pin_setKeyPressCapture](#) (bool showKeyValue, string ident="")
- RETURN_CODE [pin_cancelPINEntry](#) (string ident="")
- RETURN_CODE [device_getKeyStatus](#) (ref byte[] status, string ident="")
- RETURN_CODE [pin_promptForKeyInput](#) (string message, byte[] signature, bool maskInput, int minLen, int maxLen, string ident="")
- RETURN_CODE [pin_promptForAmountInput](#) (string message, byte[] signature, bool maskInput, int minLen, int maxLen, string ident="")
- RETURN_CODE [device_setSleepModeTime](#) (int time, string ident="")
- RETURN_CODE [device_enterStopMode](#) (string ident="")
- RETURN_CODE [device_setDateTime](#) (string ident="")
- RETURN_CODE [device_getDateTime](#) (ref byte[] dateTime, string ident="")
- RETURN_CODE [lcd_clearDisplay](#) (int lineNumber, string ident="")
- RETURN_CODE [lcd_clearAllLines](#) (string ident="")
- RETURN_CODE [lcd_savePrompt](#) (int promptNumber, string prompt, string ident="")
- RETURN_CODE [lcd_displayPrompt](#) (int promptNumber, int lineNumber, string ident="")
- RETURN_CODE [lcd_displayMessage](#) (int lineNumber, string message, string ident="")
- RETURN_CODE [lcd_enableBacklight](#) (bool enable, string ident="")
- RETURN_CODE [lcd_getBacklightStatus](#) (ref bool enabled, string ident="")
- RETURN_CODE [config_setBaudRate](#) (int baud, string ident="")
- RETURN_CODE [config_getBaudRate](#) (ref int baud, string ident="")
- RETURN_CODE [device_getRKIStatus](#) (bool isProd, bool isMultiKey, ref string status, ref Dictionary< string, RKI_KEY_TYPE > keys, string ident="")
- RETURN_CODE [device_getAnyRKIStatus](#) (bool isMultiKey, ref string status, ref Dictionary< string, RKI_KEY_TYPE > keys, string ident="")
- RETURN_CODE [device_updateFirmwareFromZip](#) (byte[] zipfile, string ident="", bool isForeground=false)
- RETURN_CODE [device_sendConfigurationFromZip](#) (byte[] zip, string filename, VC_OPERATION_TYPE type, bool matchFW, string ident="", bool isForeground=false)
- RETURN_CODE [device_sendConfiguration](#) (string filename, VC_OPERATION_TYPE type, bool matchFW, string ident="", bool isForeground=false)
- RETURN_CODE [device_readConfigurationToMemory](#) (string memo, Configs.InstallRules rules, List< Configs.DeviceCommand > cmds, string ident="", bool isForeground=false)
- RETURN_CODE [device_getConfigurationFromMemory](#) (ref string json, string ident="")
- RETURN_CODE [device_RemoteKeyInjection](#) (RKI_KEY_TYPE type, string keyName, string ident="", bool performOnForeground=false)

Static Public Member Functions

- static bool [useSerialPort](#) (int port)
- static bool [useSerialPortLinux](#) (string path)
- static bool [useSerialPortLinux](#) (string path, int baud)
- static int [getCommandTimeout](#) (string ident="")
- static void [setCommandTimeout](#) (int milliseconds, string ident="")
- static bool [useSerialPort](#) (int port, int baud, string ident="")
- static bool [closeSerialPort](#) ()
- static bool [closeUSB](#) (string ident="")
- static void [initSC](#) ()

- static void [setCallback](#) (Callback my_Callback)
- static void [setCallback](#) (IntPtr my_Callback, SynchronizationContext context)
- static String [SDK_Version](#) ()
- static void [lcd_retrieveMessage](#) (DisplayMessages.DISPLAY_MESSAGE_LANGUAGE lang, DisplayMessages.DISPLAY_MESSAGE_IDENTIFIER id, ref string line1, ref string line2)

Properties

- static [IDT_L80 SharedController](#) [get]

21.8.1 Member Function Documentation

21.8.1.1 static bool IDTechSDK.IDT_L80.closeSerialPort () [static]

Close Serial Port Interface

Instructs SDK to close the Serial Port if connected to L80

Returns

bool TRUE=successful, FALSE=failure

21.8.1.2 static bool IDTechSDK.IDT_L80.closeUSB (string ident = " ") [static]

Close USB

Instructs SDK to close the USB if connected to L80

Returns

bool TRUE=successful, FALSE=failure

21.8.1.3 RETURN_CODE IDTechSDK.IDT_L80.config_getBaudRate (ref int baud, string ident = " ")

Get Baud Rate

Gets the buad rate for RS-232 communication.

Parameters

<i>baud</i>	<ul style="list-style-type: none"> • 2 = 2400 • 3 = 4800 • 4 = 9600 • 6 = 19200 • 7 = 38400 • 9 = 115200
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString

21.8.1.4 RETURN_CODE IDTechSDK.IDT_L80.config_getModelNumber (ref string *response*, string *ident* = " ")

Polls device for Model Number

Parameters

<i>response</i>	Returns Model Number
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with errorCode.getErrorString()

21.8.1.5 RETURN_CODE IDTechSDK.IDT_L80.config_getSerialNumber (ref string *response*, string *ident* = " ")

Polls device for Serial Number

Parameters

<i>response</i>	Returns Serial Number
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with errorCode.getErrorString()

21.8.1.6 RETURN_CODE IDTechSDK.IDT_L80.config_setBaudRate (int *baud*, string *ident* = " ")

Set Baud Rate

Sets the buad rate for RS-232 communication.

Parameters

<i>baud</i>	<ul style="list-style-type: none">• 2 = 2400• 3 = 4800• 4 = 9600• 6 = 19200• 7 = 38400• 9 = 115200
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString

21.8.1.7 `bool IDTechSDK.IDT_L80.config_setCmdTimeOutDuration (int newTimeOut, string ident = " ")`

Command Acknowledgement Timeout

Sets the amount of seconds to wait for an {ACK} to a command before a timeout. Responses should normally be received under one second. Default is 3 seconds

Parameters

<i>newTimeOut</i>	Timeout value. Valid range 1 - 60 seconds
-------------------	---

Returns

Success flag. Determines if value was set and in range.

21.8.1.8 `RETURN_CODE IDTechSDK.IDT_L80.device_enterStopMode (string ident = " ")`

Enter Stop Mode

Set device enter to stio mode. In stop mode, LCD display and backlight is off. Stop mode reduces power consumption to the lowest possible level. A unit in stop mode can only be woken up by a physical key press.

Parameters

<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.
--------------	--

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString

21.8.1.9 `RETURN_CODE IDTechSDK.IDT_L80.device_getAnyRKIStatus (bool isMultiKey, ref string status, ref Dictionary< string, RKI_KEY_TYPE > keys, string ident = " ")`

Get RKI Status Polls the RKI servers to validate device RKI eligibility. Note: if device type is known in advance (production or demo device), it is more efficient to use device_getRKIStatus instead

Parameters

<i>isMultiKey</i>	True = MultiKey, False = SingleKey
<i>status</i>	RKI status returned as a string
<i>keys</i>	Returns all available keys
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with errorCode.getErrorString(). When no data is available, return code = RETURN_CODE_NO_DATA_AVAILABLE

21.8.1.10 RETURN_CODE IDTechSDK.IDT_L80.device_getConfigurationFromMemory (ref string *json*, string *ident* = " ")

Read Device Configuration From Memory Retrieves the ViVOconfig configuration data from memory after a successful device_readConfigurationToMemory

Parameters

<i>json</i>	Returns the .json file data that can be saved to disk
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with `errorCode.getErrorString()`. When no data is available, return code = RETURN_CODE_NO_DATA_AVAILABLE

21.8.1.11 RETURN_CODE IDTechSDK.IDT_L80.device_getDateTime (ref byte[] *dateTime*, string *ident* = " ")

Get Date Time

Gets current system date and time of the device.

Parameters

<i>dateTime</i>	<p>The date time returned as follows:</p> <ul style="list-style-type: none"> • byte 0: Year 00-99 • byte 1: Month 01-12 • byte 2: Date 01-31 • byte 3: Hour 00-23 • byte 4: Minute 00-59 • byte 5: Second 00-59
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with `device_getResponseCodeString`

21.8.1.12 RETURN_CODE IDTechSDK.IDT_L80.device_getFirmwareVersion (ref string *response*, string *ident* = " ")

Polls device for Firmware Version

Parameters

<i>response</i>	Response returned of Firmware Version
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with `errorCode.getErrorString()`

21.8.1.13 RETURN_CODE IDTechSDK.IDT_L80.device_getKeyStatus (ref byte[] status, string ident = " ")

Get Key Status

Gets the status of loaded keys

Parameters

<i>status</i>	byte 0: PIN DUKPT Key, 1 Exists, 0 None, 0xFF STOP byte 1: PIN Master Key, 1 Exists, 0 None byte 2: PIN Session Key, 1 Exists, 0 None byte 3: Account/MSR DUKPT Key, Does not support, always 0 byte 4: Account/ICC DUKPT Key, Does not support, always 0 byte 5: Admin DUKPT Key, 1 Exists, 0 None, 0xFF STOP
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with `errorCode.getErrorString()`

21.8.1.14 RETURN_CODE IDTechSDK.IDT_L80.device_getRKIStatus (bool isProd, bool isMultiKey, ref string status, ref Dictionary< string, RKI_KEY_TYPE > keys, string ident = " ")

Get RKI Status From Specified Server Polls the RKI server to validate device RKI eligibility

Parameters

<i>isProd</i>	TRUE = poll production RKI server, FALSE = poll demo RKI server
<i>isMultiKey</i>	True = MultiKey, False = SingleKey
<i>status</i>	RKI status returned as a string
<i>keys</i>	Returns dictionary of available keys
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with `errorCode.getErrorString()`. When no data is available, return code = RETURN_CODE_NO_DATA_AVAILABLE

21.8.1.15 RETURN_CODE IDTechSDK.IDT_L80.device_PKI_RKI (bool isProduction, string ident = " ", bool performOnForeground = false)

Start PKI Remote Key Injection

Starts a PKI remote key injection request with IDTech RKI servers. Set/Get RKI url with `IDT_Device.RKI_URL`.

Parameters

<i>isProduction</i>	TRUE = Production RKI, FALSE = Demo RKI
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.
<i>performOnForeground</i>	TRUE = block until done

Returns

RETURN_CODE: Values can be parsed with `device_getResponseCodeString`

21.8.1.16 **RETURN_CODE** IDTechSDK.IDT_L80.device_readConfigurationToMemory (string *memo*, Configs.InstallRules *rules*, List< Configs.DeviceCommand > *cmds*, string *ident* = " ", bool *isForeground* = false)

Read Device Configuration To Memory Executes a ViVOconfig read of device settings and stores the results in memory

Once a ViVOconfig read starts, all messages will be returned to the callback as DeviceState.ViVOconfig. The ViVOconfig operation is complete when the return code from a DeviceState.ViVOconfig message is one of the following values:

- RETURN_CODE_SDK_VIVOCONFIG_READ_CONFIG_SUCCESS = Read process completed successfully. Execute device_getConfigurationFromMemory to retrieve the configuration data
- RETURN_CODE_SDK_VIVOCONFIG_READ_CONFIG_FAILED = Read process FAILED.

Parameters

<i>memo</i>	Optional: populates the memo field of the captured .json file
<i>rules</i>	Optional: populates the .json file with available install rules
<i>cmds</i>	Optional: populates the .json file with commands to execute
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.
<i>isForeground</i>	If TRUE, will block program until update complete. Otherwise, FALSE performs update on background.

Returns

RETURN_CODE: Values can be parsed with errorCode.getErrorString(). When no data is available, return code = RETURN_CODE_NO_DATA_AVAILABLE

21.8.1.17 **RETURN_CODE** IDTechSDK.IDT_L80.device_rebootDevice (string *ident* = " ")

Reboot Device

Executes a command to restart the device.

- Card data is cleared, resetting card status bits.
- Response data of the previous command is cleared.
- Resetting firmware.

Parameters

<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.
--------------	--

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString

21.8.1.18 **RETURN_CODE** IDTechSDK.IDT_L80.device_RemoteKeyInjection (RKI_KEY_TYPE *type*, string *keyName*, string *ident* = " ", bool *performOnForeground* = false)

Remote Key Injection Performs a remote key injection for the device

Parameters

<i>type</i>	Remote Key Injection Type
<i>keyName</i>	Name of key (optional)
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.
<i>performOnForeground</i>	TRUE = block during process (default is false)

Returns

RETURN_CODE: Values can be parsed with `errorCode.getErrorString()`. When no data is available, return code = RETURN_CODE_NO_DATA_AVAILABLE

21.8.1.19 RETURN_CODE IDTechSDK.IDT_L80.device_sendConfiguration (string filename, VC_OPERATION_TYPE type, bool matchFW, string ident = " ", bool isForeground = false)

Send Configuration Executes a ViVOconfig update or verify to a device

Once a ViVOconfig write or verify starts, all messages will be returned to the callback as DeviceState.ViVOconfig. The ViVOconfig operation is complete when the return code from a DeviceState.ViVOconfig message is one of the following values:

- RETURN_CODE_SDK_VIVOCONFIG_VERIFY_SUCCESS = Verification process completed successfully. No differences found.
- RETURN_CODE_SDK_VIVOCONFIG_VERIFY_SUCCESS_WITH_WARNING = Verification process completed with warnings.
- RETURN_CODE_SDK_VIVOCONFIG_VERIFY_FAILED = Verification process FAILED
- RETURN_CODE_SDK_VIVOCONFIG_WRITE_SUCCESS = Write process completed successfully.
- RETURN_CODE_SDK_VIVOCONFIG_WRITE_SUCCESS_WITH_WARNING = Write process completed with warnings
- RETURN_CODE_SDK_VIVOCONFIG_WRITE_FAILED = Write process FAILED

Parameters

<i>filename</i>	The .json configuration file to update the device with NOTE: You can also use this parameter to pass the path of where the .json file is located at on local storage instead
<i>filename</i>	The name of the .json configuration file
<i>type</i>	VC_OPERATION_TYPE

- VC_OPERATION_TYPE_WRITE = Write File To Device, Hash must validate
- VC_OPERATION_TYPE_VERIFY = Verify Device With File
- VC_OPERATION_TYPE_WRITE_IGNORE_HASH = Write File To Device, Ignore Hash
- VC_OPERATION_TYPE_WRITE_FIX_HASH = Write File To Device, Fix Hash If Necessary

Parameters

<i>matchFW</i>	TRUE = Device FW must match file FW, FALSE = Don't validate device FW against file
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.
<i>isForeground</i>	If TRUE, will block program until update complete. Otherwise, FALSE performs update on background.

Returns

RETURN_CODE: Values can be parsed with `errorCode.getErrorString()`. When no data is available, return code = RETURN_CODE_NO_DATA_AVAILABLE

21.8.1.20 RETURN_CODE IDTechSDK.IDT_L80.device_sendConfigurationFromZip (byte[] *zip*, string *filename*, VC_OPERATION_TYPE *type*, bool *matchFW*, string *ident* = "", bool *isForeground* = false)

Send Configuration From Zip Executes a ViVOconfig update or verify to a device using files from a .zip archive

Once a ViVOconfig write or update starts, all messages will be returned to the callback as DeviceState.ViVOconfig. The ViVOconfig operation is complete when the return code from a DeviceState.ViVOconfig message is one of the following values:

- RETURN_CODE_SDK_VIVOCONFIG_VERIFY_SUCCESS = Verification process completed successfully. No differences found.
- RETURN_CODE_SDK_VIVOCONFIG_VERIFY_SUCCESS_WITH_WARNING = Verification process completed with warnings.
- RETURN_CODE_SDK_VIVOCONFIG_VERIFY_FAILED = Verification process FAILED
- RETURN_CODE_SDK_VIVOCONFIG_WRITE_SUCCESS = Write process completed successfully.
- RETURN_CODE_SDK_VIVOCONFIG_WRITE_SUCCESS_WITH_WARNING = Write process completed with warnings
- RETURN_CODE_SDK_VIVOCONFIG_WRITE_FAILED = Write process FAILED

Parameters

<i>zip</i>	.Zip archive containing the .json configuration file and also any associated image assets NOTE: You can also use this parameter to pass the path of where the .zip file is located at on local storage instead
<i>filename</i>	The name of the .json configuration file
<i>type</i>	VC_OPERATION_TYPE

- VC_OPERATION_TYPE_WRITE = Write File To Device, Hash must validate
- VC_OPERATION_TYPE_VERIFY = Verify Device With File
- VC_OPERATION_TYPE_WRITE_IGNORE_HASH = Write File To Device, Ignore Hash
- VC_OPERATION_TYPE_WRITE_FIX_HASH = Write File To Device, Fix Hash If Necessary

Parameters

<i>matchFW</i>	TRUE = Device FW must match file FW, FALSE = Don't validate device FW against file
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.
<i>isForeground</i>	If TRUE, will block program until update complete. Otherwise, FALSE performs update on background.

Returns

RETURN_CODE: Values can be parsed with `errorCode.getErrorString()`. When no data is available, return code = RETURN_CODE_NO_DATA_AVAILABLE

21.8.1.21 **RETURN_CODE** IDTechSDK.IDT_L80.device_sendDataCommand (string *cmd*, bool *calcLRC*, ref byte[] *response*, string *ident* = " ")

Send a data command to the device

Sends a command to the device.

Parameters

<i>cmd</i>	String representation of command to execute
<i>calcLRC</i>	If TRUE, this will wrap command with start/length/lrc/sum/end: '{STX}{Len_Low}{Len_High} data {CheckLRC} {CheckSUM} {ETX}'
<i>response</i>	Response data
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString

21.8.1.22 **RETURN_CODE** IDTechSDK.IDT_L80.device_sendDataCommand_ext (string *cmd*, bool *calcLRC*, ref byte[] *response*, int *timeout*, bool *noResponse*, string *ident* = " ")

Send a data command to the device - extended

Sends a command to the device.

Parameters

<i>cmd</i>	String representation of command to execute
<i>calcLRC</i>	If TRUE, this will wrap command as NGA with start/length/lrc/sum/end: '{STX}{Len_Low}{Len_High} data {CheckLRC} {CheckSUM} {ETX}'
<i>response</i>	Response data
<i>timeout</i>	Timeout value waiting for response, in milliseconds (1000 = 1 second)
<i>noResponse</i>	if TRUE, this will not wait for a response and immediately return SUCCESS
<i>calcITP</i>	If TRUE, this will wrap command as ITP with start/end/lrc: '{STX} data {ETX}{CheckLRC}'
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString

21.8.1.23 **RETURN_CODE** IDTechSDK.IDT_L80.device_sendPAE (string *command*, ref string *response*, int *timeout*, string *ident* = " ")

Send Payment Application Engine Command

Executes a PAE command

Parameters

<i>command</i>	ASCII command string, should start with "*PAE"
<i>response</i>	command response
<i>timeout</i>	timeout waiting for PAE response

Parameters

<i>ip</i>	Optional IP address when connected via TCP/IP
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with `errorCode.getErrorString()`

21.8.1.24 RETURN_CODE IDTechSDK.IDT_L80.device_setDateTime (string *ident* = " ")

Set Date Time

Set current system date and time to the device.

Parameters

<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.
--------------	--

Returns

RETURN_CODE: Values can be parsed with `device_getResponseCodeString`

21.8.1.25 RETURN_CODE IDTechSDK.IDT_L80.device_setSleepModeTime (int *time*, string *ident* = " ")

Set Sleep Mode Timer

Set device enter to sleep mode after the given time. In sleep mode, LCD display and backlight is off. Sleep mode reduces power consumption to the lowest possible level. A unit in Sleep mode can only be woken up by a physical key press.

Parameters

<i>time</i>	Enter sleep time value, in second.
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with `device_getResponseCodeString`

21.8.1.26 RETURN_CODE IDTechSDK.IDT_L80.device_updateDeviceFirmware (byte[] *firmwareData*, string *ident* = " ")

Update Firmware

Updates the firmware .

Parameters

<i>firmwareData</i>	Signed binary data of a firmware file provided by IDTech
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with `errorCode.getErrorString()`

After you pass the `firmwareData` file, a new thread will start to execute the firmware download. You will receive status of the progress through callbacks to the `IDTechSDK.Callback()` delegate. The following parameters will be passed back:

- `sender = IDT_DEVICE_Types.IDT_DEVICE_AUGUSTA`
- `state = DeviceState.FirmwareUpdate`
- `transactionResultCode = status of the firmware update (starting, entering bootloader, applying update, block success, firmware success)`
- `data = File Progress. Four bytes, with bytes [0][1] = current block, and bytes [2][3] = total blocks. 0x00030010 = block 3 of 16`

Example code starting a firmware update

```
OpenFileDialog diag = new OpenFileDialog(ident);

diag.Filter = "NGA FW Files|*.fm";

if (diag.ShowDialog() == DialogResult.OK)
{
    byte[] file = File.ReadAllBytes(diag.FileName, ident);
    RETURN_CODE rt = IDT_Device.SharedController.device_updateDeviceFirmware(file, ident);
    if (rt == RETURN_CODE.RETURN_CODE_DO_SUCCESS)
    {
        //Was a success
    }
    else
    {
        //Error starting firmware download
    }
}
```

Example monitoring firmware update status / success

```
private void MessageCallBack(IDTechSDK.IDT_DEVICE_Types type, DeviceState state, byte[] data,
    IDTTransactionData cardData, EMV_Callback emvCallback, RETURN_CODE transactionResultCode, string ident = "")
{
    switch (state)
    {
        case DeviceState.FirmwareUpdate:
            switch (transactionResultCode)
            {
                case RETURN_CODE.RETURN_CODE_FW_STARTING_UPDATE:
                    SetOutputText("Starting Firmware Update\n", ident);
                    break;
                case RETURN_CODE.RETURN_CODE_DO_SUCCESS:
                    SetOutputText("Firmware Update Successful\n", ident);
                    break;
                case RETURN_CODE.RETURN_CODE_APPLYING_FIRMWARE_UPDATE:
                    SetOutputText("Applying Firmware Update...\n", ident);
                    break;
                case RETURN_CODE.RETURN_CODE_ENTERING_BOOTLOADER_MODE:
                    SetOutputText("Entering Bootloader Mode...\n", ident);
                    break;
                case RETURN_CODE.RETURN_CODE_BLOCK_TRANSFER_SUCCESS:
                    int start = data[0] * 0x100 + data[1];
                    int end = data[2] * 0x100 + data[3];

                    SetOutputText("Sent block " + start.ToString() + " of " + end.ToString() + "\n", ident);
                    break;
                default:
                    SetOutputText("Firmware Update Error Code: " + "0x" + String.Format("{0:X}", (ushort)
                        transactionResultCode) + ": " + IDTechSDK.errorCode.getErrorString(transactionResultCode) + "\r\n",
                        ident);
                    break;
            }
        break;
    }
}
```

21.8.1.27 `RETURN_CODE IDTechSDK.IDT_L80.device_updateFirmwareFromZip (byte[] zipfile, string ident = " ", bool isForeground = false)`

Update Firmware From Zip

Performs one or more device firmware updates from firmware files passed as a compressed archive with a valid configuration file (.json format)

Parameters

<i>zipfile</i>	Zip file containing on or more firmware update files and a .json configuration file
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.
<i>isForeground</i>	If TRUE, will block program until firmware update complete. Otherwise, FALSE performs FW update on background.

Returns

`RETURN_CODE`: Values can be parsed with `errorCode.getErrorString(string ident = "")`

21.8.1.28 `static int IDTechSDK.IDT_L80.getCommandTimeout (string ident = " ") [static]`

Get Command Timeout

Gets the default timeout (in milliseconds) waiting for a blocking command response

Return values

<i>time</i>	Time
-------------	------

21.8.1.29 `RETURN_CODE IDTechSDK.IDT_L80.lcd_clearAllLines (string ident = " ")`

Clear LCD Display

Clears all lines of the LCD Display.

Parameters

<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.
--------------	--

Returns

`RETURN_CODE`: Values can be parsed with `device_getResponseCodeString`

21.8.1.30 `RETURN_CODE IDTechSDK.IDT_L80.lcd_clearDisplay (int lineNumber, string ident = " ")`

Clear LCD Display Line

Clears the line number of the LCD Display.

Parameters

<i>lineNumber</i>	Line number to clear (1-4)
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString

21.8.1.31 RETURN_CODE IDTechSDK.IDT_L80.lcd_displayMessage (int *lineNumber*, string *message*, string *ident* = " ")

Display Message on Line

Displays a message on a display line.

Parameters

<i>lineNumber</i>	Line number to display message on (1-4)
<i>message</i>	Message to display
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString

21.8.1.32 RETURN_CODE IDTechSDK.IDT_L80.lcd_displayPrompt (int *promptNumber*, int *lineNumber*, string *ident* = " ")

Display Prompt on Line

Displays a message prompt from L80 memory.

Parameters

<i>promptNumber</i>	Prompt number (0-9)
<i>lineNumber</i>	Line number to display message prompt (1-4)
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString

21.8.1.33 RETURN_CODE IDTechSDK.IDT_L80.lcd_enableBacklight (bool *enable*, string *ident* = " ")

Enable/Disable LCD Backlight

Turns on/off the LCD back lighting.

Parameters

<i>enable</i>	TRUE = turn ON backlight, FALSE = turn OFF backlight
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString

21.8.1.34 RETURN_CODE IDTechSDK.IDT_L80.lcd_getBacklightStatus (ref bool *enabled*, string *ident* = " ")

Get Backlight Status

Returns the status of the LCD back lighting.

Parameters

<i>enabled</i>	TRUE = Backlight is ON, FALSE = Backlight is OFF
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString

21.8.1.35 static void IDTechSDK.IDT_L80.lcd_retrieveMessage (DisplayMessages.DISPLAY_MESSAGE_LANGUAGE *lang*, DisplayMessages.DISPLAY_MESSAGE_IDENTIFIER *id*, ref string *line1*, ref string *line2*) [static]

Retrieve LCD Message

Returns the string value for a message ID returned for LCD messaging

Parameters

<i>lang</i>	Language.
<i>id</i>	Message ID
<i>line1</i>	Line 1 string value
<i>line2</i>	Line 2 string value
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString

21.8.1.36 RETURN_CODE IDTechSDK.IDT_L80.lcd_savePrompt (int *promptNumber*, string *prompt*, string *ident* = " ")

Save Prompt

Saves a message prompt to L80 memory.

Parameters

<i>promptNumber</i>	Prompt number (0-9)
<i>prompt</i>	Prompt string (up to 20 characters)
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString

21.8.1.37 RETURN_CODE IDTechSDK.IDT_L80.pin_cancelPINEntry (string *ident* = " ")

Cancel PIN Entry

Cancel "Get Function Key" & "Get Encrypted PIN" & "Get Numeric" & "Get Amount"

Parameters

<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.
--------------	--

Returns

RETURN_CODE: Values can be parsed with `errorCode.getErrorString()`

21.8.1.38 RETURN_CODE IDTechSDK.IDT_L80.pin_getEncryptedPIN (int *keyType*, string *PAN*, string *message*, int *timeout*, bool *isAES* = *false*, string *ident* = " ")

Get Encrypted PIN

Requests PIN Entry

Parameters

<i>keyType</i>	<ul style="list-style-type: none"> • 0x00- MKSK-TDES: External Plaintext PAN • 0x01- DUKPT-TDES: External Plaintext PAN • 0x10 MKSK-TDES: External Ciphertext PAN • 0x11 DUKPT-TDES: External Ciphertext PAN
<i>PAN</i>	Account Number
<i>message</i>	Message to display = timeout
<i>isAES</i>	set to TRUE if PEK is AES, not TDES
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with `errorCode.getErrorString()`

21.8.1.39 RETURN_CODE IDTechSDK.IDT_L80.pin_getFunctionKey (string *ident* = " ")

Get Function Key

Captures a function key

- Backspace = B
- Cancel = C
- Enter = E
- * = *
- # = #

- Help = ?
- Function Key 1 = F1
- Function Key 2 = F2
- Function Key 3 = F3

Timeout hard wired 3 minutes

Parameters

<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.
--------------	--

Returns

RETURN_CODE: Values can be parsed with `errorCode.getErrorString()`

21.8.1.40 RETURN_CODE IDTechSDK.IDT_L80.pin_getManualPanEntry (bool *csc*, bool *ADR*, bool *ZIP*, string *ident* = " ")

Get Manual Pan Entry

prompt the user to manually enter a card PAN and Expiry Date (and optionally CSC) from the keypad and return it to the POS.

Parameters

<i>csc</i>	Request CSS
<i>ADR</i>	Request Address
<i>ZIP</i>	Request Zip
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with `device.getResponseCodeString`

21.8.1.41 RETURN_CODE IDTechSDK.IDT_L80.pin_promptForAmountInput (string *message*, byte[] *signature*, bool *maskInput*, int *minLen*, int *maxLen*, string *ident* = " ")

Prompt for Amount Input from Encrypted Message

Prompts for amount input using the secure message data

```
@param message    Message
@param signature   Signature, 256 bytes
@param maskInput   Mask Input Entry
@param minLen     Minimum input length.  Cannot be less than 1
@param maxLen     Maximum input length.  Cannot be greater than 15
```

Timeout is hardwired to 3 minutes

Parameters

<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.
--------------	--

Returns

RETURN_CODE: Values can be parsed with `errorCode.getErrorString()`

21.8.1.42 `RETURN_CODE IDTechSDK.IDT_L80.pin_promptForKeyInput (string message, byte[] signature, bool maskInput, int minLen, int maxLen, string ident = " ")`

Prompt for Key Input from Encrypted Message

Prompts for a numeric key using the secure encrypted message data

Parameters

<i>message</i>	Message
<i>signature</i>	Signature, 256 bytes
<i>maskInput</i>	TRUE = entry is masked with '*', FALSE = entry is displayed on keypad
<i>minLen</i>	Minimum input length. Cannot be less than 1
<i>maxLen</i>	Maximum input length. Cannot be greater than 16

Integrators that desire to have custom messages for their application must submit the message request to IDTech with the following information (all fields required)

- Demo or Production Message Needed:
- Minimum PIN Requirement:
- Maximum PIN Requirement:
- Masking On or Off:
- Message Text:

The following data can be used as test data

- message: "Enter Ticket"
- minLen: 1
- maxLen: 16
- maskInput: False
- signature (demo unit): 80774ED82C8037776108A5E9E6B8F00070E7ADE1A3E6143B59B94EBC↵
B15712DB99ECA2A2193903F8FAEC83A516B2436F0C83EA143BB133620E6F94C7D47272C37657↵
A46D9B49783F5D3DDA46CAECAE168A28D9540D526F8A1D8EC50F1D051FFFC6A22622C99E46↵
A4A6178DA030D25F613119F1C8800A08933696E411742528415B5889E9ABD24BDF2E807A0518↵
C78E87B085218A3A18E291862CDDFA4B066EB1D4C0747D4800FD35D0DA47C250F7DB44146699↵
B853354081619C8B5FB3B66C293186588DA69CFD654202734408106FAD639EAF2932D8D8611EB↵
A2262728B4047AF1617370D29CDE9DD81B2B7934699F765E50017029A3739813686B194D12E2
- signature (prod unit): 8CE8E76C978871CCE1001010159F88E4968D8628213E28D9899C4103642608↵
E98AFBEC9BC29746B246E02A2C27566478E1F1C5FB9D0DB328D16A9A2F87216B93AA590D1261255↵
AAC9197007F2F4D18AF2E1DBECC5603339FCD31C12771138687E3608F7DCF835CE2DE8709E317↵
C7E12CD139792E2343683B9FB61819AB698D58A831D90596B3BD8DC1C0F5868E88C328D87C4489↵
E1AD378515B474C5DAEA876762F8D707610190175260BF6292AE910A7CB85C7386AEE30155038↵
F506BDC40D255F4ED269F8A2BA0D764BAC1146C35F7C18347204B74354FDDBC0A83DC2BF2BC34↵
E1A8161C209345122DC56F7963ACDD1183807CAA49D72B1F3CA79AB2CD12C2

Timeout is hardwired to 3 minutes

Parameters

<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.
--------------	--

Returns

RETURN_CODE: Values can be parsed with `errorCode.getErrorString()`

21.8.1.43 RETURN_CODE IDTechSDK.IDT_L80.pin_sendBeep (int *frequency*, int *duration*, string *ident* = " ")

Send Beep

Executes a beep request.

Parameters

<i>frequency</i>	Frequency, range 200-20000Hz
<i>duration</i>	Duration, range 16-65535ms
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with `device_getResponseCodeString`

21.8.1.44 RETURN_CODE IDTechSDK.IDT_L80.pin_setKeypressCapture (bool *showKeyValue*, string *ident* = " ")

Set Keypress Capture Mode

If TRUE, each keypress will be broadcast with its value 0-9, B, C, E. Function completes after timeout, C, or
If FALSE, each keypress will generate a generic keypress notification, with final results being returned after

Parameters

<i>showKeyValue</i>	TRUE = broadcast each keypress value, FALSE = only broadcast a key was pressed, not it's value.
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with `errorCode.getErrorString()`

21.8.1.45 static String IDTechSDK.IDT_L80.SDK_Version () [static]

SDK Version

- All Devices

Returns the current version of SDK

Returns

Framework version

21.8.1.46 `static void IDTechSDK.IDT_L80.setCallback (Callback my_Callback) [static]`

Set Callback

Sets the class callback

21.8.1.47 `static void IDTechSDK.IDT_L80.setCallback (IntPtr my_Callback, SynchronizationContext context) [static]`

Set Callback

Sets the class callback

Parameters

<i>my_Callback</i>	The callback function to receive the response message from device. defined as follows. public unsafe delegate void MFCCallBack(Parameters parameters, ident);
<i>context</i>	The context of the UI thread

21.8.1.48 `static void IDTechSDK.IDT_L80.setCommandTimeout (int milliseconds, string ident = " ") [static]`

Set Command Timeout

Sets the default timeout (in milliseconds) waiting for a blocking command response

Parameters

<i>milliseconds</i>	Time
---------------------	------

21.8.1.49 `static bool IDTechSDK.IDT_L80.useSerialPort (int port) [static]`

Use Serial Port Interface

Instructs SDK to attempt to use the Serial Port for communication with L80 using default baud rate

Parameters

<i>port</i>	Serial Port to use. Example COM1 = 1.
-------------	---------------------------------------

Returns

bool TRUE=successful, FALSE=failure

21.8.1.50 `static bool IDTechSDK.IDT_L80.useSerialPort (int port, int baud, string ident = " ") [static]`

Use Serial Port Interface with baud rate L80

Parameters

<i>port</i>	Serial Port to use. Example COM1 = 1.
<i>baud</i>	Baud rate to override default. Example 115200;

Returns

bool TRUE=successful, FALSE=failure

21.8.1.51 `static bool IDTechSDK.IDT_L80.useSerialPortLinux (string path) [static]`

Use Serial Port Interface on Linux

Instructs SDK to attempt to use the Serial Port for communication with BTMag using default baud rate on Linux implementations

Parameters

<i>path</i>	Path to use. Example /dev/ttyUSB*
-------------	-----------------------------------

Returns

bool TRUE=successful, FALSE=failure

21.8.1.52 `static bool IDTechSDK.IDT_L80.useSerialPortLinux (string path, int baud) [static]`

Use Serial Port Interface on Linux with baud rate

Instructs SDK to attempt to use the Serial Port for communication with BTPay on Linux implementations

Parameters

<i>path</i>	Path to use. Example /dev/ttyUSB*
<i>baud</i>	Baud rate to override default. Example 115200;

Returns

bool TRUE=successful, FALSE=failure

21.8.2 Property Documentation

21.8.2.1 `IDT_L80 IDTechSDK.IDT_L80.SharedController [static],[get]`

Singleton Instance

Establishes an singleton instance of [IDT_L80](#) class.

Returns

Instance of [IDT_L80](#)

The documentation for this class was generated from the following file:

- /Users/randy/Repo/universal-sdk/IDTechSDK_STD/DeviceHeaders/IDT_L80.cs

21.9 IDTechSDK.IDT_MiniSmartII Class Reference**Public Member Functions**

- RETURN_CODE [device_StartRKI](#) (int type, string ident="")

- RETURN_CODE [device_getDateTime](#) (ref byte[] dateTime, string ident="")
- RETURN_CODE [device_enableSecureHeadForMSII](#) (string ident="")
- RETURN_CODE [device_getFirmwareVersion](#) (ref string response, string ident="")
- RETURN_CODE [config_getModelNumber](#) (ref string response, string ident="")
- RETURN_CODE [icc_setClearPANID](#) (byte prePAN, byte postPAN, string ident="")
- RETURN_CODE [icc_getClearPANID](#) (ref byte prePAN, ref byte postPAN, string ident="")
- RETURN_CODE [config_getSerialNumber](#) (ref string response, string ident="")
- bool [config_setCmdTimeOutDuration](#) (int newTimeOut, string ident="")
- RETURN_CODE [device_rebootDevice](#) (string ident="")
- RETURN_CODE [device_sendDataCommand](#) (string cmd, bool calcLRC, ref byte[] response, string ident="")
- RETURN_CODE [device_sendDataCommand_ext](#) (string cmd, bool calcLRC, ref byte[] response, int timeout, bool noResponse, string ident="")
- RETURN_CODE [icc_exchangeAPDU](#) (string c_APDU, ref byte[] response, string ident="")
- RETURN_CODE [icc_getAPDU_KSN](#) (ref byte[] ksn, string ident="")
- RETURN_CODE [icc_getICCReaderStatus](#) (ref byte status, string ident="")
- RETURN_CODE [icc_getKeyFormatForICCDUKPT](#) (ref byte format, string ident="")
- RETURN_CODE [device_getKeyTypeForICCDUKPT](#) (ref byte type, string ident="")
- RETURN_CODE [icc_getKeyTypeForICCDUKPT](#) (ref byte type, string ident="")
- RETURN_CODE [icc_getReadingCharacteristics](#) (ref string mode, string ident="")
- RETURN_CODE [icc_powerOffICC](#) (string ident="")
- RETURN_CODE [icc_powerOnICC](#) (ref byte[] ATR, string ident="")
- RETURN_CODE [icc_setKeyFormatForICCDUKPT](#) (byte encryption, string ident="")
- RETURN_CODE [icc_setKeyTypeForICCDUKPT](#) (byte encryption, string ident="")
- RETURN_CODE [icc_setReadingCharacteristics](#) (int mode, string ident="")
- RETURN_CODE [emv_cancelTransaction](#) (string ident="")
- RETURN_CODE [emv_activateTransaction](#) (int timeout, byte[] tags, bool forceOnline, bool isFastEMV=false, string ident="")
- RETURN_CODE [emv_startTransaction](#) (double amount, double amtOther, int exponent, int type, int timeout, byte[] tags, bool forceOnline, bool isFastEMV=false, string ident="")
- RETURN_CODE [device_activateTransaction](#) (int timeout, byte[] tags, bool forceOnline, bool isFastEMV=false, string ident="")
- RETURN_CODE [device_startTransaction](#) (double amount, double amtOther, int exponent, int type, int timeout, byte[] tags, bool forceOnline, bool isFastEMV=false, string ident="")
- RETURN_CODE [emv_authenticateTransaction](#) (byte[] updatedTLV, string ident="")
- RETURN_CODE [emv_completeTransaction](#) (bool commError, byte[] authCode, byte[] iad, byte[] tlvScripts, byte[] tlv, string ident="")
- RETURN_CODE [emv_callbackResponseLCD](#) (EMV_LCD_DISPLAY_MODE type, byte selection, string ident="")
- RETURN_CODE [emv_callbackResponsePIN](#) (EMV_PIN_MODE type, byte[] KSN, byte[] PIN, string ident="")
- RETURN_CODE [emv_callbackResponseMSR](#) (byte[] MSR, string ident="")
- RETURN_CODE [device_setDateTime](#) (string ident="")
- RETURN_CODE [device_startRKI](#) (string ident="")
- RETURN_CODE [device_updateDeviceFirmware](#) (byte[] firmwareData, string ident="")
- RETURN_CODE [emv_getEMVKernelVersion](#) (ref string response, string ident="")
- RETURN_CODE [emv_getEMVKernelCheckValue](#) (ref string response, string ident="")
- RETURN_CODE [device_sendPAE](#) (string command, ref string response, int timeout, string ident="")
- RETURN_CODE [emv_getEMVConfigurationCheckValue](#) (ref string response, string ident="")
- RETURN_CODE [emv_retrieveTransactionResult](#) (byte[] tags, ref IDTTransactionData tlv, string ident="")
- RETURN_CODE [emv_removeApplicationData](#) (byte[] AID, string ident="")
- RETURN_CODE [emv_removeAllApplicationData](#) (string ident="")
- RETURN_CODE [emv_removeCAPK](#) (byte[] capk, string ident="")
- RETURN_CODE [emv_removeAllCAPK](#) (string ident="")
- RETURN_CODE [emv_removeCRL](#) (byte[] crlList, string ident="")
- RETURN_CODE [emv_removeAllCRL](#) (string ident="")
- RETURN_CODE [emv_removeTerminalData](#) (string ident="")

- RETURN_CODE [emv_retrieveAIDList](#) (ref byte[][] response, string ident="")
- RETURN_CODE [emv_retrieveApplicationData](#) (byte[] AID, ref byte[] tlv, string ident="")
- RETURN_CODE [emv_retrieveCAPK](#) (byte[] capk, ref byte[] key, string ident="")
- RETURN_CODE [emv_retrieveCAPKList](#) (ref byte[] keys, string ident="")
- RETURN_CODE [emv_retrieveCRLList](#) (ref byte[] list, string ident="")
- RETURN_CODE [emv_retrieveTerminalData](#) (ref byte[] tlv, string ident="")
- RETURN_CODE [emv_setApplicationData](#) (byte[] name, byte[] tlv, string ident="")
- RETURN_CODE [emv_setCAPK](#) (byte[] key, string ident="")
- RETURN_CODE [emv_setCRL](#) (byte[] list, string ident="")
- RETURN_CODE [emv_trySetTerminalData](#) (byte[] tlv, ref byte[] rejectedTLV, ref byte[] convertedTLV, bool overwrite=false, string ident="")
- RETURN_CODE [emv_setTerminalData](#) (byte[] tlv, string ident="")
- RETURN_CODE [emv_addTerminalData](#) (byte[] tlv, string ident="")
- RETURN_CODE [emv_setTerminalMajorConfiguration](#) (int configuration, string ident="")
- string [device_getResponseCodeString](#) (RETURN_CODE eCode)
- bool [createFastEMVData](#) (ref IDTTransactionData cData, string ident="")
- RETURN_CODE [device_getRKIStatus](#) (bool isProd, bool isMultiKey, ref string status, ref Dictionary< string, RKI_KEY_TYPE > keys, string ident="")
- RETURN_CODE [device_getAnyRKIStatus](#) (bool isMultiKey, ref string status, ref Dictionary< string, RKI_KEY_TYPE > keys, string ident="")
- RETURN_CODE [device_updateFirmwareFromZip](#) (byte[] zipfile, string ident="", bool isForeground=false)
- RETURN_CODE [device_sendConfigurationFromZip](#) (byte[] zip, string filename, VC_OPERATION_TYPE type, bool matchFW, string ident="", bool isForeground=false)
- RETURN_CODE [device_sendConfiguration](#) (string filename, VC_OPERATION_TYPE type, bool matchFW, string ident="", bool isForeground=false)
- RETURN_CODE [device_readConfigurationToMemory](#) (string memo, Configs.InstallRules rules, List< Configs.DeviceCommand > cmds, string ident="", bool isForeground=false)
- RETURN_CODE [device_getConfigurationFromMemory](#) (ref string json, string ident="")
- RETURN_CODE [device_RemoteKeyInjection](#) (RKI_KEY_TYPE type, string keyName, string ident="", bool performOnForeground=false)

Static Public Member Functions

- static bool [useSerialPort](#) (int port)
- static int [getCommandTimeout](#) (string ident="")
- static void [setCommandTimeout](#) (int milliseconds, string ident="")
- static bool [useSerialPort](#) (int port, int baud)
- static bool [useSerialPortLinux](#) (string path)
- static bool [useSerialPortLinux](#) (string path, int baud)
- static void [initSC](#) ()
- static void [setCallback](#) (Callback my_Callback)
- static void [setCallback](#) (IntPtr my_Callback, SynchronizationContext context)
- static String [SDK_Version](#) ()
- static void [emv_autoAuthenticate](#) (bool authenticate, string ident="")
- static void [emv_autoAuthenticateTags](#) (bool authenticate, byte[] tags, string ident="")
- static void [emv_allowFallback](#) (bool allow, string ident="")
- static void [lcd_retrieveMessage](#) (DisplayMessages.DISPLAY_MESSAGE_LANGUAGE lang, DisplayMessages.DISPLAY_MESSAGE_IDENTIFIER id, ref string line1, ref string line2)

Properties

- static [IDT_MiniSmartII SharedController](#) [get]

21.9.1 Detailed Description

Class for [IDT_MiniSmartII](#) MSR/ICC reader

21.9.2 Member Function Documentation

21.9.2.1 RETURN_CODE IDTechSDK.IDT_MiniSmartII.config_getModelNumber (ref string *response*, string *ident* = " ")

Polls device for Model Number

Parameters

<i>response</i>	Returns Model Number
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with `errorCode.getErrorString(string ident = "")`

21.9.2.2 RETURN_CODE IDTechSDK.IDT_MiniSmartII.config_getSerialNumber (ref string *response*, string *ident* = " ")

Polls device for Serial Number

Parameters

<i>response</i>	Returns Serial Number
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with `errorCode.getErrorString(string ident = "")`

21.9.2.3 bool IDTechSDK.IDT_MiniSmartII.config_setCmdTimeOutDuration (int *newTimeOut*, string *ident* = " ")

Command Acknowledgement Timeout

Sets the amount of seconds to wait for an {ACK} to a command before a timeout. Responses should normally be received under one second. Default is 3 seconds

Parameters

<i>newTimeOut</i>	Timeout value. Valid range 1 - 60 seconds
-------------------	---

Returns

Success flag. Determines if value was set and in range.

21.9.2.4 bool IDTechSDK.IDT_MiniSmartII.createFastEMVData (ref IDTTTransactionData *cData*, string *ident* = " ")

Create Fast EMV Data

At the completion of a Fast EMV Transaction, after the final card decision is returned and the IDTTransactionData object is provided, sending that cData object to this method will populate the .fastEMV element with string data that represents the Fast EMV data that would be returned from and IDTech FastEMV over KB protocol

Parameters

<i>cData</i>	The IDTTransactionData object populated with card data.
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with `errorCode.getErrorString(string ident = "")`

21.9.2.5 RETURN_CODE IDTechSDK.IDT_MiniSmartII.device_activateTransaction (int timeout, byte[] tags, bool forceOnline, bool isFastEMV = false, string ident = "")

Start EMV Transaction Request

Authorizes the EMV transaction for an ICC card

The tags will be returned in the callback routine.

Parameters

<i>amount</i>	Transaction amount value (tag value 9F02, string ident = "")
<i>amtOther</i>	Other amount value, if any (tag value 9F03, string ident = "")
<i>exponent</i>	Number of characters after decimile point
<i>type</i>	Transaction type (tag value 9C).
<i>timeout</i>	Timeout value in seconds.
<i>tags</i>	Any other tags to be included in the request. Passed as TLV Data. Example, tag 9F02 with amount 0x000000000100 would be 0x9F0206000000000100 If tags 9F02 (amount), 9F03 (other amount), or 9C (transaction type) are included, they will take priority over these values supplied as individual parameters to this method.
<i>forceOnline</i>	TRUE = do not allow offline approval, FALSE = allow ICC to approve offline if terminal capable Note: To request tags to be included in default response, use tag DFEE1A, and specify tag list. Example four tags 9F02, 9F36, 95, 9F37 to be included in response = DFEE1A079F029F369F9F37
<i>isFastEMV</i>	If TRUE, it will populate the IDTTransactionData.fastEMV with ASCII data similar to IDTech FastEMV KB output, after performing an auto-authenticate and auto-complete with ResultCode = Could Not Contact Host
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with `device_getResponseCodeString`

21.9.2.6 RETURN_CODE IDTechSDK.IDT_MiniSmartII.device_enableSecureHeadForMSII (string ident = "")

Enable SecureHead as MSR Device for MiniSmartII EMV transactions

This function will allow the SecureHead to be used as the MSR device when MiniSmartII device_startTransaction is executed

Parameters

<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.
--------------	--

Returns

RETURN_CODE: Values can be parsed with `errorCode.getErrorString(string ident = "")`

21.9.2.7 RETURN_CODE IDTechSDK.IDT_MiniSmartII.device_getAnyRKIStatus (bool *isMultiKey*, ref string *status*, ref Dictionary< string, RKI_KEY_TYPE > *keys*, string *ident* = " ")

Get RKI Status Polls the RKI servers to validate device RKI eligibility. Note: if device type is known in advance (production or demo device), it is more efficient to use `device_getRKIStatus` instead

Parameters

<i>isMultiKey</i>	True = MultiKey, False = SingleKey
<i>status</i>	RKI status returned as a string
<i>keys</i>	Returns all available keys
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with `errorCode.getErrorString()`. When no data is available, return code = RETURN_CODE_NO_DATA_AVAILABLE

21.9.2.8 RETURN_CODE IDTechSDK.IDT_MiniSmartII.device_getConfigurationFromMemory (ref string *json*, string *ident* = " ")

Read Device Configuration From Memory Retrieves the ViVOconfig configuration data from memory after a successful `device_readConfigurationToMemory`

Parameters

<i>json</i>	Returns the .json file data that can be saved to disk
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with `errorCode.getErrorString()`. When no data is available, return code = RETURN_CODE_NO_DATA_AVAILABLE

21.9.2.9 RETURN_CODE IDTechSDK.IDT_MiniSmartII.device_getDateTime (ref byte[] *dateTime*, string *ident* = " ")

Get Date Time

Gets current system date and time of the device.

Parameters

<i>dateTime</i>	The date time returned as follows: <ul style="list-style-type: none"> • byte 0: Year 00-99 • byte 1: Month 01-12 • byte 2: Date 01-31 • byte 3: Hour 00-23 • byte 4: Minute 00-59 • byte 5: Second 00-59
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString

21.9.2.10 RETURN_CODE IDTechSDK.IDT_MiniSmartII.device_getFirmwareVersion (ref string *response*, string *ident* = " ")

Polls device for Firmware Version

Parameters

<i>response</i>	Response returned of Firmware Version
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with errorCode.getErrorString(string ident = "")

21.9.2.11 RETURN_CODE IDTechSDK.IDT_MiniSmartII.device_getKeyTypeForICCDUKPT (ref byte *type*, string *ident* = " ")

Get Key Type for ICC DUKPT

Specifies the key type used for ICC DUKPT encryption

Parameters

<i>type</i>	Response returned from method: <ul style="list-style-type: none"> • 'DATA': Encrypted card data with Data Key DUKPT Key had been loaded.(default, string ident = "") • 'PIN': Encrypted card data with PIN Key if DUKPT Key had been loaded.
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with errorCode.getErrorString(string ident = "")

21.9.2.12 string IDTechSDK.IDT_MiniSmartIIL.device_getResponseCodeString (RETURN_CODE *eCode*)

Get the description of response result.

Parameters

<i>eCode</i>	the response result.
--------------	----------------------

Return values

<i>the</i>	string for description of response result
------------	---

- case 0: "no error, beginning task";
- case 1: "no response from reader";
- case 2: "invalid response data";
- case 3: "time out for task or CMD";
- case 4: "wrong parameter";
- case 5: "SDK is doing MSR or ICC task";
- case 6: "SDK is doing PINPad task";
- case 7: "SDK is doing CTLS task";
- case 8: "SDK is doing EMV task";
- case 9: "SDK is doing Other task";
- case 10: "err response or data";
- case 11: "no reader attached";
- case 12: "mono audio is enabled";
- case 13: "did connection";
- case 14: "audio volume is too low";
- case 15: "task or CMD be canceled";
- case 16: "UF wrong string format";
- case 17: "UF file not found";
- case 18: "UF wrong file format";
- case 19: "Attempt to contact online host failed";
- case 20: "Attempt to perform RKI failed";
- case 0x300: "Key Type(TDES) of Session Key is not same as the related Master Key.";
- case 0x400: "Related Key was not loaded.";
- case 0x500: "Key Same.";
- case 0x501: "Key is all zero";
- case 0x502: "TR-31 format error";
- case 0x702: "PAN is Error Key.";

- case 0x705: "No Internal MSR PAN (or Internal MSR PAN is erased timeout)";
- case 0X0C01: "Incorrect Frame Tag";
- case 0X0C02: "Incorrect Frame Type";
- case 0X0C03: "Unknown Frame Type";
- case 0X0C04: "Unknown Command";
- case 0X0C05: "Unknown Sub-Command";
- case 0X0C06: "CRC Error";
- case 0X0C07: "Failed";
- case 0X0C08: "Timeout";
- case 0X0C0A: "Incorrect Parameter";
- case 0X0C0B: "Command Not Supported";
- case 0X0C0C: "Sub-Command Not Supported";
- case 0X0C0D: "Parameter Not Supported / Status Abort Command";
- case 0X0C0F: "Sub-Command Not Allowed";
- case 0X0D01: "Incorrect Header Tag";
- case 0X0D02: "Unknown Command";
- case 0X0D03: "Unknown Sub-Command";
- case 0X0D04: "CRC Error in Frame";
- case 0X0D05: "Incorrect Parameter";
- case 0X0D06: "Parameter Not Supported";
- case 0X0D07: "Mal-formatted Data";
- case 0X0D08: "Timeout";
- case 0X0D0A: "Failed / NACK";
- case 0X0D0B: "Command not Allowed";
- case 0X0D0C: "Sub-Command not Allowed";
- case 0X0D0D: "Buffer Overflow (Data Length too large for reader buffer)";
- case 0X0D0E: "User Interface Event";
- case 0X0D11: "Communication type not supported, VT-1, burst, etc.";
- case 0X0D12: "Secure interface is not functional or is in an intermediate state.";
- case 0X0D13: "Data field is not mod 8";
- case 0X0D14: "Pad 0x80 not found where expected";
- case 0X0D15: "Specified key type is invalid";
- case 0X0D1: "Could not retrieve key from the SAM(InitSecureComm)";
- case 0X0D17: "Hash code problem";
- case 0X0D18: "Could not store the key into the SAM(InstallKey)";
- case 0X0D19: "Frame is too large";

- case 0X0D1A: "Unit powered up in authentication state but POS must resend the InitSecureComm command";
- case 0X0D1B: "The EEPROM may not be initialized because SecCommInterface does not make sense";
- case 0X0D1C: "Problem encoding APDU";
- case 0X0D20: "Unsupported Index(ILM) SAM Transceiver error – problem communicating with the SAM(Key Mgr)";
- case 0X0D2: "Unexpected Sequence Counter in multiple frames for single bitmap(ILM) Length error in data returned from the SAM(Key Mgr)";
- case 0X0D22: "Improper bit map(ILM)";
- case 0X0D23: "Request Online Authorization";
- case 0X0D24: "ViVOCard3 raw data read successful";
- case 0X0D25: "Message index not available(ILM) ViVOcomm activate transaction card type(ViVOcomm)";
- case 0X0D26: "Version Information Mismatch(ILM)";
- case 0X0D27: "Not sending commands in correct index message index(ILM)";
- case 0X0D28: "Time out or next expected message not received(ILM)";
- case 0X0D29: "ILM languages not available for viewing(ILM)";
- case 0X0D2A: "Other language not supported(ILM)";
- case 0X0D41: "Unknown Error from SAM";
- case 0X0D42: "Invalid data detected by SAM";
- case 0X0D43: "Incomplete data detected by SAM";
- case 0X0D44: "Reserved";
- case 0X0D45: "Invalid key hash algorithm";
- case 0X0D46: "Invalid key encryption algorithm";
- case 0X0D47: "Invalid modulus length";
- case 0X0D48: "Invalid exponent";
- case 0X0D49: "Key already exists";
- case 0X0D4A: "No space for new RID";
- case 0X0D4B: "Key not found";
- case 0X0D4C: "Crypto not responding";
- case 0X0D4D: "Crypto communication error";
- case 0X0D4E: "Module-specific error for Key Manager";
- case 0X0D4F: "All key slots are full (maximum number of keys has been installed)";
- case 0X0D50: "Auto-Switch OK";
- case 0X0D51: "Auto-Switch failed";
- case 0X0D90: "Account DUKPT Key not exist";
- case 0X0D91: "Account DUKPT Key KSN exhausted";
- case 0x0D00: "This Key had been loaded.";

- case 0x0E00: "Base Time was loaded.";
- case 0x0F00: "Encryption Or Decryption Failed.";
- case 0x1000: "Battery Low Warning (It is High Priority Response while Battery is Low.)";
- case 0x1800: "Send "Cancel Command" after send "Get Encrypted PIN" & "Get Numeric "& "Get Amount"";
- case 0x1900: "Press "Cancel" key after send "Get Encrypted PIN" & "Get Numeric "& "Get Amount"";
- case 0x30FF: "Security Chip is not connect";
- case 0x3000: "Security Chip is deactivation & Device is In Removal Legally State.";
- case 0x3101: "Security Chip is activation & Device is In Removal Legally State.";
- case 0x5500: "No Admin DUKPT Key.";
- case 0x5501: "Admin DUKPT Key STOP.";
- case 0x5502: "Admin DUKPT Key KSN is Error.";
- case 0x5503: "Get Authentication Code1 Failed.";
- case 0x5504: "Validate Authentication Code Error.";
- case 0x5505: "Encrypt or Decrypt data failed.";
- case 0x5506: "Not Support the New Key Type.";
- case 0x5507: "New Key Index is Error.";
- case 0x5508: "Step Error.";
- case 0x5509: "KSN Error";
- case 0x550A: "MAC Error.";
- case 0x550B: "Key Usage Error.";
- case 0x550C: "Mode Of Use Error.";
- case 0x550F: "Other Error.";
- case 0x6000: "Save or Config Failed / Or Read Config Error.";
- case 0x6200: "No Serial Number.";
- case 0x6900: "Invalid Command - Protocol is right, but task ID is invalid.";
- case 0x6A01: "Unsupported Command – Protocol and task ID are right, but command is invalid – In this State";
- case 0x6A00: "Unsupported Command - Protocol and task ID are right, but command is invalid.";
- case 0x6B00: "Unknown parameter in command - Protocol task ID and command are right, but parameter is invalid.";
- case 0x6C00: "Unknown parameter in command – Protocol task ID and command are right, but length is out of the requirement.";
- case 0x7200: "Device is suspend (MKS suspend or press password suspend).";
- case 0x7300: "PIN DUKPT is STOP (21 bit 1).";
- case 0x7400: "Device is Busy.";
- case 0xE100: "Can not enter sleep mode";
- case 0xE200: "File has existed";

- case 0xE300: "File has not existed";
- case 0xE313: "IO line low -- Card error after session start";
- case 0xE400: "Open File Error";
- case 0xE500: "SmartCard Error";
- case 0xE600: "Get MSR Card data is error";
- case 0xE700: "Command time out";
- case 0xE800: "File read or write is error";
- case 0xE900: "Active 1850 error!";
- case 0xEA00: "Load bootloader error";
- case 0xEF00: "Protocol Error- STX or ETX or check error.";
- case 0xEB00: "Picture is not exist";
- case 0x2C02: "No Microprocessor ICC seated";
- case 0x2C06: "no card seated to request ATR";
- case 0x2D01: "Card Not Supported,";
- case 0x2D03: "Card Not Supported, wants CRC";
- case 0x690D: "Command not supported on reader without ICC support";
- case 0x8100: "ICC error time out on power-up";
- case 0x8200: "invalid TS character received - Wrong operation step";
- case 0x8300: "Decode MSR Error";
- case 0x8400: "TriMagII no Response";
- case 0x8500: "No Swipe MSR Card";
- case 0x8510: "No Financial Card";
- case 0x8600: "Unsupported F, D, or combination of F and D";
- case 0x8700: "protocol not supported EMV TD1 out of range";
- case 0x8800: "power not at proper level";
- case 0x8900: "ATR length too long";
- case 0x8B01: "EMV invalid TA1 byte value";
- case 0x8B02: "EMV TB1 required";
- case 0x8B03: "EMV Unsupported TB1 only 00 allowed";
- case 0x8B04: "EMV Card Error, invalid BWI or CWI";
- case 0x8B06: "EMV TB2 not allowed in ATR";
- case 0x8B07: "EMV TC2 out of range";
- case 0x8B08: "EMV TC2 out of range";
- case 0x8B09: "per EMV96 TA3 must be > 0xF";
- case 0x8B10: "ICC error on power-up";
- case 0x8B11: "EMV T=1 then TB3 required";

- case 0x8B12: "Card Error, invalid BWI or CWI";
- case 0x8B13: "Card Error, invalid BWI or CWI";
- case 0x8B17: "EMV TC1/TB3 conflict-";
- case 0x8B20: "EMV TD2 out of range must be T=1";
- case 0x8C00: "TCK error";
- case 0xA304: "connector has no voltage setting";
- case 0xA305: "ICC error on power-up invalid (SBLK(IFSD) exchange";
- case 0xE301: "ICC error after session start";
- case 0xFF00: "Request to go online";
- case 0xFF01: "EMV: Accept the offline transaction";
- case 0xFF02: "EMV: Decline the offline transaction";
- case 0xFF03: "EMV: Accept the online transaction";
- case 0xFF04: "EMV: Decline the online transaction";
- case 0xFF05: "EMV: Application may fallback to magstripe technology";
- case 0xFF06: "EMV: ICC detected tah the conditions of use are not satisfied";
- case 0xFF07: "EMV: ICC didn't accept transaction";
- case 0xFF08: "EMV: Transaction was cancelled";
- case 0xFF09: "EMV: Application was not selected by kernel or ICC format error or ICC missing data error";
- case 0xFF0A: "EMV: Transaction is terminated";
- case 0xFF0B: "EMV: Other EMV Error";
- case 0xFFFF: "NO RESPONSE";
- case 0xF002: "ICC communication timeout";
- case 0xF003: "ICC communication Error";
- case 0xF00F: "ICC Card Seated and Highest Priority, disable MSR work request";
- case 0xF200: "AID List / Application Data is not exist";
- case 0xF201: "Terminal Data is not exist";
- case 0xF202: "TLV format is error";
- case 0xF203: "AID List is full";
- case 0xF204: "Any CA Key is not exist";
- case 0xF205: "CA Key RID is not exist";
- case 0xF206: "CA Key Index it not exist";
- case 0xF207: "CA Key is full";
- case 0xF208: "CA Key Hash Value is Error";
- case 0xF209: "Transaction format error";
- case 0xF20A: "The command will not be processing";
- case 0xF20B: "CRL is not exist";

- case 0xF20C: "CRL number exceed max number";
- case 0xF20D: "Amount,Other Amount,Trasaction Type are missing";
- case 0xF20E: "The Identification of algorithm is mistake";
- case 0xF20F: "No Financial Card";
- case 0xF210: "In Encrypt Result state, TLV total Length is greater than Max Length";
- case 0x1001: "INVALID ARG";
- case 0x1002: "FILE_OPEN_FAILED";
- case 0x1003: "FILE OPERATION_FAILED";
- case 0x2001: "MEMORY_NOT_ENOUGH";
- case 0x3002: "SMARTCARD_FAIL";
- case 0x3003: "SMARTCARD_INIT_FAILED";
- case 0x3004: "FALLBACK_SITUATION";
- case 0x3005: "SMARTCARD_ABSENT";
- case 0x3006: "SMARTCARD_TIMEOUT";
- case 0x5001: "EMV_PARSING_TAGS_FAILED";
- case 0x5002: "EMV_DUPLICATE_CARD_DATA_ELEMENT";
- case 0x5003: "EMV_DATA_FORMAT_INCORRECT";
- case 0x5004: "EMV_NO_TERM_APP";
- case 0x5005: "EMV_NO_MATCHING_APP";
- case 0x5006: "EMV_MISSING_MANDATORY_OBJECT";
- case 0x5007: "EMV_APP_SELECTION_RETRY";
- case 0x5008: "EMV_GET_AMOUNT_ERROR";
- case 0x5009: "EMV_CARD_REJECTED";
- case 0x5010: "EMV_AIP_NOT_RECEIVED";
- case 0x5011: "EMV_AFL_NOT_RECEIVED";
- case 0x5012: "EMV_AFL_LEN_OUT_OF_RANGE";
- case 0x5013: "EMV_SFI_OUT_OF_RANGE";
- case 0x5014: "EMV_AFL_INCORRECT";
- case 0x5015: "EMV_EXP_DATE_INCORRECT";
- case 0x5016: "EMV_EFF_DATE_INCORRECT";
- case 0x5017: "EMV_ISS_COD_TBL_OUT_OF_RANGE";
- case 0x5018: "EMV_CRYPTOGAM_TYPE_INCORRECT";
- case 0x5019: "EMV_PSE_NOT_SUPPORTED_BY_CARD";
- case 0x5020: "EMV_USER_SELECTED_LANGUAGE";
- case 0x5021: "EMV_SERVICE_NOT_ALLOWED";
- case 0x5022: "EMV_NO_TAG_FOUND";

- case 0x5023: "EMV_CARD_BLOCKED";
- case 0x5024: "EMV_LEN_INCORRECT";
- case 0x5025: "CARD_COM_ERROR";
- case 0x5026: "EMV_TSC_NOT_INCREASED";
- case 0x5027: "EMV_HASH_INCORRECT";
- case 0x5028: "EMV_NO_ARC";
- case 0x5029: "EMV_INVALID_ARC";
- case 0x5030: "EMV_NO_ONLINE_COMM";
- case 0x5031: "TRAN_TYPE_INCORRECT";
- case 0x5032: "EMV_APP_NO_SUPPORT";
- case 0x5033: "EMV_APP_NOT_SELECT";
- case 0x5034: "EMV_LANG_NOT_SELECT";
- case 0x5035: "EMV_NO_TERM_DATA";
- case 0x6001: "CVM_TYPE_UNKNOWN";
- case 0x6002: "CVM_AIP_NOT_SUPPORTED";
- case 0x6003: "CVM_TAG_8E_MISSING";
- case 0x6004: "CVM_TAG_8E_FORMAT_ERROR";
- case 0x6005: "CVM_CODE_IS_NOT_SUPPORTED";
- case 0x6006: "CVM_COND_CODE_IS_NOT_SUPPORTED";
- case 0x6007: "NO_MORE_CVM";
- case 0x6008: "PIN_BYPASSED_BEFORE";
- case 0x7001: "PK_BUFFER_SIZE_TOO_BIG";
- case 0x7002: "PK_FILE_WRITE_ERROR";
- case 0x7003: "PK_HASH_ERROR";
- case 0x8001: "NO_CARD_HOLDER_CONFIRMATION";
- case 0x8002: "GET_ONLINE_PIN";
- case 0xD000: "Data not exist";
- case 0xD001: "Data access error";
- case 0xD100: "RID not exist";
- case 0xD101: "RID existed";
- case 0xD102: "Index not exist";
- case 0xD200: "Maximum exceeded";
- case 0xD201: "Hash error";
- case 0xD205: "System Busy";
- case 0x0E01: "Unable to go online";
- case 0x0E02: "Technical Issue";

- case 0x0E03: "Declined";
- case 0x0E04: "Issuer Referral transaction";
- case 0x0F01: "Decline the online transaction";
- case 0x0F02: "Request to go online";
- case 0x0F03: "Transaction is terminated";
- case 0x0F05: "Application was not selected by kernel or ICC format error or ICC missing data error";
- case 0x0F07: "ICC didn't accept transaction";
- case 0x0F0A: "Application may fallback to magstripe technology";
- case 0x0F0C: "Transaction was cancelled";
- case 0x0F0D: "Timeout";
- case 0x0F0F: "Other EMV Error";
- case 0x0F10: "Accept the offline transaction";
- case 0x0F11: "Decline the offline transaction";
- case 0x0F21: "ICC detected tah the conditions of use are not satisfied";
- case 0x0F22: "No app were found on card matching terminal configuration";
- case 0x0F23: "Terminal file does not exist";
- case 0x0F24: "CAPK file does not exist";
- case 0x0F25: "CRL Entry does not exist";
- case 0x0FFE: "code when blocking is disabled";
- case 0x0FFF: "code when command is not applicable on the selected device";
- case 0xF005: "ICC Encrypted C-APDU Data Structure Length Error Or Format Error.";
- case 0xBBE0: "CM100 Success";
- case 0xBBE1: "CM100 Parameter Error";
- case 0xBBE2: "CM100 Low Output Buffer";
- case 0xBBE3: "CM100 Card Not Found";
- case 0xBBE4: "CM100 Collision Card Exists";
- case 0xBBE5: "CM100 Too Many Cards Exist";
- case 0xBBE6: "CM100 Saved Data Does Not Exist";
- case 0xBBE8: "CM100 No Data Available";
- case 0xBBE9: "CM100 Invalid CID Returned";
- case 0xBBEA: "CM100 Invalid Card Exists";
- case 0xBBEC: "CM100 Command Unsupported";
- case 0xBBED: "CM100 Error In Command Process";
- case 0xBBEE: "CM100 Invalid Command";
- case 0X9031: "Unknown command";
- case 0X9032: "Wrong parameter (such as the length of the command is incorrect)";

- case 0X9038: "Wait (the command couldn't be finished in BWT)";
- case 0X9039: "Busy (a previously command has not been finished)";
- case 0X903A: "Number of retries over limit";
- case 0X9040: "Invalid Manufacturing system data";
- case 0X9041: "Not authenticated";
- case 0X9042: "Invalid Master DUKPT Key";
- case 0X9043: "Invalid MAC Key";
- case 0X9044: "Reserved for future use";
- case 0X9045: "Reserved for future use";
- case 0X9046: "Invalid DATA DUKPT Key";
- case 0X9047: "Invalid PIN Pairing DUKPT Key";
- case 0X9048: "Invalid DATA Pairing DUKPT Key";
- case 0X9049: "No nonce generated";
- case 0X9949: "No GUID available. Perform getVersion first.";
- case 0X9950: "MAC Calculation unsuccessful. Check BDK value.";
- case 0X904A: "Not ready";
- case 0X904B: "Not MAC data";
- case 0X9050: "Invalid Certificate";
- case 0X9051: "Duplicate key detected";
- case 0X9052: "AT checks failed";
- case 0X9053: "TR34 checks failed";
- case 0X9054: "TR31 checks failed";
- case 0X9055: "MAC checks failed";
- case 0X9056: "Firmware download failed";
- case 0X9060: "Log is full";
- case 0X9061: "Removal sensor unengaged";
- case 0X9062: "Any hardware problems";
- case 0X9070: "ICC communication timeout";
- case 0X9071: "ICC data error (such check sum error)";
- case 0X9072: "Smart Card not powered up";

21.9.2.13 `RETURN_CODE IDTechSDK.IDT_MiniSmartII.device_getRKIStatus (bool isProd, bool isMultiKey, ref string status, ref Dictionary< string, RKI_KEY_TYPE > keys, string ident = " ")`

Get RKI Status From Specified Server Polls the RKI server to validate device RKI eligibility

Parameters

<i>isProd</i>	TRUE = poll production RKI server, FALSE = poll demo RKI server
<i>isMultiKey</i>	True = MultiKey, False = SingleKey
<i>status</i>	RKI status returned as a string
<i>keys</i>	Returns dictionary of available keys
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with `errorCode.getErrorString()`. When no data is available, return code = RETURN_CODE_NO_DATA_AVAILABLE

21.9.2.14 RETURN_CODE IDTechSDK.IDT_MiniSmartII.device_readConfigurationToMemory (string *memo*, Configs.InstallRules *rules*, List< Configs.DeviceCommand > *cmds*, string *ident* = " ", bool *isForeground* = false)

Read Device Configuration To Memory Executes a ViVOconfig read of device settings and stores the results in memory

Once a ViVOconfig read starts, all messages will be returned to the callback as DeviceState.ViVOconfig. The ViVOconfig operation is complete when the return code from a DeviceState.ViVOconfig message is one of the following values:

- RETURN_CODE_SDK_VIVOCONFIG_READ_CONFIG_SUCCESS = Read process completed successfully. Execute `device_getConfigurationFromMemory` to retrieve the configuration data
- RETURN_CODE_SDK_VIVOCONFIG_READ_CONFIG_FAILED = Read process FAILED.

Parameters

<i>memo</i>	Optional: populates the memo field of the captured .json file
<i>rules</i>	Optional: populates the .json file with available install rules
<i>cmds</i>	Optional: populates the .json file with commands to execute
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.
<i>isForeground</i>	If TRUE, will block program until update complete. Otherwise, FALSE performs update on background.

Returns

RETURN_CODE: Values can be parsed with `errorCode.getErrorString()`. When no data is available, return code = RETURN_CODE_NO_DATA_AVAILABLE

21.9.2.15 RETURN_CODE IDTechSDK.IDT_MiniSmartII.device_rebootDevice (string *ident* = " ")

Reboot Device

Executes a command to restart the device.

Parameters

<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.
--------------	--

Returns

RETURN_CODE: Values can be parsed with `errorCode.getErrorString(string ident = "")`

21.9.2.16 `RETURN_CODE IDTechSDK.IDT_MiniSmartII.device_RemoteKeyInjection (RKI_KEY_TYPE type, string keyName, string ident = " ", bool performOnForeground = false)`

Remote Key Injection Performs a remote key injection for the device

Parameters

<i>type</i>	Remote Key Injection Type
<i>keyName</i>	Name of key (optional)
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.
<i>performOnForeground</i>	TRUE = block during process (default is false)

Returns

RETURN_CODE: Values can be parsed with `errorCode.getErrorString()`. When no data is available, return code = RETURN_CODE_NO_DATA_AVAILABLE

21.9.2.17 `RETURN_CODE IDTechSDK.IDT_MiniSmartII.device_sendConfiguration (string filename, VC_OPERATION_TYPE type, bool matchFW, string ident = " ", bool isForeground = false)`

Send Configuration Executes a ViVOconfig update or verify to a device

Once a ViVOconfig write or verify starts, all messages will be returned to the callback as `DeviceState.ViVOconfig`. The ViVOconfig operation is complete when the return code from a `DeviceState.ViVOconfig` message is one of the following values:

- RETURN_CODE_SDK_VIVOCONFIG_VERIFY_SUCCESS = Verification process completed successfully. No differences found.
- RETURN_CODE_SDK_VIVOCONFIG_VERIFY_SUCCESS_WITH_WARNING = Verification process completed with warnings.
- RETURN_CODE_SDK_VIVOCONFIG_VERIFY_FAILED = Verification process FAILED
- RETURN_CODE_SDK_VIVOCONFIG_WRITE_SUCCESS = Write process completed successfully.
- RETURN_CODE_SDK_VIVOCONFIG_WRITE_SUCCESS_WITH_WARNING = Write process completed with warnings
- RETURN_CODE_SDK_VIVOCONFIG_WRITE_FAILED = Write process FAILED

Parameters

<i>filename</i>	The .json configuration file to update the device with NOTE: You can also use this parameter to pass the path of where the .json file is located at on local storage instead
<i>filename</i>	The name of the .json configuration file
<i>type</i>	VC_OPERATION_TYPE

- VC_OPERATION_TYPE_WRITE = Write File To Device, Hash must validate
- VC_OPERATION_TYPE_VERIFY = Verify Device With File

- VC_OPERATION_TYPE_WRITE_IGNORE_HASH = Write File To Device, Ignore Hash
- VC_OPERATION_TYPE_WRITE_FIX_HASH = Write File To Device, Fix Hash If Necessary

Parameters

<i>matchFW</i>	TRUE = Device FW must match file FW, FALSE = Don't validate device FW against file
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.
<i>isForeground</i>	If TRUE, will block program until update complete. Otherwise, FALSE performs update on background.

Returns

RETURN_CODE: Values can be parsed with `errorCode.getErrorString()`. When no data is available, return code = RETURN_CODE_NO_DATA_AVAILABLE

21.9.2.18 RETURN_CODE IDTechSDK.IDT_MiniSmartII.device_sendConfigurationFromZip (byte[] zip, string filename, VC_OPERATION_TYPE type, bool matchFW, string ident = " ", bool isForeground = false)

Send Configuration From Zip Executes a ViVOconfig update or verify to a device using files from a .zip archive

Once a ViVOconfig write or update starts, all messages will be returned to the callback as DeviceState.ViVOconfig. The ViVOconfig operation is complete when the return code from a DeviceState.ViVOconfig message is one of the following values:

- RETURN_CODE_SDK_VIVOCONFIG_VERIFY_SUCCESS = Verification process completed successfully. No differences found.
- RETURN_CODE_SDK_VIVOCONFIG_VERIFY_SUCCESS_WITH_WARNING = Verification process completed with warnings.
- RETURN_CODE_SDK_VIVOCONFIG_VERIFY_FAILED = Verification process FAILED
- RETURN_CODE_SDK_VIVOCONFIG_WRITE_SUCCESS = Write process completed successfully.
- RETURN_CODE_SDK_VIVOCONFIG_WRITE_SUCCESS_WITH_WARNING = Write process completed with warnings
- RETURN_CODE_SDK_VIVOCONFIG_WRITE_FAILED = Write process FAILED

Parameters

<i>zip</i>	.Zip archive containing the .json configuration file and also any associated image assets NOTE: You can also use this parameter to pass the path of where the .zip file is located at on local storage instead
<i>filename</i>	The name of the .json configuration file
<i>type</i>	VC_OPERATION_TYPE

- VC_OPERATION_TYPE_WRITE = Write File To Device, Hash must validate
- VC_OPERATION_TYPE_VERIFY = Verify Device With File
- VC_OPERATION_TYPE_WRITE_IGNORE_HASH = Write File To Device, Ignore Hash
- VC_OPERATION_TYPE_WRITE_FIX_HASH = Write File To Device, Fix Hash If Necessary

Parameters

<i>matchFW</i>	TRUE = Device FW must match file FW, FALSE = Don't validate device FW against file
----------------	--

Parameters

<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.
<i>isForeground</i>	If TRUE, will block program until update complete. Otherwise, FALSE performs update on background.

Returns

RETURN_CODE: Values can be parsed with `errorCode.getErrorString()`. When no data is available, return code = RETURN_CODE_NO_DATA_AVAILABLE

21.9.2.19 RETURN_CODE IDTechSDK.IDT_MiniSmartII.device_sendDataCommand (string *cmd*, bool *calcLRC*, ref byte[] *response*, string *ident* = " ")

Send a data command to the device

Sends a command to the device.

Parameters

<i>cmd</i>	String representation of command to execute
<i>calcLRC</i>	If TRUE, this will wrap command with start/length/lrc/sum/end: '{STX}{Len_Low}{Len_High} data {CheckLRC} {CheckSUM} {ETX}'
<i>response</i>	Response data
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with `device_getResponseCodeString`

21.9.2.20 RETURN_CODE IDTechSDK.IDT_MiniSmartII.device_sendDataCommand_ext (string *cmd*, bool *calcLRC*, ref byte[] *response*, int *timeout*, bool *noResponse*, string *ident* = " ")

Send a data command to the device - extended

Sends a command to the device.

Parameters

<i>cmd</i>	String representation of command to execute
<i>calcLRC</i>	If TRUE, this will wrap command as NGA with start/length/lrc/sum/end: '{STX}{Len_Low}{Len_High} data {CheckLRC} {CheckSUM} {ETX}'
<i>response</i>	Response data
<i>timeout</i>	Timeout value waiting for response, in milliseconds (1000 = 1 second, string <i>ident</i> = "")
<i>noResponse</i>	if TRUE, this will not wait for a response and immediately return SUCCESS
<i>calcITP</i>	If TRUE, this will wrap command as ITP with start/end/lrc: '{STX} data {ETX}{CheckLRC}'
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with `device_getResponseCodeString`

21.9.2.21 **RETURN_CODE** IDTechSDK.IDT_MiniSmartII.device_sendPAE (string *command*, ref string *response*, int *timeout*, string *ident* = " ")

Send Payment Application Engine Command

Executes a PAE command

Parameters

<i>command</i>	ASCII command string, should start with "*PAE"
<i>response</i>	command response
<i>timeout</i>	timeout waiting for PAE response
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with `errorCode.getErrorString(string ident = "")`

21.9.2.22 **RETURN_CODE** IDTechSDK.IDT_MiniSmartII.device_setDateTime (string *ident* = " ")

Set Date Time

Set current system date and time to the device.

Parameters

<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.
--------------	--

Returns

RETURN_CODE: Values can be parsed with `device_getResponseCodeString`

21.9.2.23 **RETURN_CODE** IDTechSDK.IDT_MiniSmartII.device_StartRKI (int *type*, string *ident* = " ")

Start Remote Key Injection

Starts a remote key injection request with IDTech RKI servers. Set/Get RKI url with `IDT_Device.RKI_URL`.

Parameters

<i>type</i>	0 = Symmetric RKI Demo Unit 1 = Symmetric RKI Production Unit 2 = PKI-RKI Demo Unit 3 = PKI-RKI Production Unit
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with `device_getResponseCodeString`

21.9.2.24 **RETURN_CODE** IDTechSDK.IDT_MiniSmartII.device_startRKI (string *ident* = " ")

Start Remote Key Injection

Starts a remote key injection request with IDTech RKI servers.

Parameters

<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.
--------------	--

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString

21.9.2.25 RETURN_CODE IDTechSDK.IDT_MiniSmartII.device_startTransaction (double *amount*, double *amtOther*, int *exponent*, int *type*, int *timeout*, byte[] *tags*, bool *forceOnline*, bool *isFastEMV* = false, string *ident* = " ")

Start EMV Transaction Request

Authorizes the EMV transaction for an ICC card

The tags will be returned in the callback routine.

Parameters

<i>amount</i>	Transaction amount value (tag value 9F02, string ident = "")
<i>amtOther</i>	Other amount value, if any (tag value 9F03, string ident = "")
<i>exponent</i>	Number of characters after decimile point
<i>type</i>	Transaction type (tag value 9C).
<i>timeout</i>	Timeout value in seconds.
<i>tags</i>	Any other tags to be included in the request. Passed as TLV Data. Example, tag 9F02 with amount 0x000000000100 would be 0x9F0206000000000100 If tags 9F02 (amount), 9F03 (other amount), or 9C (transaction type) are included, they will take priority over these values supplied as individual parameters to this method.
<i>forceOnline</i>	TRUE = do not allow offline approval, FALSE = allow ICC to approve offline if terminal capable Note: To request tags to be included in default response, use tag DFEE1A, and specify tag list. Example four tags 9F02, 9F36, 95, 9F37 to be included in response = DFEE1A079F029F369f9F37
<i>isFastEMV</i>	If TRUE, it will populate the IDTTransactionData.fastEMV with ASCII data similar to IDTech FastEMV KB output, after performing an auto-authenticate and auto-complete with ResultCode = Could Not Contact Host
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString

21.9.2.26 RETURN_CODE IDTechSDK.IDT_MiniSmartII.device_updateDeviceFirmware (byte[] *firmwareData*, string *ident* = " ")

Update Firmware

Updates the firmware .

Parameters

<i>firmwareData</i>	Signed binary data of a firmware file provided by IDTech
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with `errorCode.getErrorString(string ident = "")`

After you pass the `firmwareData` file, a new thread will start to execute the firmware download. You will receive status of the progress through callbacks to the `IDTechSDK.Callback()` delegate. The following parameters will be passed back:

- `sender = IDT_DEVICE_Types.IDT_DEVICE_AUGUSTA`
- `state = DeviceState.FirmwareUpdate`
- `transactionResultCode = status of the firmware update (starting, entering bootloader, applying update, block success, firmware success, string ident = "")`
- `data = File Progress. Four bytes, with bytes [0][1] = current block, and bytes [2][3] = total blocks. 0x00030010 = block 3 of 16`

Example code starting a firmware update

```
OpenFileDialog diag = new OpenFileDialog( ident);
diag.Filter = "NGA FW Files|*.fm";

if (diag.ShowDialog() == DialogResult.OK, string ident = "")
{
    byte[] file = File.ReadAllBytes(diag.FileName, ident);
    RETURN_CODE rt = IDT_Device.SharedController.device_updateDeviceFirmware(file, ident);
    if (rt == RETURN_CODE.RETURN_CODE_DO_SUCCESS, string ident = "")
    {
        //Was a success
    }
    else
    {
        //Error starting firmware download
    }
}
```

Example monitoring firmware update status / success

```
private void MessageCallBack(IDTechSDK.IDT_DEVICE_Types type, DeviceState state, byte[] data,
    IDTTransactionData cardData, EMV_Callback emvCallback, RETURN_CODE transactionResultCode, string ident = "")
{
    switch (state, string ident = "")
    {
        case DeviceState.FirmwareUpdate:
            switch (transactionResultCode, string ident = "")
            {
                case RETURN_CODE.RETURN_CODE_FW_STARTING_UPDATE:
                    SetOutputText("Starting Firmware Update\n", ident);
                    break;
                case RETURN_CODE.RETURN_CODE_DO_SUCCESS:
                    SetOutputText("Firmware Update Successful\n", ident);
                    break;
                case RETURN_CODE.RETURN_CODE_APPLYING_FIRMWARE_UPDATE:
                    SetOutputText("Applying Firmware Update...\n", ident);
                    break;
                case RETURN_CODE.RETURN_CODE_ENTERING_BOOTLOADER_MODE:
                    SetOutputText("Entering Bootloader Mode...\n", ident);
                    break;
                case RETURN_CODE.RETURN_CODE_BLOCK_TRANSFER_SUCCESS:
                    int start = data[0] * 0x100 + data[1];
                    int end = data[2] * 0x100 + data[3];

                    SetOutputText("Sent block " + start.ToString() + " of " + end.ToString() + "\n", ident);
                    break;
                default:
                    SetOutputText("Firmware Update Error Code: " + "0x" + String.Format("{0:X}", (ushort)
transactionResultCode) + ": " + IDTechSDK.errorCode.getErrorString(transactionResultCode) + "\r\n",
ident);
                    break;
            }
        break;
    }
}
```

21.9.2.27 `RETURN_CODE IDTechSDK.IDT_MiniSmartII.device_updateFirmwareFromZip (byte[] zipfile, string ident = " ", bool isForeground = false)`

Update Firmware From Zip

Performs one or more device firmware updates from firmware files passed as a compressed archive with a valid configuration file (.json format)

Parameters

<i>zipfile</i>	Zip file containing on or more firmware update files and a .json configuration file
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.
<i>isForeground</i>	If TRUE, will block program until firmware update complete. Otherwise, FALSE performs FW update on background.

Returns

RETURN_CODE: Values can be parsed with `errorCode.getErrorString(string ident = "")`

21.9.2.28 `RETURN_CODE IDTechSDK.IDT_MiniSmartII.emv_activateTransaction (int timeout, byte[] tags, bool forceOnline, bool isFastEMV = false, string ident = " ")`

Start EMV Transaction Request

Authorizes the EMV transaction for an ICC card

The tags will be returned in the callback routine.

Parameters

<i>amount</i>	Transaction amount value (tag value 9F02, string ident = "")
<i>amtOther</i>	Other amount value, if any (tag value 9F03, string ident = "")
<i>exponent</i>	Number of characters after decimile point
<i>type</i>	Transaction type (tag value 9C).
<i>timeout</i>	Timeout value in seconds.
<i>tags</i>	Any other tags to be included in the request. Passed as TLV Data. Example, tag 9F02 with amount 0x000000000100 would be 0x9F0206000000000100 If tags 9F02 (amount), 9F03 (other amount), or 9C (transaction type) are included, they will take priority over these values supplied as individual parameters to this method.
<i>forceOnline</i>	TRUE = do not allow offline approval, FALSE = allow ICC to approve offline if terminal capable Note: To request tags to be included in default response, use tag DFEE1A, and specify tag list. Example four tags 9F02, 9F36, 95, 9F37 to be included in response = DFEE1A079F029F369F9F37
<i>isFastEMV</i>	If TRUE, it will populate the <code>IDTTransactionData.fastEMV</code> with ASCII data similar to IDTech FastEMV KB output, after performing an auto-authenticate and auto-complete with ResultCode = Could Not Contact Host
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with `device_getResponseCodeString`

21.9.2.29 **RETURN_CODE** IDTechSDK.IDT_MiniSmartII.emv_addTerminalData (byte[] *tlv*, string *ident* = " ")

Add Terminal Data

Adds to the exosting Terminal Data

Parameters

<i>tlv</i>	TerminalData TLV Data
<i>ident</i>	Optional identifier

Return values

RETURN_CODE	Return codes listed as typedef enum in IDTCommon:RETURN_CODE. Values can be parsed with IDT_BTPay::device_getResponseCodeString:()
--------------------	--

21.9.2.30 static void IDTechSDK.IDT_MiniSmartII.emv_allowFallback (bool *allow*, string *ident* = " ") [static]

Allow fallback for EMV transactions. Default is TRUE

Parameters

<i>allow</i>	TRUE = allow fallback, FALSE = don't allow fallback
--------------	---

21.9.2.31 **RETURN_CODE** IDTechSDK.IDT_MiniSmartII.emv_authenticateTransaction (byte[] *updatedTLV*, string *ident* = " ")

Authenticate EMV Transaction Request

Authenticates the EMV transaction for an ICC card. Execute this after receiving response with result code 0x10 to emv_startTransaction

The tags will be returned in the callback routine.

@param updatedTLV TLV stream that can be used to update the following values:

- 9F02: Amount
- 9F03: Other amount
- 9C: Transaction type
- 5F57: Account type

In addition tag DFEE1A can be sent to specify tag list to include in results. Example four tags 9F02, 9F36, 95

@param ident Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString

21.9.2.32 static void IDTechSDK.IDT_MiniSmartII.emv_autoAuthenticate (bool *authenticate*, string *ident* = " ") [static]

Enables authenticate for EMV transactions. If a startEMVTransaction results in code 0x0010 (start transaction success), then emv_authenticateEMVTransaction can automatically execute if parameter is set to TRUE

Parameters

<i>authenticate</i>	TRUE = auto authenticate, FALSE = manually authenticate
---------------------	---

21.9.2.33 `static void IDTechSDK.IDT_MiniSmartII.emv_autoAuthenticateTags (bool authenticate, byte[] tags, string ident = " ") [static]`

Enables authenticate for EMV transactions. If a startEMVTransaction results in code 0x0010 (start transaction success), then emv_authenticateEMVTransaction can automatically execute if parameter is set to TRUE

Parameters

<i>authenticate</i>	TRUE = auto authenticate, FALSE = manually authenticate
<i>tags</i>	Tags to pass during authentication stage;

21.9.2.34 `RETURN_CODE IDTechSDK.IDT_MiniSmartII.emv_callbackResponseLCD (EMV_LCD_DISPLAY_MODE type, byte selection, string ident = " ")`

Callback Response LCD Display

Provides menu selection responses to the kernel after a callback was received with DeviceState.EMVCallback, and callbackType = EMV_CALLBACK_TYPE.EMV_CALLBACK_TYPE_LCD, and lcd_displayMode = EMV_LCD_DISPLAY_MODE_MENU, EMV_LCD_DISPLAY_MODE_PROMPT, or EMV_LCD_DISPLAY_MODE_LANGUAGE_SELECT

Parameters

<i>type</i>	If Cancel key pressed during menu selection, then value is EMV_LCD_DISPLAY_MODE_CANCEL. Otherwise, value can be EMV_LCD_DISPLAY_MODE_MENU, EMV_LCD_DISPLAY_MODE_PROMPT, or EMV_LCD_DISPLAY_MODE_LANGUAGE_SELECT
<i>selection</i>	If type = EMV_LCD_DISPLAY_MODE_MENU or EMV_LCD_DISPLAY_MODE_LANGUAGE_SELECT, provide the selection ID line number. Otherwise, if type = EMV_LCD_DISPLAY_MODE_PROMPT supply either 0x43 ('C') for Cancel, or 0x45 ('E') for Enter/accept
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with errorCode.getErrorString()

21.9.2.35 `RETURN_CODE IDTechSDK.IDT_MiniSmartII.emv_callbackResponseMSR (byte[] MSR, string ident = " ")`

Callback Response MSR Entry

Provides MSR information to kernel after a callback was received with DeviceState.EMVCallback, and callbackType = EMV_CALLBACK_MSR

Parameters

<i>MSR</i>	Swiped track data
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with errorCode.getErrorString()

21.9.2.36 `RETURN_CODE IDTechSDK.IDT_MiniSmartI11.emv_callbackResponsePIN (EMV_PIN_MODE type, byte[] KSN, byte[] PIN, string ident = " ")`

Callback Response PIN Entry

Provides (or cancels) PIN entry information to kernel after a callback was received with DeviceState.EMVCallback, and callbackType = EMV_CALLBACK_TYPE_PINPAD

Parameters

<i>type</i>	If Cancel key pressed during PIN entry, then value is EMV_PIN_MODE_CANCEL. Otherwise, value can be EMV_PIN_MODE_ONLINE_DUKPT, EMV_PIN_MODE_ONLINE_MKSK, or EMV_PIN_MODE_OFFLINE
<i>KSN</i>	If enciphered PIN, this is either the PIN DUKPT Key (EMV_PIN_MODE_ONLINE_DUKPT) or PIN Session Key (EMV_PIN_MODE_ONLINE_MKSK), or PIN Pairing DUKPT key (EMV_PIN_MODE_OFFLINE, string ident = "")
<i>PIN</i>	If enciphered PIN, this is encrypted PIN block. If device does not implement pairing function, this is plaintext PIN
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString

21.9.2.37 `RETURN_CODE IDTechSDK.IDT_MiniSmartI11.emv_cancelTransaction (string ident = " ")`

Cancel EMV Transaction

Cancels the currently executing EMV transaction.

Parameters

<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.
--------------	--

Returns

RETURN_CODE: Values can be parsed with errorCode.getErrorString()

21.9.2.38 `RETURN_CODE IDTechSDK.IDT_MiniSmartI11.emv_completeTransaction (bool commError, byte[] authCode, byte[] iad, byte[] tlvScripts, byte[] tlv, string ident = " ")`

Complete EMV Transaction Request

Completes the EMV transaction for an ICC card when online authorization request is received from emv_← authenticateTransaction

The tags will be returned in the callback routine.

Parameters

<i>commError</i>	Communication error with host. Set to TRUE if host was unreachable, or FALSE if host response received. If Communication error, authCode, iad, tlvScripts can be null.
<i>authCode</i>	Authorization code from host. Two bytes. Example 0x3030. (Tag value 8A). Required
<i>iad</i>	Issuer Authentication Data, if any. Example 0x11223344556677883030 (tag value 91).
<i>tlvScripts</i>	71/72 scripts, if any

Parameters

<i>tlv</i>	Additional TVL data to return with transaction results (if any, string ident = "")
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString

NOTE: There are three possible outcomes for Authorization Code: Approval, Refer To Bank, Decline. The kernel maps these three outcomes to valid authorization response codes using tag DFEE1B through 8 bytes: {2 bytes Approval Code}{2 bytes Referral Code}{2 bytes Decline Code}{2 bytes RFU} If your gateway uses "00" for Approval, "01" for Referral, and "05" for Decline, then DFEE1B 08 3030 3031 3035 0000 If you use an authorization code value that is not defined in DFEE1B, the kernel will use the DECLINE value of DFEE1B by default.

21.9.2.39 RETURN_CODE IDTechSDK.IDT_MiniSmartII.emv_getEMVConfigurationCheckValue (ref string response, string ident = " ")

Get EMV Kernel configuration check value info

Parameters

<i>response</i>	Response returned of Kernel configuration check value info
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString

21.9.2.40 RETURN_CODE IDTechSDK.IDT_MiniSmartII.emv_getEMVKernelCheckValue (ref string response, string ident = " ")

Get EMV Kernel check value info

Parameters

<i>response</i>	Response returned of Kernel check value info
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString

21.9.2.41 RETURN_CODE IDTechSDK.IDT_MiniSmartII.emv_getEMVKernelVersion (ref string response, string ident = " ")

Polls device for EMV Kernel Version

Parameters

<i>response</i>	Response returned of Kernel Version
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString

21.9.2.42 RETURN_CODE IDTechSDK.IDT_MiniSmartII.emv_removeAllApplicationData (string *ident* = " ")

Remove All Application Data

Removes all the Application Data

Parameters

<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.
--------------	--

Returns

RETURN_CODE: Values can be parsed with errorCode.getErrorString(string ident = "")

21.9.2.43 RETURN_CODE IDTechSDK.IDT_MiniSmartII.emv_removeAllCAPK (string *ident* = " ")

Remove All Certificate Authority Public Key

Removes all the CAPK

Parameters

<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.
--------------	--

Returns

RETURN_CODE: Values can be parsed with errorCode.getErrorString(string ident = "")

21.9.2.44 RETURN_CODE IDTechSDK.IDT_MiniSmartII.emv_removeAllCRL (string *ident* = " ")

Remove All Certificate Revocation List Entries

Removes all CRLEntry entries

Parameters

<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.
--------------	--

Returns

RETURN_CODE: Values can be parsed with errorCode.getErrorString(string ident = "")

21.9.2.45 RETURN_CODE IDTechSDK.IDT_MiniSmartII.emv_removeApplicationData (byte[] *AID*, string *ident* = " ")

Remove Application Data by AID

Removes the Application Data as specified by the AID name passed as a parameter

Parameters

<i>AID</i>	Name of ApplicationID Must be between 5 and 16 bytes
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with `errorCode.getErrorString(string ident = "")`

21.9.2.46 RETURN_CODE IDTechSDK.IDT_MiniSmartII.emv_removeCAPK (byte[] *capk*, string *ident* = " ")

Remove Certificate Authority Public Key

Removes the CAPK as specified by the RID/Index

Parameters

<i>capk</i>	6 byte CAPK = 5 bytes RID + 1 byte INDEX
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with `errorCode.getErrorString(string ident = "")`

21.9.2.47 RETURN_CODE IDTechSDK.IDT_MiniSmartII.emv_removeCRL (byte[] *crlList*, string *ident* = " ")

Remove Certificate Revocation List Entries

Removes CRL entries as specified by the RID and Index and serial number passed as 9 bytes

Parameters

<i>crlList</i>	containing the list of CRL to remove: [CRL1][CRL2]...[CRLn] where each [CRL] is 9 bytes: [5 bytes RID][1 byte CAPK Index][3 bytes serial number]
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with `device_getResponseCodeString`

21.9.2.48 RETURN_CODE IDTechSDK.IDT_MiniSmartII.emv_removeTerminalData (string *ident* = " ")

Remove Terminal Data

Removes the Terminal Data

Parameters

<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.
--------------	--

Returns

RETURN_CODE: Values can be parsed with `errorCode.getErrorString(string ident = "")`

21.9.2.49 RETURN_CODE IDTechSDK.IDT_MiniSmartII.emv_retrieveAIDList (ref byte *response*[], string *ident* = " ")

Retrieve AID list

Returns all the AID names installed on the terminal.

Parameters

<i>response</i>	array of AID name byte arrays
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with `errorCode.getErrorString(string ident = "")`

21.9.2.50 RETURN_CODE IDTechSDK.IDT_MiniSmartII.emv_retrieveApplicationData (byte[] *AID*, ref byte[] *tlv*, string *ident* = " ")

Retrieve Application Data by AID

Retrieves the Application Data as specified by the AID name passed as a parameter.

Parameters

<i>AID</i>	Name of ApplicationID. Must be between 5 and 16 bytes
<i>tlv</i>	The TLV elements of the requested AID
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with `errorCode.getErrorString()`

21.9.2.51 RETURN_CODE IDTechSDK.IDT_MiniSmartII.emv_retrieveCAPK (byte[] *capk*, ref byte[] *key*, string *ident* = " ")

Retrieve Certificate Authority Public Key

Retrieves the CAPK as specified by the RID/Index passed as a parameter.

Parameters

<i>capk</i>	6 bytes CAPK = 5 bytes RID + 1 byte Index
-------------	---

Parameters

<i>key</i>	<p>Response returned as a CAKey format: [5 bytes RID][1 byte Index][1 byte Hash Algorithm][1 byte Encryption Algorithm][20 bytes HashValue][4 bytes Public Key Exponent][2 bytes Modulus Length][Variable bytes Modulus] Where:</p> <ul style="list-style-type: none"> • Hash Algorithm: The only algorithm supported is SHA-1. The value is set to 0x01 • Encryption Algorithm: The encryption algorithm in which this key is used. Currently support only one type: RSA. The value is set to 0x01. • HashValue: Which is calculated using SHA-1 over the following fields: RID & Index & Modulus & Exponent • Public Key Exponent: Actually, the real length of the exponent is either one byte or 3 bytes. It can have two values: 3 (Format is 0x00 00 00 03), or 65537 (Format is 0x00 01 00 01, string ident = "") • Modulus Length: LenL LenH Indicated the length of the next field. • Modulus: This is the modulus field of the public key. Its length is specified in the field above.
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with `errorCode.getErrorString()`

21.9.2.52 RETURN_CODE IDTechSDK.IDT_MiniSmartII.emv_retrieveCAPKList (ref byte[] keys, string ident = " ")

Retrieve the Certificate Authority Public Key list

Returns all the CAPK RID and Index installed on the terminal.

Parameters

<i>keys</i>	[key1][key2]...[keyn], each key 6 bytes where key = 5 bytes RID + 1 byte index
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with `errorCode.getErrorString()`

21.9.2.53 RETURN_CODE IDTechSDK.IDT_MiniSmartII.emv_retrieveCRLList (ref byte[] list, string ident = " ")

Retrieve the Certificate Revocation List

Returns the CRL entries on the terminal.

Parameters

<i>list</i>	[CRL1][CRL2]...[CRLn], each CRL 9 bytes where CRL = 5 bytes RID + 1 byte index + 3 bytes serial number
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with `errorCode.getErrorString()`

21.9.2.54 RETURN_CODE IDTechSDK.IDT_MiniSmartII.emv_retrieveTerminalData (ref byte[] *tlv*, string *ident* = " ")

Retrieve Terminal Data

Retrieves the Terminal Data.

Parameters

<i>tlv</i>	Response returned as a TLV
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with `errorCode.getErrorString()`

21.9.2.55 RETURN_CODE IDTechSDK.IDT_MiniSmartII.emv_retrieveTransactionResult (byte[] *tags*, ref IDTTransactionData *tlv*, string *ident* = " ")

Retrieve Transaction Results

Retrieves specified EMV tags from the currently executing transaction.

Parameters

<i>tags</i>	Tags to be retrieved. Example 0x9F028A will retrieve tags 9F02 and 8A
<i>tlv</i>	All requested tags returned as unencrypted, encrypted and masked TLV data in IDTTransactionData object
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with `device.getResponseCodeString`

21.9.2.56 RETURN_CODE IDTechSDK.IDT_MiniSmartII.emv_setApplicationData (byte[] *name*, byte[] *tlv*, string *ident* = " ")

Set Application Data by AID

Sets the Application Data as specified by the application name and TLV data

Parameters

<i>name</i>	Application name, 10-32 ASCII hex characters representing 5-16 bytes Example "a0000000031010"
<i>tlv</i>	Application data in TLV format
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with `errorCode.getErrorString()`

21.9.2.57 RETURN_CODE IDTechSDK.IDT_MiniSmartII.emv_setCAPK (byte[] key, string ident = " ")

Set Certificate Authority Public Key

Sets the CAPK as specified by the CAKey structure

Parameters

<i>key</i>	CAKey format: [5 bytes RID][1 byte Index][1 byte Hash Algorithm][1 byte Encryption Algorithm][20 bytes HashValue][4 bytes Public Key Exponent][2 bytes Modulus Length][Variable bytes Modulus] Where: <ul style="list-style-type: none"> • Hash Algorithm: The only algorithm supported is SHA-1. The value is set to 0x01 • Encryption Algorithm: The encryption algorithm in which this key is used. Currently support only one type: RSA. The value is set to 0x01. • HashValue: Which is calculated using SHA-1 over the following fields: RID & Index & Modulus & Exponent • Public Key Exponent: Actually, the real length of the exponent is either one byte or 3 bytes. It can have two values: 3 (Format is 0x00 00 00 03), or 65537 (Format is 0x00 01 00 01, string ident = "") • Modulus Length: LenL LenH Indicated the length of the next field. • Modulus: This is the modulus field of the public key. Its length is specified in the field above.
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with errorCode.getErrorString()

21.9.2.58 RETURN_CODE IDTechSDK.IDT_MiniSmartII.emv_setCRL (byte[] list, string ident = " ")

Set Certificate Revocation List

Sets the CRL

Parameters

<i>list</i>	CRL Entries containing the RID, Index, and serial numbers to set [CRL1][CRL2]...[CRLn] where each [CRL] is 9 bytes: [5 bytes RID][1 byte CAPK Index][3 bytes serial number]
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with errorCode.getErrorString()

21.9.2.59 RETURN_CODE IDTechSDK.IDT_MiniSmartII.emv_setTerminalData (byte[] tlv, string ident = " ")

Set Terminal Data

Sets the Terminal Data

Parameters

<i>tlv</i>	TerminalData TLV Data
<i>ident</i>	Optional identifier

Return values

<i>RETURN_CODE</i>	Return codes listed as typedef enum in IDTCommon:RETURN_CODE. Values can be parsed with IDT_BTPay::device_getResponseCodeString:()
--------------------	--

21.9.2.60 **RETURN_CODE** IDTechSDK.IDT_MiniSmartIII.emv_setTerminalMajorConfiguration (int *configuration*, string *ident* = " ")

Set Terminal Major Configuration

Sets the Terminal Data Major Configuration setting

Parameters

<i>configuration</i>	The configuration to set (1-23, string ident = "")
----------------------	--

Return values

<i>RETURN_CODE</i>	Return codes listed as typedef enum in IDTCommon:RETURN_CODE. Values can be parsed with IDT_BTPay::device_getResponseCodeString:(string ident = "")
--------------------	--

21.9.2.61 **RETURN_CODE** IDTechSDK.IDT_MiniSmartIII.emv_startTransaction (double *amount*, double *amtOther*, int *exponent*, int *type*, int *timeout*, byte[] *tags*, bool *forceOnline*, bool *isFastEMV* = false, string *ident* = " ")

Start EMV Transaction Request

Authorizes the EMV transaction for an ICC card

The tags will be returned in the callback routine.

Parameters

<i>amount</i>	Transaction amount value (tag value 9F02, string ident = "")
<i>amtOther</i>	Other amount value, if any (tag value 9F03, string ident = "")
<i>exponent</i>	Number of characters after decimile point
<i>type</i>	Transaction type (tag value 9C).
<i>timeout</i>	Timeout value in seconds.
<i>tags</i>	Any other tags to be included in the request. Passed as TLV Data. Example, tag 9F02 with amount 0x000000000100 would be 0x9F0206000000000100 If tags 9F02 (amount), 9F03 (other amount), or 9C (transaction type) are included, they will take priority over these values supplied as individual parameters to this method.
<i>forceOnline</i>	TRUE = do not allow offline approval, FALSE = allow ICC to approve offline if terminal capable Note: To request tags to be included in default response, use tag DFEE1A, and specify tag list. Example four tags 9F02, 9F36, 95, 9F37 to be included in response = DFEE1A079F029F369F9F37
<i>isFastEMV</i>	If TRUE, it will populate the IDTTransactionData.fastEMV with ASCII data similar to IDTech FastEMV KB output, after performing an auto-authenticate and auto-complete with ResultCode = Could Not Contact Host
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString

21.9.2.62 **RETURN_CODE** IDTechSDK.IDT_MiniSmartII.emv_trySetTerminalData (byte[] *tlv*, ref byte[] *rejectedTLV*, ref byte[] *convertedTLV*, bool *overwrite* = false, string *ident* = " ")

Try to Set Terminal Data

Attempts to set the Terminal Data. Use this functions to attempt to set the terminal data to the device. This function will allow the cross-device tag compatible writing of IDTech Legacy/New tag definitions

Parameters

<i>tlv</i>	TerminalData TLV Data
<i>rejectedTLV</i>	Contains the tags (if any) that were rejected by the firmware
<i>convertedTLV</i>	Contains the tags (if any) that were converted and accepted by the firmware
<i>overwrite</i>	TRUE = add TLV to existing tags, FALSE = replace existing tags with TLV
<i>ident</i>	Optional identifier

Return values

RETURN_CODE	Return codes listed as typedef enum in IDTCommon:RETURN_CODE. Values can be parsed with IDT_BTPay::device_getResponseCodeString:()
--------------------	--

21.9.2.63 **static int** IDTechSDK.IDT_MiniSmartII.getCommandTimeout (string *ident* = " ") [static]

Get Command Timeout

Gets the default timeout (in milliseconds) waiting for a blocking command response

Return values

<i>time</i>	Time
-------------	------

21.9.2.64 **RETURN_CODE** IDTechSDK.IDT_MiniSmartII.icc_exchangeAPDU (string *c_APDU*, ref byte[] *response*, string *ident* = " ")

Exchange APDU

Sends an APDU packet to the ICC. If successful, response is returned in APDUResult class instance in response parameter.

Parameters

<i>c_APDU</i>	APDU data packet
<i>response</i>	Unencrypted/encrypted parsed APDU response
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with `errorCode.getErrorString(string ident = "")`

21.9.2.65 RETURN_CODE IDTechSDK.IDT_MiniSmartII.icc_getAPDU_KSN (ref byte[] *ksn*, string *ident* = " ")

Get APDU KSN

Retrieves the KSN used in ICC Encrypted APDU usage

Parameters

<i>ksn</i>	Returns the encrypted APDU packet KSN
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with `errorCode.getErrorString(string ident = "")`

21.9.2.66 RETURN_CODE IDTechSDK.IDT_MiniSmartII.icc_getClearPANID (ref byte *prePAN*, ref byte *postPAN*, string *ident* = " ")

Get Pre/Post PAN Data Len**Parameters**

<i>prePAN</i>	Pre PAN length
<i>postPAN</i>	Post PAN length
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with `errorCode.getErrorString(string ident = "")`

21.9.2.67 RETURN_CODE IDTechSDK.IDT_MiniSmartII.icc_getICCReaderStatus (ref byte *status*, string *ident* = " ")

Get Reader Status

Returns the reader status

Parameters

<i>status</i>	Pointer that will return with the ICCReaderStatus results. bit 0: 0 = ICC Power Not Ready, 1 = ICC Powered bit 1: 0 = Card not seated, 1 = card seated
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with `errorCode.getErrorString(string ident = "")`

21.9.2.68 RETURN_CODE IDTechSDK.IDT_MiniSmartII.icc_getKeyFormatForICCDUKPT (ref byte *format*, string *ident* = " ")

Get Key Format For ICC DUKPT

Specifies how data will be encrypted with Data Key or PIN key (if DUKPT key loaded, string *ident* = "")

Parameters

<i>format</i>	Response returned from method: <ul style="list-style-type: none"> • 'TDES': Encrypted card data with TDES if DUKPT Key had been loaded.(default, string <i>ident</i> = "") • 'AES': Encrypted card data with AES if DUKPT Key had been loaded. • 'NONE': No Encryption.
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with `errorCode.getErrorString(string ident = "")`

21.9.2.69 RETURN_CODE IDTechSDK.IDT_MiniSmartII.icc_getKeyTypeForICCDUKPT (ref byte *type*, string *ident* = " ")

Get Key Type for ICC DUKPT

Specifies the key type used for ICC DUKPT encryption

Parameters

<i>type</i>	Response returned from method: <ul style="list-style-type: none"> • 'DATA': Encrypted card data with Data Key DUKPT Key had been loaded.(default, string <i>ident</i> = "") • 'PIN': Encrypted card data with PIN Key if DUKPT Key had been loaded.
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with `errorCode.getErrorString(string ident = "")`

21.9.2.70 RETURN_CODE IDTechSDK.IDT_MiniSmartII.icc_getReadingCharacteristics (ref string *mode*, string *ident* = " ")

Get ICC Reading Characteristics

Gets the ICC reading characteristics: ICC ON/OFF and enable insert/removal notifications

Parameters

<i>mode</i>	Reading Characterists Mode <ul style="list-style-type: none"> • "0": ICC Functions OFF • "1": ICC Functions ON, Notifications OFF • "2": ICC Functions ON, Notifications ON
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with `errorCode.getErrorString(string ident = "")`

Note: If ICC Reading Function Off, the following functions will return an error:

- Power On (Get ATR, string ident = "")
- Exchange APDU Encryption for special case
- Exchange APDU Encryption
- Start Transaction
- Authenticate Transaction
- Complete Transaction

21.9.2.71 RETURN_CODE IDTechSDK.IDT_MiniSmartll.icc_powerOffICC (string *ident* = " ")

Power Off ICC

Powers down the ICC

Parameters

<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.
--------------	--

Returns

RETURN_CODE: Values can be parsed with `errorCode.getErrorString(string ident = "")`

If Success, empty If Failure, ASCII encoded data of error string

21.9.2.72 RETURN_CODE IDTechSDK.IDT_MiniSmartll.icc_powerOnICC (ref byte[] *ATR*, string *ident* = " ")

Power On ICC

Power up the currently selected microprocessor card in the ICC reader

Parameters

<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.
--------------	--

Returns

RETURN_CODE: Values can be parsed with `errorCode.getErrorString(string ident = "")`

21.9.2.73 RETURN_CODE IDTechSDK.IDT_MiniSmartll.icc_setClearPANID (byte *prePAN*, byte *postPAN*, string *ident* = " ")

Set Pre/Post PAN Data Len

Parameters

<i>prePAN</i>	Pre PAN length, needs to be 0~6, default is 4
<i>postPAN</i>	Post PAN length, needs to be 0~4, default is 4
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with `errorCode.getErrorString(string ident = "")`

21.9.2.74 RETURN_CODE IDTechSDK.IDT_MiniSmartII.icc_setKeyFormatForICCDUKPT (byte *encryption*, string *ident* = " ")

Set Key Format for ICC DUKPT

Sets how data will be encrypted, with either TDES or AES (if DUKPT key loaded, string *ident* = "")

Parameters

<i>encryption</i>	Encryption Type <ul style="list-style-type: none"> • 00: Encrypt with TDES • 01: Encrypt with AES
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with `errorCode.getErrorString(string ident = "")`

21.9.2.75 RETURN_CODE IDTechSDK.IDT_MiniSmartII.icc_setKeyTypeForICCDUKPT (byte *encryption*, string *ident* = " ")

Set Key Type for ICC DUKPT Key

Sets which key the data will be encrypted with, with either Data Key or PIN key (if DUKPT key loaded, string *ident* = "")

Parameters

<i>encryption</i>	Encryption Type <ul style="list-style-type: none"> • 00: Encrypt with Data Key • 01: Encrypt with PIN Key
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with `errorCode.getErrorString(string ident = "")`

21.9.2.76 RETURN_CODE IDTechSDK.IDT_MiniSmartII.icc_setReadingCharacteristics (int *mode*, string *ident* = " ")

Set ICC Reading Characteristics

Sets the ICC reading characteristics: ICC ON/OFF and enable insert/removal notifications

Parameters

<i>mode</i>	Reading Characterists Mode <ul style="list-style-type: none"> • 0: ICC Functions OFF • 1: ICC Functions ON, Notifications OFF • 2: ICC Functions ON, Notifications ON
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with `errorCode.getErrorString(string ident = "")`

Note: If ICC Reading Function Off, the following functions will return an error:

- Power On (Get ATR, string ident = "")
- Exchange APDU Encryption for special case
- Exchange APDU Encryption
- Start Transaction
- Authenticate Transaction
- Complete Transaction

21.9.2.77 `static void IDTechSDK.IDT_MiniSmartII.Icd_retrieveMessage (DisplayMessages.DISPLAY_MESSAGE_LANGUAGE lang, DisplayMessages.DISPLAY_MESSAGE_IDENTIFIER id, ref string line1, ref string line2) [static]`

Retrieve LCD Message

Returns the string value for a message ID returned for LCD messaging

Parameters

<i>lang</i>	Language.
<i>id</i>	Message ID
<i>line1</i>	Line 1 string value
<i>line2</i>	Line 2 string value

21.9.2.78 `static String IDTechSDK.IDT_MiniSmartII.SDK_Version () [static]`

SDK Version

- All Devices

Returns the current version of SDK

Returns

Framework version

21.9.2.79 `static void IDTechSDK.IDT_MiniSmartII.setCallback (Callback my_Callback) [static]`

Set Callback

Sets the class callback

21.9.2.80 `static void IDTechSDK.IDT_MiniSmartII.setCallback (IntPtr my_Callback, SynchronizationContext context) [static]`

Set Callback

Sets the class callback

Parameters

<i>my_Callback</i>	The callback function to receive the response message from device. defined as follows. public unsafe delegate void MFCCallBack(Parameters parameters, ident);
<i>context</i>	The context of the UI thread

21.9.2.81 `static void IDTechSDK.IDT_MiniSmartII.setCommandTimeout (int milliseconds, string ident = " ") [static]`

Set Command Timeout

Sets the default timeout (in milliseconds) waiting for a blocking command response

Parameters

<i>milliseconds</i>	Time
---------------------	------

21.9.2.82 `static bool IDTechSDK.IDT_MiniSmartII.useSerialPort (int port) [static]`

Use Serial Port Interface

Instructs SDK to attempt to use the Serial Port for communication with MiniSmartII using default baud rate

Parameters

<i>port</i>	Serial Port to use. Example COM1 = 1.
-------------	---------------------------------------

Returns

bool TRUE=successful, FALSE=failure

21.9.2.83 `static bool IDTechSDK.IDT_MiniSmartII.useSerialPort (int port, int baud) [static]`

Use Serial Port Interface with baud rate

Instructs SDK to attempt to use the Serial Port for communication with MiniSmartII

Parameters

<i>port</i>	Serial Port to use. Example COM1 = 1.
<i>baud</i>	Baud rate to override default. Example 115200;

Returns

bool TRUE=successful, FALSE=failure

21.9.2.84 static bool IDTechSDK.IDT_MiniSmartII.useSerialPortLinux (string *path*) [static]

Use Serial Port Interface on Linux

Instructs SDK to attempt to use the Serial Port for communication with BTMag using default baud rate on Linux implementations

Parameters

<i>path</i>	Path to use. Example /dev/ttyUSB*
-------------	-----------------------------------

Returns

bool TRUE=successful, FALSE=failure

21.9.2.85 static bool IDTechSDK.IDT_MiniSmartII.useSerialPortLinux (string *path*, int *baud*) [static]

Use Serial Port Interface on Linux with baud rate

Instructs SDK to attempt to use the Serial Port for communication with BTPay on Linux implementations

Parameters

<i>path</i>	Path to use. Example /dev/ttyUSB*
<i>baud</i>	Baud rate to override default. Example 115200;

Returns

bool TRUE=successful, FALSE=failure

21.9.3 Property Documentation

21.9.3.1 IDT_MiniSmartII IDTechSDK.IDT_MiniSmartII.SharedController [static],[get]

Singleton Instance

Establishes an singleton instance of [IDT_MiniSmartII](#) class.

Returns

Instance of [IDT_MiniSmartII](#)

The documentation for this class was generated from the following file:

- /Users/randy/Repo/universal-sdk/IDTechSDK_STD/DeviceHeaders/IDT_MiniSmartII.cs

21.10 IDTechSDK.IDT_NEO2 Class Reference

Public Member Functions

- RETURN_CODE [device_loadCertCA](#) (byte certType, byte[] certData, string ident="")

- RETURN_CODE [config_setDUKPTEncryptionType](#) (byte type, string ident="")
- RETURN_CODE [config_getDUKPT_KSN](#) (ref byte[] ksn, string ident="")
- RETURN_CODE [config_getSalt_KCV](#) (ref byte[] kcv, string ident="")
- RETURN_CODE [config_getDUKPTEncryptionType](#) (ref byte type, string ident="")
- RETURN_CODE [config_setKeyslot_PEK_DEK](#) (bool isPEK, byte keyslot, string ident="")
- RETURN_CODE [device_resetNVM](#) (string ident="")
- RETURN_CODE [config_setRKLKeys](#) (short keyNumber, byte[] tr31, byte[] nonce, byte[] hmac, ref byte[] kv, ref byte[] nonce_device, ref byte[] hmac_device, string ident="")
- RETURN_CODE [config_getKeyslot_PEK_DEK](#) (ref byte keyslotPEK, ref byte keyslotDEK, string ident="")
- RETURN_CODE [config_setDUKPT_DEK_Attribution_TDES](#) (byte keyslot, byte outputMode, byte variant, string ident="")
- RETURN_CODE [config_setDUKPT_DEK_Attribution_AES](#) (byte keyslot, byte workingKey, byte keyUsage, string ident="")
- RETURN_CODE [config_getDUKPT_DEK_Attribution](#) (byte keyslot, ref byte mode, ref byte outputMode, ref byte workingKey, ref byte variant_keyUsage, string ident="")
- RETURN_CODE [config_checkDUKPTKey](#) (byte keyindex, ref byte[] val, string ident="")
- RETURN_CODE [config_setDEKVariantType](#) (byte type, string ident="")
- RETURN_CODE [config_getDEKVariantType](#) (ref byte type, string ident="")
- RETURN_CODE [config_setBaudRate](#) (int baud, string ident="")
- bool [ip_firstConnectToSocket](#) (string IP)
- RETURN_CODE [device_getDRS](#) (ref byte[] codeDRS, string ident="")
- bool [ip_connectToSocket](#) (string IP, bool isSecure=false)
- void [ip_autoConnectToSocket](#) ()
- void [ip_monitorSocketConnectionStatus](#) (bool enable, bool monitorConnect, int interval, int retryCount)
- RETURN_CODE [device_queryFile](#) (string directory, string filename, bool isSD, ref bool exists, ref string timestamp, ref int fileSize, string ident="")
- RETURN_CODE [device_readFileFromSD](#) (string directory, string filename, ref byte[] fileData, string ident="")
- RETURN_CODE [device_getBatteryVoltage](#) (ref string voltage, string ident="")
- RETURN_CODE [device_pollForToken](#) (byte seconds, ref byte card, ref byte[] serialNumber, string ident="")
- RETURN_CODE [config_getSerialNumber](#) (ref string response, string ident="")
- RETURN_CODE [device_terminalInfo](#) (ref byte[] tlv, string ident="")
- RETURN_CODE [config_setWhiteList](#) (byte[] data, string ident="")
- RETURN_CODE [config_setTrackFormat](#) (byte option, string ident="")
- RETURN_CODE [config_getTrackFormat](#) (ref byte option, string ident="")
- RETURN_CODE [config_setMasking](#) (byte prePAN, byte postPAN, byte asciiMask, byte hexMask, bool maskExp, string ident="")
- RETURN_CODE [config_getMasking](#) (ref byte prePAN, ref byte postPAN, ref byte asciiMask, ref byte hexMask, ref bool maskExp, string ident="")
- RETURN_CODE [config_getWhiteList](#) (ref Dictionary< string, string > data, string ident="")
- RETURN_CODE [config_getEthernetMACAddress](#) (ref byte[] address, string ident="")
- RETURN_CODE [config_getNetworkConfiguration](#) (ref bool isStatic, ref string address, ref string subnet, ref string gateway, ref string dns, string ident="")
- RETURN_CODE [config_setSSLServerEthernet](#) (string address, int port, string ident="")
- RETURN_CODE [config_getSSLServerEthernet](#) (ref string address, ref int port, string ident="")
- RETURN_CODE [device_getDeviceTime](#) (ref DateTime time, string ident="")
- RETURN_CODE [config_setBluetoothParameters](#) (string name, string oldPW, string newPW, string ident="")
- RETURN_CODE [config_setSwipeandDone](#) (byte swipeVal, byte doneVal, byte delay, string ident="")
- RETURN_CODE [config_getSwipeandDone](#) (ref byte swipeVal, ref byte doneVal, ref byte delay, string ident="")
- RETURN_CODE [device_disBlueLED](#) (string ident="")
- RETURN_CODE [device_enaBlueLED](#) (byte[] dataCmd, string ident="")
- RETURN_CODE [device_onYellowLED](#) (string ident="")
- RETURN_CODE [device_offYellowLED](#) (string ident="")
- RETURN_CODE [device_buzzerOnOff](#) (string ident="")
- RETURN_CODE [device_getLightSensorVal](#) (ref UInt16 lightVal, string ident="")

- RETURN_CODE [device_setSelfCheckTime](#) (byte hour, byte minutes, string ident="")
- RETURN_CODE [device_logRead](#) (DeviceLogCallback callback, string ident="")
- RETURN_CODE [device_logClear](#) (string ident="")
- RETURN_CODE [device_logEnable](#) (bool enable, string ident="")
- RETURN_CODE [device_extendedErrorCondition](#) (bool enable, string ident="")
- RETURN_CODE [device_getSelfCheckTime](#) (ref byte hour, ref byte minutes, string ident="")
- RETURN_CODE [config_setNetworkConfiguration](#) (bool isStatic, string address, string subnet, string gateway, string dns, string ident="")
- RETURN_CODE [config_setWifiConfig](#) (int mode, string ssid, string password, string ip, string netMask, string gateway, string ident="", string remoteIP=null, string remotePort=null)
- RETURN_CODE [config_getWifiConfig](#) (ref int mode, ref string ssid, ref string password, ref string ip, ref string netMask, ref string gateway, ref string remoteIP, ref string remotePort, string ident="")
- RETURN_CODE [device_getFirmwareVersion](#) (ref string response, string ident="")
- RETURN_CODE [device_getBLEName](#) (ref string name, string ident="")
- RETURN_CODE [device_getDeviceTreeVersion](#) (ref string response, bool isDeviceTree, string ident="")
- RETURN_CODE [device_get1050BootloaderVersion](#) (ref string response, string ident="")
- RETURN_CODE [device_get1050FuseStatus](#) (ref byte[] status, string ident="")
- RETURN_CODE [device_getBootloaderVersion](#) (ref string response, string ident="")
- RETURN_CODE [config_setWirelessWorkMode](#) (int mode, string ident="")
- RETURN_CODE [config_sendSSLRequestWiFi](#) (string address, int port, string ident="")
- RETURN_CODE [config_getWirelessWorkMode](#) (ref int mode, string ident="")
- RETURN_CODE [device_getRT1050FirmwareVersion](#) (ref string response, string ident="")
- bool [createFastEMVData](#) (ref IDTTransactionData cData, string ident="")
- RETURN_CODE [device_sendPAE](#) (string command, ref string response, int timeout, string ident="")
- RETURN_CODE [device_setTransArmorEncryption](#) (byte[] cert, string ident="")
- RETURN_CODE [device_getTransArmorID](#) (ref string TID, string ident="")
- RETURN_CODE [device_setTransArmorID](#) (string TID, string ident="")
- RETURN_CODE [device_retrieveAIDList](#) (ref byte[][] response, string ident="")
- RETURN_CODE [device_rebootDevice](#) (string ident="")
- RETURN_CODE [device_sendDataCommand](#) (string cmd, bool calcLRC, ref byte[] response, string ident="")
- RETURN_CODE [device_sendDataCommand_ext](#) (string cmd, bool calcLRC, ref byte[] response, int timeout, bool noResponse, string ident="")
- RETURN_CODE [msr_setMSRTrack](#) (int val, string ip, string ident="")
- RETURN_CODE [msr_getMSRTrack](#) (ref int val, string ident="")
- RETURN_CODE [msr_startMSRSwipe](#) (int timeout, string ident="")
- RETURN_CODE [msr_startMSRSwipe_ext](#) (int timeout, string ident, SwipeCallback swipeCallback, TimeoutCallback timeoutCallback, FailureCallback failureCallback)
- RETURN_CODE [msr_cancelMSRSwipe](#) (string ident="")
- RETURN_CODE [emv_cancelTransaction](#) (string ident="")
- RETURN_CODE [ctls_cancelTransaction](#) (string ident="")
- RETURN_CODE [emv_startTransactionCB](#) (double amount, double amtOther, int exponent, int type, int timeout, byte[] tags, bool forceOnline, CallBackIP callback, bool isFastEMV=false, string ident="")
- RETURN_CODE [emv_startTransaction](#) (double amount, double amtOther, int exponent, int type, int timeout, byte[] tags, bool forceOnline, bool isFastEMV=false, string ident="")
- RETURN_CODE [emv_activateTransaction](#) (int timeout, byte[] tags, bool forceOnline, bool isFastEMV=false, string ident="")
- RETURN_CODE [emv_authenticateTransactionCB](#) (byte[] updatedTLV, CallBackIP callback, string ident="")
- RETURN_CODE [emv_authenticateTransaction](#) (byte[] updatedTLV, string ident="")
- RETURN_CODE [ctls_startTransaction](#) (double amount, double amtOther, int exponent, int type, int timeout, byte[] tags, bool forceOnline, bool isFastEMV=false, string ident="")
- RETURN_CODE [ctls_activateTransaction](#) (int timeout, byte[] tags, bool forceOnline, bool isFastEMV=false, string ident="")
- RETURN_CODE [ctls_updateBalance](#) (byte statusCode, byte[] authCode, byte[] date, byte[] time, string ident="")
- RETURN_CODE [device_startTransactionCB](#) (double amount, double amtOther, int exponent, int type, int timeout, byte[] tags, CallBackIP callback, string ident="", bool isFastEMV=false)

- RETURN_CODE [device_startTransaction](#) (double amount, double amtOther, int exponent, int type, int timeout, byte[] tags, bool isFastEMV=false, string ident="")
- RETURN_CODE [device_activateTransaction](#) (int timeout, byte[] tags, bool isFastEMV=false, string ident="")
- RETURN_CODE [emv_completeTransactionCB](#) (bool commError, byte[] authCode, byte[] iad, byte[] tlvScripts, byte[] tlv, CallbackIP callback, string ident="")
- RETURN_CODE [emv_completeTransaction](#) (bool commError, byte[] authCode, byte[] iad, byte[] tlvScripts, byte[] tlv, string ident="")
- RETURN_CODE [emv_callbackResponseLCD](#) (EMV_LCD_DISPLAY_MODE type, byte selection, string ident="")
- RETURN_CODE [emv_callbackResponsePIN](#) (EMV_PIN_MODE type, byte[] KSN, byte[] PIN, string ident="")
- RETURN_CODE [emv_callbackResponsePIN_ETC](#) (EMV_PIN_MODE type, byte[] KSN, byte[] PIN, string ident="")
- RETURN_CODE [emv_generateDUKPT](#) (byte[] cert, byte[] signature, ref byte[] key, string ident="")
- RETURN_CODE [emv_exchangeCerts](#) (ref byte[] cert, ref byte[] nonce, ref byte[] signature, string ident="")
- RETURN_CODE [emv_callbackResponseKSN](#) (byte[] KSN, string ident="")
- RETURN_CODE [emv_verifyDUKPTLoaded](#) (byte[] KCV, string ident="")
- RETURN_CODE [emv_callbackResponseMSR](#) (byte[] MSR, string ident="")
- RETURN_CODE [emv_getEMVKernelCheckValue](#) (ref string response, string ident="")
- RETURN_CODE [emv_getEMVConfigurationCheckValue](#) (ref string response, string ident="")
- RETURN_CODE [emv_getEMVKernelVersion](#) (ref string response, string ident="")
- RETURN_CODE [emv_retrieveTransactionResult](#) (byte[] tags, ref IDTTransactionData tlv, string ident="")
- RETURN_CODE [emv_removeApplicationData](#) (byte[] AID, string ident="")
- RETURN_CODE [ctls_removeApplicationData](#) (byte[] AID, string ident="")
- RETURN_CODE [emv_removeAllApplicationData](#) (string ident="")
- RETURN_CODE [emv_removeCAPK](#) (byte[] capk, string ident="")
- RETURN_CODE [ctls_removeCAPK](#) (byte[] capk, string ident="")
- RETURN_CODE [emv_removeAllCAPK](#) (string ident="")
- RETURN_CODE [config_getKeySlotInfo](#) (int index, int slot, ref byte[] keyslot, string ident="")
- RETURN_CODE [adf_ApplicationControl](#) (ADF_APP_CONTROL state, string ident="")
- RETURN_CODE [adf_getModuleBytes](#) (ADF_TYPE type, ref List< byte[] > adfInfo, string ident="")
- RETURN_CODE [adf_getModuleInfo](#) (ADF_TYPE type, ref List< ADF_Info > adfInfo, string ident="")
- RETURN_CODE [adf_setJTAG](#) (bool enable, string ident="")
- RETURN_CODE [adf_setADFMode](#) (bool enable, string ident="")
- RETURN_CODE [adf_getADFMode](#) (ref bool enable, string ident="")
- RETURN_CODE [adf_eraseFlash](#) (ADF_TYPE type, string ident="")
- RETURN_CODE [ctls_removeAllCAPK](#) (string ident="")
- RETURN_CODE [emv_removeCRL](#) (byte[] crlList, string ident="")
- RETURN_CODE [emv_removeAllCRL](#) (string ident="")
- RETURN_CODE [emv_retrieveApplicationData](#) (byte[] AID, ref byte[] tlv, string ident="")
- RETURN_CODE [ctls_retrieveApplicationData](#) (byte[] AID, ref byte[] tlv, string ident="")
- RETURN_CODE [emv_retrieveAIDList](#) (ref byte[][] response, string ident="")
- RETURN_CODE [ctls_retrieveAIDList](#) (ref byte[][] response, string ident="")
- RETURN_CODE [emv_retrieveCAPK](#) (byte[] capk, ref byte[] key, string ident="")
- RETURN_CODE [ctls_retrieveCAPK](#) (byte[] capk, ref byte[] key, string ident="")
- RETURN_CODE [emv_retrieveCAPKList](#) (ref byte[] keys, string ident="")
- RETURN_CODE [ctls_retrieveCAPKList](#) (ref byte[] keys, string ident="")
- RETURN_CODE [emv_retrieveCRLList](#) (ref byte[] list, string ident="")
- RETURN_CODE [ctls_retrieveTerminalData](#) (ref byte[] tlv, string ident="")
- RETURN_CODE [device_retrieveTerminalData](#) (ref byte[] tlv, string ident="")
- RETURN_CODE [emv_retrieveTerminalData](#) (ref byte[] tlv, string ident="")
- RETURN_CODE [emv_removeTerminalData](#) (string ident="")
- RETURN_CODE [emv_setApplicationData](#) (byte[] name, byte[] tlv, string ident="")
- RETURN_CODE [emv_setApplicationData](#) (byte[] tlv, string ident="")
- RETURN_CODE [ctls_setApplicationData](#) (byte[] tlv, string ident="")
- RETURN_CODE [ctls_setDefaultConfiguration](#) (string ident="")

- RETURN_CODE [ctls_setConfigurationGroup](#) (byte[] tlv, string ident="")
- RETURN_CODE [emv_setCAPK](#) (byte[] key, string ident="")
- RETURN_CODE [ctls_setCAPK](#) (byte[] key, string ident="")
- RETURN_CODE [emv_setCRL](#) (byte[] list, string ident="")
- RETURN_CODE [emv_trySetTerminalData](#) (byte[] tlv, ref byte[] rejectedTLV, ref byte[] convertedTLV, bool overwrite=false, string ident="")
- RETURN_CODE [emv_setTerminalData](#) (byte[] tlv, string ident="")
- RETURN_CODE [emv_addTerminalData](#) (byte[] tlv, string ident="")
- RETURN_CODE [ctls_setTerminalData](#) (byte[] tlv, string ident="")
- RETURN_CODE [ctls_trySetTerminalData](#) (byte[] tlv, ref byte[] rejectedTLV, ref byte[] convertedTLV, string ident="")
- RETURN_CODE [device_setTerminalData](#) (byte[] tlv, string ident="")
- RETURN_CODE [device_sendVivoCommandP3](#) (byte command, byte subCommand, byte[] data, ref byte[] response, string ident="")
- RETURN_CODE [device_sendVivoCommandP3_ext](#) (byte command, byte subCommand, byte[] data, ref byte[] response, int timeout, bool noResponse, string ident="")
- RETURN_CODE [device_sendVivoCommandP4](#) (byte command, byte subCommand, byte[] data, ref byte[] response, string ident="")
- RETURN_CODE [device_sendVivoCommandP4_ext](#) (byte command, byte subCommand, byte[] data, ref byte[] response, int timeout, bool noResponse, string ident="")
- RETURN_CODE [emv_setTerminalMajorConfiguration](#) (int configuration, string ident="")
- RETURN_CODE [emv_getTerminalMajorConfiguration](#) (ref int configuration, string ident="")
- RETURN_CODE [device_lowPowerMode](#) (bool stopMode, bool wakeOnTrans, string ident="")
- RETURN_CODE [device_pingDevice](#) (string ident="")
- RETURN_CODE [device_buzzer](#) (string ident="")
- RETURN_CODE [device_cancelTransaction](#) (string ident="")
- RETURN_CODE [device_resetTransaction](#) (string ident="")
- RETURN_CODE [device_setLED](#) (byte indexLED, byte control, string ident="")
- RETURN_CODE [device_getProductType](#) (ref byte[] type, string ident="")
- RETURN_CODE [device_getProcessorType](#) (ref byte[] type, string ident="")
- RETURN_CODE [device_getHardwareInfor](#) (ref string ascii, string ident="")
- RETURN_CODE [device_getUIDofMCU](#) (ref string uid, string ident="")
- RETURN_CODE [device_getModuleVer](#) (ref string moduleVer, string ident="")
- RETURN_CODE [device_getUsbBootLoader](#) (ref string bootLoader, string ident="")
- RETURN_CODE [device_getRemoteKeyInjectionTO](#) (ref int timeout, string ident="")
- RETURN_CODE [device_getCashTranRiskPara](#) (ref byte[] tlv, string ident="")
- RETURN_CODE [device_getMsrSecurePar](#) (bool b0, bool b1, bool b2, bool b3, ref byte[] tlv, string ident="")
- RETURN_CODE [emv_resetConfigurationGroup](#) (int group, string ident="")
- RETURN_CODE [device_getMerchantRecord](#) (int index, ref byte[] record, string ident="")
- RETURN_CODE [device_resetConfigurationGroup](#) (int group, string ident="")
- RETURN_CODE [ctls_resetConfigurationGroup](#) (int group, string ident="")
- RETURN_CODE [device_controlUserInterface](#) (byte[] values, string ident="")
- RETURN_CODE [device_controlLED](#) (byte indexLED, byte control, string ident="")
- RETURN_CODE [device_sendVivoCommandP2](#) (byte command, byte subCommand, byte[] data, ref byte[] response, string ident="")
- RETURN_CODE [device_sendVivoCommandP2_ext](#) (byte command, byte subCommand, byte[] data, ref byte[] response, int timeout, bool noResponse, string ident="")
- RETURN_CODE [device_StartRKI](#) (int type, string ident="")
- RETURN_CODE [ctls_getConfigurationGroup](#) (int group, ref byte[] tlv, string ident="")
- RETURN_CODE [ctls_removeConfigurationGroup](#) (int group, string ident="")
- RETURN_CODE [ctls_getAllConfigurationGroups](#) (ref byte[][] response, string ident="")
- RETURN_CODE [device_setBurstMode](#) (byte mode, string ident="")
- RETURN_CODE [device_setPollMode](#) (byte mode, string ident="")
- RETURN_CODE [device_setMerchantRecord](#) (int index, bool enabled, string merchantID, string merchantURL, string ident="")

- RETURN_CODE [device_startRKI](#) (bool isTest, string ident="")
- RETURN_CODE [device_certificateType](#) (ref int type, string ident="")
- RETURN_CODE [device_updateDeviceFromManifest](#) (string filepath, string ident="", bool isForeground=false)
- RETURN_CODE [device_updateFirmwareType](#) (string path, FIRMWARE_TYPE type, FirmwareUpdateCallback callback, string ident="", UInt32 address=0)
- RETURN_CODE [device_updateFirmwareKernels](#) (string path, FirmwareUpdateCallback callback, string ip="", UInt32 address=0, string ident="")
- RETURN_CODE [device_updateFirmwareIP](#) (string path, int type, FirmwareUpdateCallback callback, string ip, string ident="", bool performOnForeground=false)
- RETURN_CODE [icc_getICCReaderStatus](#) (ref byte status, string ident="")
- RETURN_CODE [icc_powerOffICC](#) (string ident="")
- RETURN_CODE [icc_powerOnICC](#) (ref byte[] ATR, byte interfaces, string ident="")
- RETURN_CODE [icc_exchangeAPDU](#) (string c_APDU, ref byte[] response, string ident="")
- RETURN_CODE [lcd_displayMessage](#) (int lineNumber, string message, string ident="")
- RETURN_CODE [lcd_clearAllLines](#) (string ident="")
- RETURN_CODE [lcd_linkUIWithTransactionMessageId](#) (byte messageId, string screenName, string ident="")
- RETURN_CODE [lcd_showScreen](#) (string screenName, string ident="")
- RETURN_CODE [lcd_createScreen](#) (string screenName, ref UInt16 screenID, string ident="")
- RETURN_CODE [lcd_cloneScreen](#) (string screenName, string cloneName, ref UInt16 cloneID, string ident="")
- RETURN_CODE [lcd_destroyScreen](#) (string screenName, string ident="")
- RETURN_CODE [lcd_getActiveScreen](#) (ref string screenName, string ident="")
- RETURN_CODE [lcd_getButtonEvent](#) (ref UInt16 screenID, ref UInt16 objectID, ref string screenName, ref string objectName, ref bool isLongPress, string ident="")
- RETURN_CODE [lcd_addButton](#) (string screenName, string buttonName, byte type, byte alignment, UInt16 xCord, UInt16 yCord, string label, ref lcdItem returnItem, buttonCallback callback, string ident="")
- RETURN_CODE [lcd_addEthernet](#) (string screenName, string objectName, byte alignment, UInt16 xCord, UInt16 yCord, ref lcdItem returnItem, string ip, string ident="")
- RETURN_CODE [lcd_addLED](#) (string screenName, string objectName, byte alignment, UInt16 xCord, UInt16 yCord, ref lcdItem returnItem, byte[] LED, string ident="")
- RETURN_CODE [lcd_addImage](#) (string screenName, string objectName, byte alignment, UInt16 xCord, UInt16 yCord, string filename, ref lcdItem returnItem, string ident="")
- void [lcd_setButtonCallback](#) (string screenName, string buttonName, buttonCallback callback, string ip, string ident="")
- void [lcd_setPinInputCallback](#) (SwipeCallback callback, string ident="")
- RETURN_CODE [device_enterStandbyMode](#) (string ident="")
- void [lcd_setPinSwipeCallback](#) (SwipeCallback callback, string ident="")
- void [lcd_setPinFailureCallback](#) (FailureCallback callback, string ident="")
- void [lcd_setPinTimeoutCallback](#) (TimeoutCallback callback, string ident="")
- void [lcd_setPinCancelPromptCallback](#) (CancelPromptCallback callback, string ident="")
- RETURN_CODE [lcd_addText](#) (string screenName, string objectName, byte alignment, UInt16 xCord, UInt16 yCord, UInt16 width, UInt16 height, byte fontID, byte[] color, string label, ref lcdItem returnItem, string ident="")
- RETURN_CODE [lcd_updateLabel](#) (string screenName, string objectName, string label, string ident="")
- RETURN_CODE [lcd_updateColor](#) (string screenName, string objectName, byte[] color, string ident="")
- RETURN_CODE [lcd_updatePosition](#) (string screenName, string objectName, byte alignment, UInt16 new_xCord, UInt16 new_yCord, string ident="")
- RETURN_CODE [lcd_removeItem](#) (string screenName, string objectName, string ident="")
- RETURN_CODE [lcd_storeScreenInfo](#) (string ident="")
- RETURN_CODE [lcd_loadScreenInfo](#) (string ident="")
- RETURN_CODE [lcd_clearScreenInfo](#) (string ident="")
- RETURN_CODE [lcd_getAllScreens](#) (ref byte screenNumbers, ref Dictionary< UInt16, string > returnScreens, string ident="")
- RETURN_CODE [lcd_getAllObjects](#) (string screenName, ref byte objectNumbers, ref Dictionary< UInt16, string > returnObjects, string ident="")
- RETURN_CODE [lcd_queryScreenbyName](#) (string screenName, ref byte result, string ident="")

- RETURN_CODE [lcd_queryObjectbyName](#) (string objectName, ref byte objectNumbers, ref List< string > returnItems, string ident="")
- RETURN_CODE [lcd_queryScreenbyID](#) (UInt16 screenID, ref byte result, ref string screenName, string ident="")
- RETURN_CODE [lcd_queryObjectbyID](#) (UInt16 objectID, ref byte objectNumbers, ref List< string > returnItems, string ident="")
- RETURN_CODE [lcd_setBacklight](#) (bool isBacklightOn, byte backlightVal, string ident="")
- RETURN_CODE [device_enablePassThrough](#) (bool enablePassThrough, string ident="")
- RETURN_CODE [device_enableL100PassThrough](#) (bool enablePassThrough, string ident="")
- RETURN_CODE [device_enableL80PassThrough](#) (bool enablePassThrough, string ident="")
- string [device_getResponseCodeString](#) (RETURN_CODE eCode)
- RETURN_CODE [device_getTransactionResults](#) (ref IDTTransactionData results, string ident="")
- RETURN_CODE [pin_getFunctionKey](#) (int timeout, string ident="")
- RETURN_CODE [pin_getFunctionKey_ext](#) (int timeout, string ip, SwipeCallback inputCallback, FailureCallback failureCallback, TimeoutCallback timeoutCallback, CancelPromptCallback cancelPromptCallback)
- RETURN_CODE [pin_capturePin](#) (int timeout, int type, string PAN, int minPIN, int maxPIN, string message, string ident="")
- RETURN_CODE [pin_capturePinExt](#) (int timeout, int type, string PAN, int minPIN, int maxPIN, string message1, string message2, string verify1, string verify2, string ident="")
- RETURN_CODE [pin_capturePin_ext](#) (int timeout, int type, string PAN, int minPIN, int maxPIN, string message, string ident, SwipeCallback inputCallback, FailureCallback failureCallback, TimeoutCallback timeoutCallback, CancelPromptCallback cancelPromptCallback)
- RETURN_CODE [pin_sendBeep](#) (string ident="")
- RETURN_CODE [pin_getPanEntry](#) (bool csc, bool expDate, bool ADR, bool ZIP, bool mod10, UInt16 timeout, bool encPANOnly=false, string ident="")
- RETURN_CODE [pin_getPanEntry_ext](#) (bool csc, bool expDate, bool ADR, bool ZIP, bool mod10, UInt16 timeout, bool encPANOnly, string ip, SwipeCallback swipeCallback, TimeoutCallback timeoutCallback, CancelPromptCallback cancelPromptCallback)
- RETURN_CODE [pin_promptForInput](#) (int messageID, short timeout, string ip=null, string ident="")
- RETURN_CODE [pin_promptForInput_ext](#) (int messageID, short timeout, string ident, SwipeCallback inputCallback, SwipeCallback swipeCallback, FailureCallback failureCallback, TimeoutCallback timeoutCallback, CancelPromptCallback cancelPromptCallback)
- RETURN_CODE [pin_promptForNumericKeyWithSwipe](#) (short timeout, byte function, int minLen, int maxLen, string line1, string line2, byte[] signature, string ident="")
- RETURN_CODE [pin_promptForNumericKeyWithSwipe_ext](#) (short timeout, byte function, int minLen, int maxLen, string line1, string line2, byte[] signature, string ident, SwipeCallback inputCallback, SwipeCallback swipeCallback, FailureCallback failureCallback, TimeoutCallback timeoutCallback, CancelPromptCallback cancelPromptCallback)
- RETURN_CODE [pin_promptForAmount](#) (int timeout, int minLen, int maxLen, string message, byte[] signature, string ident="")
- RETURN_CODE [pin_promptForAmount_ext](#) (int timeout, int minLen, int maxLen, string message, byte[] signature, string ident, SwipeCallback inputCallback, FailureCallback failureCallback, TimeoutCallback timeoutCallback, CancelPromptCallback cancelPromptCallback)
- RETURN_CODE [pin_cancelPINEntry](#) (string ident="")
- void [device_listenForNotifications](#) (bool enable, string ident="")
- RETURN_CODE [device_deleteDirectory](#) (string filename, string ident="")
- RETURN_CODE [device_listDirectory](#) (string directoryName, bool recursive, bool onSD, ref string directory, string ident="")
- RETURN_CODE [device_transferFile](#) (string fileName, byte[] file, bool isSD=false, string ident="")
- RETURN_CODE [device_deleteFile](#) (string filename, bool isSD=false, string ident="")
- RETURN_CODE [felica_authentication](#) (byte[] key, string ident="")
- RETURN_CODE [felica_SendCommand](#) (byte[] command, ref byte[] response, string ident="")
- RETURN_CODE [felica_readWithMac](#) (int numBlocks, byte[] blockList, ref byte[] blocks, string ident="")
- RETURN_CODE [felica_writeWithMac](#) (int blockNumber, byte[] data, string ident="")
- RETURN_CODE [felica_read](#) (byte[] serviceCode, int numBlocks, byte[] blockList, ref byte[] blocks, string ident="")

- RETURN_CODE [felica_write](#) (byte[] serviceCode, int blockCount, byte[] blockList, byte[] data, ref byte[] statusFlag, string ident="")
- RETURN_CODE [ctls_nfcCommand](#) (byte[] nfcCmdPkt, ref byte[] response, string ident="")
- RETURN_CODE [felica_requestService](#) (byte[] nodeCode, ref byte[] response, string ident="")
- RETURN_CODE [lcd_startScreenSaver](#) (string name, string ident="")
- RETURN_CODE [lcd_playAudio](#) (string name, int type, string ident="")
- RETURN_CODE [lcd_stopAudio](#) (string ident="")
- RETURN_CODE [lcd_getAudioVolume](#) (ref int volume, string ident="")
- RETURN_CODE [lcd_setAudioVolume](#) (int volume, string ident="")
- RETURN_CODE [device_playAudio](#) (string name, int type, string ident="")
- RETURN_CODE [device_stopAudio](#) (string ident="")
- RETURN_CODE [device_getAudioVolume](#) (ref int volume, string ident="")
- RETURN_CODE [device_setAudioVolume](#) (int volume, string ident="")
- RETURN_CODE [lcd_startScanQR](#) (ushort timeout, string ident="")
- RETURN_CODE [lcd_startScanQR_ext](#) (ushort timeout, ushort xcord, ushort ycord, ushort width, ushort height, string ident="")
- RETURN_CODE [lcd_stopScanQR](#) (string ident="")
- RETURN_CODE [lcd_startCameraCapture](#) (ushort timeout, string ident="")
- RETURN_CODE [lcd_stopCameraCapture](#) (string ident="")
- RETURN_CODE [config_setEncryptionControl](#) (bool msr, bool icc, string ident="")
- RETURN_CODE [config_getEncryptionControl](#) (ref bool msr, ref bool icc, string ident="")
- RETURN_CODE [msr_setConfiguration](#) (byte[] config, string ident="")
- RETURN_CODE [msr_getConfiguration](#) (ref byte[] config, string ident="")
- RETURN_CODE [device_getRKIStatus](#) (bool isProd, bool isMultiKey, ref string status, ref Dictionary< string, RKI_KEY_TYPE > keys, string ident="")
- RETURN_CODE [device_getAnyRKIStatus](#) (bool isMultiKey, ref string status, ref Dictionary< string, RKI_KEY_TYPE > keys, string ident="")
- RETURN_CODE [emv_getEMVKernelVersionExt](#) (ref string response, string ident="")
- RETURN_CODE [device_updateFirmwareFromZip](#) (byte[] zipfile, string ident="", bool isForeground=false)
- RETURN_CODE [device_sendConfigurationFromZip](#) (byte[] zip, string filename, VC_OPERATION_TYPE type, bool matchFW, string ident="", bool isForeground=false)
- RETURN_CODE [device_sendConfiguration](#) (string filename, VC_OPERATION_TYPE type, bool matchFW, string ident="", bool isForeground=false)
- RETURN_CODE [device_readConfigurationToMemory](#) (string memo, Configs.InstallRules rules, List< Configs.DeviceCommand > cmds, string ident="", bool isForeground=false)
- RETURN_CODE [device_getConfigurationFromMemory](#) (ref string json, string ident="")
- RETURN_CODE [device_RemoteKeyInjection](#) (RKI_KEY_TYPE type, string keyName, string ident="", bool performOnForeground=false)
- RETURN_CODE [device_rrcUninstallApp](#) (string appName, string ident="")
- RETURN_CODE [device_rrcRunApp](#) (string appName, string ident="")
- RETURN_CODE [device_rrcInstallApp](#) (string appName, string ident="")
- RETURN_CODE [device_rrcDownloadApp](#) (string appName, byte[] appData, string ident="")
- RETURN_CODE [device_rrcConnect](#) (string ident="")
- RETURN_CODE [device_rrcDisconnect](#) (string ident="")

Static Public Member Functions

- static bool [useSerialPort](#) (int port)
- static bool [useSerialPort](#) (int port, int baud)
- static bool [useSerialPortLinux](#) (string path)
- static bool [useSerialPortLinux](#) (string path, int baud)
- static int [getCommandTimeout](#) (string ident="")
- static void [setCommandTimeout](#) (int milliseconds, string ident="")
- static void [hasUI](#) (bool val)

- static void [monitorNetworkForDevices](#) (bool monitorON, int port=0, bool authClient=false, int sendTimeout=6000, int receiveTimeout=6000)
- static bool [ip_switchToSocket](#) (string IP)
- static bool [closeSocket](#) (string IP)
- static List< string > [ip_getSocketList](#) ()
- static string [getlastErrorString](#) (string ident="")
- static bool [ip_isConnected](#) (string ip, int attempts=1, bool isSecure=false)
- static void [initSC](#) ()
- static void [setCallback](#) (CallBack my_Callback)
- static void [setLongPressCallback](#) (longPressCallback callback, string ident="")
- static void [setCallbackIP](#) (CallBackIP my_Callback, string ident="")
- static void [setCallback](#) (IntPtr my_Callback, SynchronizationContext context)
- static String [SDK_Version](#) ()
- static void [emv_autoAuthenticate](#) (bool authenticate, string ident="")
- static void [emv_autoAuthenticateTags](#) (bool authenticate, byte[] tags, string ident="")
- static void [emv_allowFallback](#) (bool allow, string ident="")
- static RETURN_CODE [device_updateFirmwareType](#) (FIRMWARE_TYPE type, byte[] firmwareData, string ident="", UInt32 address=0, bool performOnForeground=false, int manifestScriptsCount=0, int processedScriptCount=0, string DT_PRJ_filename="")
- static RETURN_CODE [device_updateDeviceFirmware](#) (byte[] firmwareData, string ident="")
- static RETURN_CODE [device_wakeDevice](#) (string macAddress="", string ipAddress="")
- static RETURN_CODE [lcd_clearDisplay](#) (string ident="")
- static void [lcd_retrieveMessage](#) (DisplayMessages.DISPLAY_MESSAGE_LANGUAGE lang, DisplayMessages.DISPLAY_MESSAGE_IDENTIFIER id, ref string line1, ref string line2)

Properties

- static bool [bypassUSBCheck](#) [get, set]
- static [IDT_NEO2 SharedController](#) [get]

21.10.1 Detailed Description

Class for VP6300 reader

21.10.2 Member Function Documentation

21.10.2.1 RETURN_CODE IDTechSDK.IDT_NEO2.adf_ApplicationControl (ADF_APP_CONTROL state, string ident = " ")

ADF Application Control

Starts or stop application running in ADF environment

Parameters

<i>state</i>	Application Control state: <ul style="list-style-type: none"> • ADF_APP_CONTROL_STOP_ALL = Stop All Applications • ADF_APP_CONTROL_START_ALL = Start All Applications • ADF_APP_CONTROL_START_FIRST = Start Only First Application • ADF_APP_CONTROL_START_ADDITIONAL = Start All Applicatins EXCEPT First Application
--------------	--

Returns

RETURN_CODE: Values can be parsed with `errorCode.getErrorString(string ident = "")`

21.10.2.2 RETURN_CODE IDTechSDK.IDT_NEO2.adf_eraseFlash (ADF_TYPE *type*, string *ident* = " ")

Erase ADF Flash

Erases ADF Flash Memory

Parameters

<i>type</i>	TRUE = ADF type, ADF_TYPE_SDK or ADF_TYPE_APP
<i>ident</i>	optional - device identifier

Returns

RETURN_CODE: Values can be parsed with `errorCode.getErrorString(string ident = "")`

21.10.2.3 RETURN_CODE IDTechSDK.IDT_NEO2.adf_getADFMode (ref bool *enable*, string *ident* = " ")

Get ADF Mode

Gets the enabled state of the ADF environment

Parameters

<i>enable</i>	TRUE = ADF enabled, FALSE = ADF disabled
---------------	--

Returns

RETURN_CODE: Values can be parsed with `errorCode.getErrorString(string ident = "")`

21.10.2.4 RETURN_CODE IDTechSDK.IDT_NEO2.adf_getModuleBytes (ADF_TYPE *type*, ref List< byte[]> *adfInfo*, string *ident* = " ")

Get Module Bytes

Retrieves first 64 bytes of module info running in ADF environment

Parameters

<i>type</i>	Type: <ul style="list-style-type: none"> • ADF_TYPE_SDK = SDK • ADF_TYPE_APP = Application
<i>adfInfo</i>	List of Modules First 64 bytes

Returns

RETURN_CODE: Values can be parsed with `errorCode.getErrorString(string ident = "")`

21.10.2.5 **RETURN_CODE** IDTechSDK.IDT_NEO2.adf_getModuleInfo (**ADF_TYPE** *type*, ref List< **ADF_Info** > *adfInfo*, string *ident* = " ")

Get Module Info Info

Retrieves module info running in ADF environment

Parameters

<i>type</i>	Type: <ul style="list-style-type: none"> • ADF_TYPE_SDK = SDK • ADF_TYPE_APP = Application
<i>adfInfo</i>	List of Modules Info

Returns

RETURN_CODE: Values can be parsed with `errorCode.getErrorString(string ident = "")`

21.10.2.6 **RETURN_CODE** IDTechSDK.IDT_NEO2.adf_setADFMode (bool *enable*, string *ident* = " ")

Set ADF Mode

Enables/Disables the ADF environment

Parameters

<i>enable</i>	TRUE = ADF enabled, FALSE = ADF disabled
---------------	--

Returns

RETURN_CODE: Values can be parsed with `errorCode.getErrorString(string ident = "")`

21.10.2.7 **RETURN_CODE** IDTechSDK.IDT_NEO2.adf_setJTAG (bool *enable*, string *ident* = " ")

Set JTAG

Enables/Disables the JTAG pin in the ADF environment

Parameters

<i>enable</i>	TRUE = JTAG enabled, FALSE = JTAG disabled
---------------	--

Returns

RETURN_CODE: Values can be parsed with `errorCode.getErrorString(string ident = "")`

21.10.2.8 **static bool** IDTechSDK.IDT_NEO2.closeSocket (string *IP*) [static]

Close Socket

Instructs SDK to attempt to use close existing TCP/IP socket used for communication with [IDT_NEO2](#)

Parameters

<i>IP</i>	Valid established IP address of the existing device. Must match original IP string exactly. Example: Connect to device as 192.168.1.155:50#Device_1. You must use that whole string, not just 192.168.1.155, or 192.168.1.155:50.
-----------	---

Return values

<i>success</i>	TRUE = socket closed, FALSE = socket not found
----------------	--

21.10.2.9 RETURN_CODE IDTechSDK.IDT_NEO2.config_checkDUKPTKey (byte *keyindex*, ref byte[] *val*, string *ident* = " ")

Check DUKPT Key

This command checks whether a valid DUKPT key is stored at the specified slot and if a valid key is found then some basic information related to the type of key is returned. The actual Key data is never returned. This command can be used to check whether a key is already present before injecting a key in a KeyIndex to prevent overwriting an existing DUKPT key.

Parameters

<i>keyindex</i>	Data Encryption Key Index (usually 5)
<i>val</i>	<ul style="list-style-type: none"> • Byte 0 = Key State: 00h = Unused, 01h = Valid, 02h = End Of Life, FFh = Not Available • Byte 1-2 = Key Usage (ASCII). "D0" = Use to encrypt data • Byte 3 = Algorithm (ASCII). "T" = Triple DES • Byte 4 = Mode of Use (ASCII). "E" = Encryption Only • Byte 5-6 = Key Version (ASCII). "00" = key version not used
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString

21.10.2.10 RETURN_CODE IDTechSDK.IDT_NEO2.config_getDEKVariantType (ref byte *type*, string *ident* = " ")

Get Data Encryption Key Variant Type

Returns the Data Encryption Key Type

Parameters

<i>type</i>	<ul style="list-style-type: none"> • 0 = Data Variant • 1 = Pin Variant
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString

21.10.2.11 RETURN_CODE IDTechSDK.IDT_NEO2.config_getDUKPT_DEK_Attribution (byte *keyslot*, ref byte *mode*, ref byte *outputMode_workingKey*, ref byte *variant_keyUsage*, string *ident* = " ")

Get DUKPT DEK Attribution based on KeySlot

Command settings can only be changed for each key one time.

Parameters

<i>keyslot</i>	Key Slot
<i>mode</i>	<ul style="list-style-type: none"> • 0 = TDES • 1 = AES
<i>outputMode_workingKey</i>	Output Mode is when TDES, Working Key is when AES <ul style="list-style-type: none"> • 0 = TDES / TDES • 1 = AES-128 / TDES3 • 2 = TransArmor / AES-128
<i>variant_keyUsage</i>	Variant is when TDES, Key Usage is when AES <ul style="list-style-type: none"> • 0 = Data Key / Encrypt/Decrypt • 1 = PIN Key / Encrypt • 2 = N/A / Decrypt
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString

21.10.2.12 RETURN_CODE IDTechSDK.IDT_NEO2.config_getDUKPT_KSN (ref byte[] *ksn*, string *ident* = " ")

Get DUKPT Key Serial Number (KSN)

Host can use this command to retrieve the KSN of the DUKPT key.

Parameters

<i>ksn</i>	Key Serial Number
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString

21.10.2.13 RETURN_CODE IDTechSDK.IDT_NEO2.config_getDUKPTEncryptionType (ref byte *type*, string *ident* = " ")

Get DUKPT Key Encryption Type

Returns DUKPT Key Encryption Type.

Parameters

<i>type</i>	<ul style="list-style-type: none"> • 0 = TDES • 1 = AES • 2 = TransArmor
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString

21.10.2.14 RETURN_CODE IDTechSDK.IDT_NEO2.config_getEncryptionControl (ref bool *msr*, ref bool *icc*, string *ident* = " ")

Get Encryption Control

Get Encryption Control to switch status between MSR and ICC/EMV function. Following Encryption status supported:

- MSR ON, ICC/EMV ON,
- MSR ON, ICC/EMV OFF,
- MSR OFF, ICC/EMV OFF,

Parameters

<i>msr</i>	<ul style="list-style-type: none"> • true: enabled MSR with Encryption, • false: disabled MSR with Encryption,
<i>icc</i>	<ul style="list-style-type: none"> • true: enabled ICC with Encryption, • false: disabled ICC with Encryption,
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString

21.10.2.15 RETURN_CODE IDTechSDK.IDT_NEO2.config_getEthernetMACAddress (ref byte[] *address*, string *ident* = " ")

Get Device Ethernet MAC Address

Parameters

<i>address</i>	6-byte MAC Address
<i>ip</i>	Optional: The callback function will only be applicable to the provided IP.
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString

21.10.2.16 RETURN_CODE IDTechSDK.IDT_NEO2.config_getKeyslot_PEK_DEK (ref byte *keyslotPEK*, ref byte *keyslotDEK*, string *ident* = " ")

Get KeySlot for DUKPT PEK and DUKPT DEK

- Returns the KeySlot for PEK and DEK (when exists).

Parameters

<i>keyslotPEK</i>	PEK KeySlot (0-9). Value of 255 (0xff) = does not exist
<i>keyslotDEK</i>	DEK KeySlot (0-9). Value of 255 (0xff) = does not exist
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString

21.10.2.17 RETURN_CODE IDTechSDK.IDT_NEO2.config_getKeySlotInfo (int *index*, int *slot*, ref byte[] *keyslot*, string *ident* = " ")

Get Key Slot Info

This command checks whether a valid DUKPT key is stored at the specified index and slot and if a valid key is found then some basic information related to the type of key is returned

Parameters

<i>index</i>	Key index: 0x01 = PIN, 0x02 = Data, 0x05 = MAC, 0x14 = KEK, 0x0A = PCI Pairing Key
<i>slot</i>	Key Slot: Slot number of particular key Key Slot Data. If 0x00, no key at the slot, otherwise: <ul style="list-style-type: none"> • Byte 0 - Key State: 0x00 = Unused, 0x01 = Valid, 0x02 = End of Life, 0xFF = Not Available • Byte 1 - KSN Length. Value 10 = TDES PIN DUKPT Key, value 12 = AES PIN DUKPT KEY • Byte 2-12 or 2-14 - DUKPT Key KSN. If length 10, then TDES. If length 12 then AES
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString

21.10.2.18 **RETURN_CODE** IDTechSDK.IDT_NEO2.config_getMasking (ref byte *prePAN*, ref byte *postPAN*, ref byte *asciiMask*, ref byte *hexMask*, ref bool *maskExp*, string *ident* = " ")

Get Masking

Parameters

<i>prePAN</i>	the number of pre-PAN characters to display in the clear. Valid values 0-6. Default 4.
<i>postPAN</i>	the number of post-PAN characters to display in the clear. Valid values 0-6. Default 4.
<i>asciiMask</i>	Mask character for ASCII output masked data. Valid values 0x20-0x7E. Default 0x2A (*, string <i>ident</i> = "") param <i>hexMask</i> Mask character for compressed numeric masked data. Valid values are 0x0A = 0x0F. Default 0x0C param <i>maskExp</i> TRUE = mask expiration data, FALSE = display expiration date in the clear.
<i>ip</i>	Optional: The callback function will only be applicable to the provided IP.
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with `device_getResponseCodeString`

21.10.2.19 **RETURN_CODE** IDTechSDK.IDT_NEO2.config_getNetworkConfiguration (ref bool *isStatic*, ref string *address*, ref string *subnet*, ref string *gateway*, ref string *dns*, string *ident* = " ")

Get Device Network Configuration

Parameters

<i>isStatic</i>	TRUE = Static IP, FALSE = DHCP
<i>address</i>	Device IP Address as string. Example "192.168.1.15"
<i>subnet</i>	Device Subnet as string. Example "255.255.255.0"
<i>gateway</i>	Device Gateway address as a string. Example "8.8.8.8"
<i>dns</i>	Device DNS address as string. Example "192.168.1.22"
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with `device_getResponseCodeString`

21.10.2.20 **RETURN_CODE** IDTechSDK.IDT_NEO2.config_getSalt_KCV (ref byte[] *kcv*, string *ident* = " ")

Get Salt KCV

Host can use this command to retrieve the Salt KCV.

Parameters

<i>kcv</i>	KCV
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString

21.10.2.21 RETURN_CODE IDTechSDK.IDT_NEO2.config_getSerialNumber (ref string *response*, string *ident* = " ")

Polls device for Serial Number

Parameters

<i>response</i>	Returns Serial Number
<i>ip</i>	Optional: The callback function will only be applicable to the provided IP.
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString

21.10.2.22 RETURN_CODE IDTechSDK.IDT_NEO2.config_getSSLServerEthernet (ref string *address*, ref int *port*, string *ident* = " ")

Get SSL Sever Ethernet

Parameters

<i>address</i>	IP Address of TLS Server.
<i>port</i>	Port of server.
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString

21.10.2.23 RETURN_CODE IDTechSDK.IDT_NEO2.config_getSwipeandDone (ref byte *swipeVal*, ref byte *doneVal*, ref byte *delay*, string *ident* = " ")

Get Swipe Button and Done Button Configuration

Read the button configuration from the ViVOpay Vendi reader.

Parameters

<i>swipe</i>	Value of Swipe Button. The value set to 01h, the Swipe Card switch is enabled. If value is set to 00h the Swipe Card switch is disabled.
<i>done</i>	Value of Done Button. If value is set to 01h, the Done switch is enabled. If value is set to 00h, the DONE switch is disabled.
<i>delay</i>	The Delay is an unsigned delay value in seconds. This should probably not be set to values larger than 30 seconds
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString

21.10.2.24 RETURN_CODE IDTechSDK.IDT_NEO2.config_getTrackFormat (ref byte *option*, string *ident* = " ")

Get Track Format

Parameters

<i>option</i>	Format Options <ul style="list-style-type: none"> • 0 = No start/end sentinels, No LRC • 1 = Include start/end sentinels, No LRC • 2 = Include start/end sentinels, Include LRC
<i>ip</i>	Optional: The callback function will only be applicable to the provided IP.
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString

21.10.2.25 RETURN_CODE IDTechSDK.IDT_NEO2.config_getWhiteList (ref Dictionary< string, string > *data*, string *ident* = " ")

Get White List

Parameters

<i>data</i>	The white list data returned as Dictionary. Key = start bin range, Value = end bin range. When no range, Value = empty string.
<i>ip</i>	Optional: The callback function will only be applicable to the provided IP.
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString

21.10.2.26 RETURN_CODE IDTechSDK.IDT_NEO2.config_getWifiConfig (ref int *mode*, ref string *ssid*, ref string *password*, ref string *ip*, ref string *netMask*, ref string *gateway*, ref string *remoteIP*, ref string *remotePort*, string *ident* = " ")

Get Wifi Configuration

Return all the network configurations.

Parameters

<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.
--------------	--

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString

21.10.2.27 RETURN_CODE IDTechSDK.IDT_NEO2.config_getWirelessWorkMode (ref int *mode*, string *ident* = " ")

Get Wireless Work Mode

Return Wireless Work Mode.

Parameters

<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.
--------------	--

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString

21.10.2.28 RETURN_CODE IDTechSDK.IDT_NEO2.config_sendSSLRequestWiFi (string *address*, int *port*, string *ident* = " ")

Send SSL Request Over WiFi

Attempts to connect to TLS server over WiFi.

Parameters

<i>address</i>	TLS Server Address.
<i>port</i>	TLS Server port. IF 0, will use 1443.
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString

21.10.2.29 RETURN_CODE IDTechSDK.IDT_NEO2.config_setBaudRate (int *baud*, string *ident* = " ")

Set Baud Rate

Sets the buad rate for RS-232 communication.

Parameters

<i>baud</i>	<ul style="list-style-type: none"> • 4 = 9600 • 6 = 19200 • 7 = 38400 • 8 = 57600 • 9 = 115200
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString

21.10.2.30 RETURN_CODE IDTechSDK.IDT_NEO2.config_setBluetoothParameters (string *name*, string *oldPW*, string *newPW*, string *ident* = " ")

Set BluetoothParameters

Sets the name and password for the BLE module.

Sending null to all three parameters resets the default password to 123456

Parameters

<i>name</i>	Device name, 1-25 characters
<i>oldPW</i>	Old password, as a six character string, example "123456"
<i>newPW</i>	New password, as a six character string, example "654321"
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString

21.10.2.31 RETURN_CODE IDTechSDK.IDT_NEO2.config_setDEKVariantType (byte *type*, string *ident* = " ")

Set Data Encryption Key Variant Type

This command exists to specify the key variant type of Data encryption Key (Slot = 0), and MUST be used before the initial loading of the Data encryption Key into the device. The key variant type CANNOT be changed once the Data encryption Key is present. It must remain either Data Variant or PIN Variant.

Parameters

<i>type</i>	<ul style="list-style-type: none"> • 0 = Data Variant • 1 = Pin Variant
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString

21.10.2.32 RETURN_CODE IDTechSDK.IDT_NEO2.config_setDUKPT_DEK_Attribution_AES (byte *keyslot*, byte *workingKey*, byte *keyUsage*, string *ident* = " ")

Set DUKPT DEK Attribution based on KeySlot AES

Command settings can only be changed for each key one time.

Parameters

<i>keyslot</i>	Key Slot
----------------	----------

Parameters

<i>workingKey</i>	<ul style="list-style-type: none"> • 0 = TDES • 1 = TDES3 • 2 = AES-128
<i>keyUsage</i>	<ul style="list-style-type: none"> • 0 = Encrypt/Decrypt • 1 = Encrypt • 2 = Decrypt
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString

21.10.2.33 RETURN_CODE IDTechSDK.IDT_NEO2.config_setDUKPT_DEK_Attribution_TDES (byte *keyslot*, byte *outputMode*, byte *variant*, string *ident* = " ")

Set DUKPT DEK Attribution based on KeySlot TDES

Command settings can only be changed for each key one time. However switching the Output mode (TDES and TransArmor TDES) is always valid.

Parameters

<i>keyslot</i>	Key Slot
<i>outputMode</i>	<ul style="list-style-type: none"> • 0 = TDES • 1 = AES-128 • 2 = TransArmor TDES
<i>variant</i>	<ul style="list-style-type: none"> • 0 = Data Variant • 1 = Pin Variant
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString

21.10.2.34 RETURN_CODE IDTechSDK.IDT_NEO2.config_setDUKPTEncryptionType (byte *type*, string *ident* = " ")

Set DUKPT Key Encryption Type

This command exists to specify the encryption type of Data encryption Key(Slot = 0), and MUST be used before the

initial loading of the Data encryption Key into the device. The encryption type CANNOT be changed once the Data encryption Key is present. It must remain either TDES or AES.

Parameters

<i>type</i>	<ul style="list-style-type: none"> • 0 = TDES • 1 = AES • 2 = TransArmor
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString

21.10.2.35 RETURN_CODE IDTechSDK.IDT_NEO2.config_setEncryptionControl (bool *msr*, bool *icc*, string *ident* = " ")

Set Encryption Control

Set Encryption Control to switch status between MSR and ICC/EMV function. Following Encryption status supported:

- MSR ON, ICC/EMV ON,
- MSR ON, ICC/EMV OFF,
- MSR OFF, ICC/EMV OFF,

Parameters

<i>msr</i>	<ul style="list-style-type: none"> • true: enable MSR with Encryption, • false: disable MSR with Encryption,
<i>icc</i>	<ul style="list-style-type: none"> • true: enable ICC with Encryption, • false: disable ICC with Encryption,
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString

21.10.2.36 RETURN_CODE IDTechSDK.IDT_NEO2.config_setKeyslot_PEK_DEK (bool *isPEK*, byte *keyslot*, string *ident* = " ")

Set KeySlot for DUKPT PEK and DUKPT DEK

- Selects one of the available PEKs for use in subsequent PIN encryption operations.
- Selects one of the available DEKs for use in subsequent Data encryption operations.

Parameters

<i>isPEK</i>	TRUE = Set PEK, FALSE = Set DEK
<i>keyslot</i>	The KeySlot number specifies the key this command will select
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString

21.10.2.37 RETURN_CODE IDTechSDK.IDT_NEO2.config_setMasking (byte *prePAN*, byte *postPAN*, byte *asciiMask*, byte *hexMask*, bool *maskExp*, string *ident* = " ")

Set Masking

Parameters

<i>prePAN</i>	the number of pre-PAN characters to display in the clear. Valid values 0-6. Default 4.
<i>postPAN</i>	the number of post-PAN characters to display in the clear. Valid values 0-6. Default 4.
<i>asciiMask</i>	Mask character for ASCII output masked data. Valid values 0x20-0x7E. Default 0x2A (*, string <i>ident</i> = "") param <i>hexMask</i> Mask character for compressed numeric masked data. Valid values are 0x0A = 0x0F. Default 0x0C param <i>maskExp</i> TRUE = mask expiration data, FALSE = display expiration date in the clear.
<i>ip</i>	Optional: The callback function will only be applicable to the provided IP.
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString

21.10.2.38 RETURN_CODE IDTechSDK.IDT_NEO2.config_setNetworkConfiguration (bool *isStatic*, string *address*, string *subnet*, string *gateway*, string *dns*, string *ident* = " ")

Set Device Network Configuration

Parameters

<i>isStatic</i>	TRUE = Static IP, FALSE = DHCP
<i>address</i>	Device IP Address as string. Example "192.168.1.15"
<i>subnet</i>	Device Subnet as string. Example "255.255.255.0"
<i>gateway</i>	Device Gateway address as a string. Example "8.8.8.8"
<i>dns</i>	Device DNS address as string. Example "192.168.1.22"
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString

21.10.2.39 **RETURN_CODE** IDTechSDK.IDT_NEO2.config_setRKLEKeys (short *keyNumber*, byte[] *tr31*, byte[] *nonce*, byte[] *hmac*, ref byte[] *kv*, ref byte[] *nonce_device*, ref byte[] *hmac_device*, string *ident* = " ")

Set RKL Keys

- Sets the RKL Keys.

Parameters

<i>keyNumber</i>	Number of keys remaining to load
<i>tr31</i>	ASN.1 structure of encrypted key(s)
<i>nonce</i>	Nonce generated by RKMS to generate an HMAC used for auth.
<i>hmac</i>	HMAC-SHA256 generated from RKMS.
<i>kv</i>	ASN.1 Key Verification Structure returned from device.
<i>nonce_device</i>	Nonce generated by the device to generate a MAC used for auth.
<i>hmac_device</i>	HMAC-SHA256 generated from device terminal.
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString

21.10.2.40 **RETURN_CODE** IDTechSDK.IDT_NEO2.config_setSSLServerEthernet (string *address*, int *port*, string *ident* = " ")

Set SSL Sever Ethernet

Parameters

<i>address</i>	IP Address of TLS Server. Example "192.168.1.15"
<i>port</i>	Port of server. If 0, 1443 will be used
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString

21.10.2.41 **RETURN_CODE** IDTechSDK.IDT_NEO2.config_setSwipeandDone (byte *swipeVal*, byte *doneVal*, byte *delay*, string *ident* = " ")

Set Swipe Button and Done Button Configuration

Configure the buttons on the ViVOpay Vendi reader. Both the SWIPE and DONE buttons can be independently disabled with this command. This command also sets the TAP disable time for when the SWIPE button is pressed. When the SWIPE button is enabled, the contactless reader is turned off for the programmed delay time so that a false read does not occur when the user wishes to swipe a dual contactless/MagStripe card.

Parameters

<i>swipe</i>	Value of Swipe Button. The value set to 01h, the Swipe Card switch is enabled. If value is set to 00h the Swipe Card switch is disabled.
<i>done</i>	Value of Done Button. If value is set to 01h, the Done switch is enabled. If value is set to 00h, the DONE switch is disabled.

Parameters

<i>delay</i>	The Delay is an unsigned delay value in seconds. This should probably not be set to values larger than 30 seconds
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString

21.10.2.42 RETURN_CODE IDTechSDK.IDT_NEO2.config_setTrackFormat (byte *option*, string *ident* = " ")

Set Track Format

Parameters

<i>option</i>	Format Options <ul style="list-style-type: none"> • 0 = No start/end sentinels, No LRC • 1 = Include start/end sentinels, No LRC • 2 = Include start/end sentinels, Include LRC
<i>ip</i>	Optional: The callback function will only be applicable to the provided IP.
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString

21.10.2.43 RETURN_CODE IDTechSDK.IDT_NEO2.config_setWhiteList (byte[] *data*, string *ident* = " ")

Set White List

Parameters

<i>data</i>	The signed white list data
<i>ip</i>	Optional: The callback function will only be applicable to the provided IP.
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString

21.10.2.44 RETURN_CODE IDTechSDK.IDT_NEO2.config_setWifiConfig (int *mode*, string *ssid*, string *password*, string *ip*, string *netMask*, string *gateway*, string *ident* = " ", string *remoteIP* = null, string *remotePort* = null)

Set Wifi Configuration

Set wifi configuration. Configuration will be saved in flash.

Parameters

<i>mode</i>	WiFi work mode 0 – Set NULL Mode, WiFi RF will be disabled 1 - Set Station Mode Default) 2 – Set SoftAP Mode 3 – Set SoftAP+Station Mode (Default)
<i>ssid</i>	SSID of the target AP as string.
<i>password</i>	Password of the target AP as string.
<i>ip</i>	Device IP Address as string.
<i>netMask</i>	Device Subnet as string.
<i>gateway</i>	Device Gateway address as a string.
<i>remoteIP</i>	Optional. IP Address of TLS server as string.
<i>remotePort</i>	Optional. Port of TLS server as string. Default 1443.
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString

21.10.2.45 RETURN_CODE IDTechSDK.IDT_NEO2.config_setWirelessWorkMode (int *mode*, string *ident* = " ")

Set Wireless Work Mode

Set wireless work mode(Wi-Fi/BLE). Configuration will be saved in flash.

Parameters

<i>mode</i>	Wireless work mode 0 – Set Wi-Fi TCP Server Mode 1 - Set Wi-Fi SSL Server Mode 2 – Set BLE Server Mode 3 - Set Wi-Fi UDP Server Mode 5 – Disable Wi-Fi and BLE, power off wireless module
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString

21.10.2.46 bool IDTechSDK.IDT_NEO2.createFastEMVData (ref IDTTTransactionData *cData*, string *ident* = " ")

Create Fast EMV Data

At the completion of a Fast EMV Transaction, after the final card decision is returned and the IDTTTransactionData object is provided, sending that cData object to this method will populate the .fastEMV element with string data that represents the Fast EMV data that would be returned from and IDTech FastEMV over KB protocol

Parameters

<i>cData</i>	The IDTTTransactionData object populated with card data.
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with errorCode.getErrorString(string ident = "")

21.10.2.47 **RETURN_CODE** IDTechSDK.IDT_NEO2.ctls_activateTransaction (int *timeout*, byte[] *tags*, bool *forceOnline*, bool *isFastEMV* = false, string *ident* = " ")

Start a CTLS Transaction Request

Authorizes the CTLS transaction for an CTLS card

The tags will be returned in the callback routine.

Parameters

<i>timeout</i>	Timeout value in seconds.
<i>tags</i>	The tags to be included in the request. Passed as TLV Data. Example, tag 9F02 with amount 0x000000000100 would be 0x9F0206000000000100
<i>forceOnline</i>	TRUE = do not allow offline approval, FALSE = allow ICC to approve offline if terminal capable
<i>isFastEMV</i>	If TRUE, it will populate the IDTTransactionData.fastEMV with ASCII data similar to IDTech FastEMV KB output, after performing an auto-authenticate and auto-complete with ResultCode = Could Not Contact Host
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with errorCode.getErrorString()

NOTE ON APPLEPAY VAS: To enable ApplePay VAS, first a merchant record must be defined in one of the six available index positions (1-6) using device_setMerchantRecord, then container tag FFEE06 must be sent as part of the additional tags parameter of ctls_startTransaction. Tag FFEE06 must contain tag 9F26 and 9F22, and can optionally contain tags 9F2B and DFO1. Example FFEE06189F220201009F2604000000009F2B050100000000DF010101 9F22 = two bytes = ApplePay Terminal Application Version Number. Hard defined as 0100 for now. (required, string ident = "") 9F26 = four bytes = ApplePay Terminal Capabilities Information (required, string ident = "")

- Byte 1 = RFU
- Byte 2 = Terminal Type
- - Bit 8 = VAS Support (1=on, 0 = off, string ident = "")
- - Bit 7 = Touch ID Required (1=on, 0 = off, string ident = "")
- - Bit 6 = RFU
- - Bit 5 = RFU
- - Bit 1,2,3,4
- - - 0 = Payment Terminal
- - - 1 = Transit Terminal
- - - 2 = Access Terminal
- - - 3 = Wireless Handoff Terminal
- - - 4 = App Handoff Terminal
- - - 15 = Other Terminal
- Byte 3 = RFU
- Byte 4 = Terminal Mode
- - 0 = ApplePay VAS OR ApplePay

- - 1 = ApplePay VAS AND ApplePay
- - 2 = ApplePay VAS ONLY
- - 3 = ApplePay ONLY 9F2B = 5 bytes = ApplePay VAS Filter. Each byte filters for that specific merchant index (optional, string ident = "") DF01 = 1 byte = ApplePay VAS Protocol. (optional, string ident = "")
- - Bit 1 : 1 = URL VAS, 0 = Full VAS
- - Bit 2 : 1 = VAS Beeps, 0 = No VAS Beeps
- - Bit 3 : 1 = Silent Comm Error, 2 = EMEA Comm Error
- - Bit 4-8 : RFU

21.10.2.48 RETURN_CODE IDTechSDK.IDT_NEO2.ctls_cancelTransaction (string *ident* = " ")

Cancel Transaction

Cancels the currently executing EMV or CTLS transaction.

Parameters

<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.
--------------	--

Returns

RETURN_CODE: Values can be parsed with `errorCode.getErrorString()`

21.10.2.49 RETURN_CODE IDTechSDK.IDT_NEO2.ctls_getAllConfigurationGroups (ref byte *response*[], string *ident* = " ")

Retrieve All Configuration Groups

Returns all the Configuration Groups installed on the terminal for CTLS

Parameters

<i>response</i>	array of CTLS groups as TLV bytes
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with `errorCode.getErrorString(string ident = "")`

21.10.2.50 RETURN_CODE IDTechSDK.IDT_NEO2.ctls_getConfigurationGroup (int *group*, ref byte[] *tlv*, string *ident* = " ")

Get Configuration Group

Retrieves the Configuration for the specified Group. Group 0 = terminal settings.

Parameters

<i>group</i>	Configuration Group
<i>tlv</i>	return data
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with `errorCode.getErrorString()`

21.10.2.51 RETURN_CODE IDTechSDK.IDT_NEO2.ctls_nfcCommand (byte[] nfcCmdPkt, ref byte[] response, string ident = "")

NFC Command

This command uses `nfcCmdPkt[0]` in command data field to implement different functions. This command should be used in Pass-Through mode and command with "Poll for a NFC Tag" data should be used first. Command with other data can only be used once the "Poll for a NFC Tag" command has indicated that a NFC tag is present.

Parameters

<i>nfcCmdPkt</i>	<p>System Code</p> <ul style="list-style-type: none"> • Poll for NFC Tag: <code>nfcCmdPkt[0] = 0xff</code>, <code>nfcCmdPkt[1]</code> = timeout value (in seconds, string <code>ident = ""</code>) • Tag1 Static Get All Data: <code>nfcCmdPkt[0] = 0x11</code> • Tag1 Static Read a Byte: <code>nfcCmdPkt[0] = 0x12</code>, <code>nfcCmdPkt[1]</code> = Address of Data • Tag1 Static Write a Byte: <code>nfcCmdPkt[0] = 0x13</code> <code>nfcCmdPkt[1]</code> = Address of Data, <code>nfcCmdPkt[2]</code> = Data to be written • Tag1 Static Write a Byte NE: <code>nfcCmdPkt[0] = 0x14</code>, <code>nfcCmdPkt[1]</code> = Address of Data, <code>nfcCmdPkt[2]</code> = Data to be written • Tag1 Dynamic Read a Segment: <code>nfcCmdPkt[0] = 0x15</code>, <code>nfcCmdPkt[1]</code> = Address of Segment • Tag1 Dynamic Read 8 Bytes: <code>nfcCmdPkt[0] = 0x16</code>, <code>nfcCmdPkt[1]</code> = Address of Data • Tag1 Dynamic Write 8 Bytes: <code>nfcCmdPkt[0] = 0x17</code>, <code>nfcCmdPkt[1]</code> = Address of Data, <code>nfcCmdPkt[2]~nfcCmdPkt[9]</code> = Data to be written • Tag1 Dynamic Write 8 Bytes NE: <code>nfcCmdPkt[0] = 0x18</code>, <code>nfcCmdPkt[1]</code> = Address of Data, <code>nfcCmdPkt[2]~nfcCmdPkt[9]</code> = Data to be written • Tag2 Read Data (16 bytes): <code>nfcCmdPkt[0] = 0x21</code>, <code>nfcCmdPkt[1]</code> = Address of Data • Tag2 Write Data (4 bytes): <code>nfcCmdPkt[0] = 0x22</code>, <code>nfcCmdPkt[1]</code> = Address of Data, <code>nfcCmdPkt[2]~nfcCmdPkt[5]</code> = Data to be written • Tag2 Select Sect: <code>nfcCmdPkt[0] = 0x23</code>, <code>nfcCmdPkt[1]</code> = Sect number • Tag3 Read Data: – <code>nfcCmdPkt[0] = 0x41</code>, – <code>nfcCmdPkt[1]</code> = Number of services, value <code>n</code> – <code>nfcCmdPkt[2]~nfcCmdPkt[2n+1]</code>: Service code list – <code>nfcCmdPkt[2n+2]</code>: Number of blocks, value <code>m</code>. – <code>nfcCmdPkt[2n+3....]</code>: Block list, length is <code>2m~3m</code> • Tag3 Write Data: – <code>nfcCmdPkt[0] = 0x41</code>, – <code>nfcCmdPkt[1]</code> = Number of services, value <code>n</code> – <code>nfcCmdPkt[2]~nfcCmdPkt[2n+1]</code>: Service code list – <code>nfcCmdPkt[2n+2]</code>: Number of blocks, value <code>m</code>. – <code>nfcCmdPkt[2n+3....]</code>: Block list, length is <code>2m~3m</code> – <code>nfcCmdPkt[...]</code>: Block data, length is <code>16m</code> • Tag4 Command: <code>nfcCmdPkt[0] = 0x81</code>, <code>nfcCmdPkt[1]~nfcCmdPkt[n]</code>:data
<i>response</i>	Response as explained in FeliCA Lite-S User's Manual
<i>ip</i>	IP Address of target device (optional, string <code>ident = ""</code>)
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with `errorCode.getErrorString(string ident = "")`

21.10.2.52 RETURN_CODE IDTechSDK.IDT_NEO2.ctls_removeAllCAPK (string *ident* = " ")

Remove All Certificate Authority Public Key

Removes all the CAPK for CTLS

Parameters

<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.
--------------	--

Returns

RETURN_CODE: Values can be parsed with `errorCode.getErrorString(string ident = "")`

21.10.2.53 RETURN_CODE IDTechSDK.IDT_NEO2.ctls_removeApplicationData (byte[] *AID*, string *ident* = " ")

Remove Application Data by AID

Removes the Application Data for CTLS as specified by the AID name passed as a parameter

Parameters

<i>AID</i>	Name of ApplicationID Must be between 5 and 16 bytes
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with `errorCode.getErrorString(string ident = "")`

21.10.2.54 RETURN_CODE IDTechSDK.IDT_NEO2.ctls_removeCAPK (byte[] *capk*, string *ident* = " ")

Remove Certificate Authority Public Key

Removes the CAPK for CTLS as specified by the RID/Index

Parameters

<i>capk</i>	6 byte CAPK = 5 bytes RID + 1 byte INDEX
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with `errorCode.getErrorString(string ident = "")`

21.10.2.55 RETURN_CODE IDTechSDK.IDT_NEO2.ctls_removeConfigurationGroup (int *group*, string *ident* = " ")

Remove Configuration Group

Removes the Configuration as specified by the Group. Must not by group 0

Parameters

<i>group</i>	Configuration Group
--------------	---------------------

Return values

<i>RETURN_CODE</i>	Return codes listed as typedef enum in IDTCommon:RETURN_CODE. Values can be parsed with IDT_NEO2::device_getResponseCodeString :(string ident = "")
--------------------	--

21.10.2.56 **RETURN_CODE** IDTechSDK.IDT_NEO2.ctls_resetConfigurationGroup (int *group*, string *ident* = " ")

Reset Configuration Group

This command allows resetting a dataset to its default configuration. If the file exists, it will be overwritten. If not, it will be created.

Parameters

<i>group</i>	Configuration Group
--------------	---------------------

Return values

<i>RETURN_CODE</i>	Return codes listed as typedef enum in IDTCommon:RETURN_CODE. Values can be parsed with IDT_VP8800::device_getResponseCodeString :(string ident = "")
--------------------	--

21.10.2.57 **RETURN_CODE** IDTechSDK.IDT_NEO2.ctls_retrieveAIDList (ref byte *response*[[], string *ident* = " ")

Retrieve AID list

Returns all the AID names and their assigned groups installed on the terminal for CTLS.

Parameters

<i>response</i>	array of TLV data objects: DFEE2D (group name) followed by 9F06 (AID), and DFEE4F (Interface 01 = CTLS, 02 = CONTACT, string ident = "")
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with `errorCode.getErrorString`(string ident = "")

21.10.2.58 **RETURN_CODE** IDTechSDK.IDT_NEO2.ctls_retrieveApplicationData (byte[] *AID*, ref byte[] *tlv*, string *ident* = " ")

Retrieve Application Data by AID

Retrieves the CTLS Application Data as specified by the AID name passed as a parameter.

Parameters

<i>AID</i>	Name of ApplicationID. Must be between 5 and 16 bytes
<i>tlv</i>	The TLV elements of the requested AID
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with `errorCode.getErrorString()`

21.10.2.59 RETURN_CODE IDTechSDK.IDT_NEO2.ctls_retrieveCAPK (byte[] *capk*, ref byte[] *key*, string *ident* = " ")

Retrieve Certificate Authority Public Key

Retrieves the CAPK for CTLS as specified by the RID/Index passed as a parameter.

Parameters

<i>capk</i>	6 bytes CAPK = 5 bytes RID + 1 byte Index
<i>key</i>	Response returned as a CAKey format: [5 bytes RID][1 byte Index][1 byte Hash Algorithm][1 byte Encryption Algorithm][20 bytes HashValue][4 bytes Public Key Exponent][2 bytes Modulus Length][Variable bytes Modulus] Where: <ul style="list-style-type: none"> • Hash Algorithm: The only algorithm supported is SHA-1. The value is set to 0x01 • Encryption Algorithm: The encryption algorithm in which this key is used. Currently support only one type: RSA. The value is set to 0x01. • HashValue: Which is calculated using SHA-1 over the following fields: RID & Index & Modulus & Exponent • Public Key Exponent: Actually, the real length of the exponent is either one byte or 3 bytes. It can have two values: 3 (Format is 0x00 00 00 03), or 65537 (Format is 0x00 01 00 01, string <i>ident</i> = "") • Modulus Length: LenL LenH Indicated the length of the next field. • Modulus: This is the modulus field of the public key. Its length is specified in the field above.
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with `errorCode.getErrorString()`

21.10.2.60 RETURN_CODE IDTechSDK.IDT_NEO2.ctls_retrieveCAPKList (ref byte[] *keys*, string *ident* = " ")

Retrieve the Certificate Authority Public Key list

Returns all the CAPK RID and Index installed on the terminal for CTLS.

Parameters

<i>keys</i>	[key1][key2]...[keyn], each key 6 bytes where key = 5 bytes RID + 1 byte index
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with `errorCode.getErrorString()`

21.10.2.61 RETURN_CODE IDTechSDK.IDT_NEO2.ctls_retrieveTerminalData (ref byte[] *tlv*, string *ident* = " ")

Retrieve Terminal Data

Retrieves the Terminal Data for CTLS. This is configuration group 0 (Tag FFEE - > FFEE0100). The terminal data can also be retrieved by `ctls_getConfiguraitonGroup(0)`.

Parameters

<i>tlv</i>	Response returned as a TLV
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with `errorCode.getErrorString()`

21.10.2.62 RETURN_CODE IDTechSDK.IDT_NEO2.ctls_setApplicationData (byte[] tlv, string ident = " ")

Set Application Data by AID

Sets the Application Data as specified by TLV data

Parameters

<i>tlv</i>	Application data in TLV format The first tag of the TLV data must be the group number (DFEE2D). Group 0 = System, all other Groups = User The second tag of the TLV data must be the AID (9F06) If tag DFEE4F is included, it must have a value of 0x01 to be a CTLS AID
------------	--

Example valid TLV, for Group #2, AID a0000000045010: "dfee2d01029f0607a0000000045010dfee4b0101dfee2e0110dfee4c0101dfee4f0101"

Tags:

- DFEE2D : Group Number. Mandatory First Tag. 0 = System, 1-255 = User
- 9F06 : AID. Mandatory Second Tag
- DFEE4B : Partial AID Matching. 01 = allowed, 00 = Disabled. Mandatory for Visa
- DFEE4C : Application Flow, System: Never, User: Mandatory – 0x01 = MasterCard – 0x02 = AMEX – 0x03 = MasterCard w/Strip Application – 0x06 = Visa – 0x0D = Discover – 0x0E = JCB – 0x15 = Reserved – 0x16 = Reserved – 0x17 = Reserved
- DFEE4D = PPSE Disable, Optional
- DFEE2E = Max AID Length, Mandatory if DFEE4B included. Visa must be set to 16
- DFEE2F = Disable System Aid (no effect on User AID, string ident = "")
- DFEE4F = Interface Support. 01 = CTLS, 02 = Contact. If missing, defaults to CTLS

Parameters

<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.
--------------	--

Returns

RETURN_CODE: Values can be parsed with `errorCode.getErrorString()`

21.10.2.63 RETURN_CODE IDTechSDK.IDT_NEO2.ctls_setCAPK (byte[] key, string ident = " ")

Set Certificate Authority Public Key

Sets the CAPK for CTLS as specified by the CAKey structure

Parameters

<i>key</i>	CAKey format: [5 bytes RID][1 byte Index][1 byte Hash Algorithm][1 byte Encryption Algorithm][20 bytes HashValue][4 bytes Public Key Exponent][2 bytes Modulus Length][Variable bytes Modulus] Where: <ul style="list-style-type: none"> • Hash Algorithm: The only algorithm supported is SHA-1. The value is set to 0x01 • Encryption Algorithm: The encryption algorithm in which this key is used. Currently support only one type: RSA. The value is set to 0x01. • HashValue: Which is calculated using SHA-1 over the following fields: RID & Index & Modulus & Exponent • Public Key Exponent: Actually, the real length of the exponent is either one byte or 3 bytes. It can have two values: 3 (Format is 0x00 00 00 03), or 65537 (Format is 0x00 01 00 01, string ident = "") • Modulus Length: LenL LenH Indicated the length of the next field. • Modulus: This is the modulus field of the public key. Its length is specified in the field above.
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

ReturnsRETURN_CODE: Values can be parsed with `errorCode.getErrorString()`**21.10.2.64 RETURN_CODE IDTechSDK.IDT_NEO2.ctls_setConfigurationGroup (byte[] tlv, string ident = " ")**

Set Configuration Group

Sets the Configuration Group for CTLS as specified by the TLV data

Parameters

<i>tlv</i>	Configuration Group Data in TLV format The first tag of the TLV data must be the group number (DFEE2D). A second tag must exist
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

ReturnsRETURN_CODE: Values can be parsed with `errorCode.getErrorString()`**21.10.2.65 RETURN_CODE IDTechSDK.IDT_NEO2.ctls_setDefaultConfiguration (string ident = " ")**

Set Default Configuration Group

Resets the device to default CTLS configuration group settings

Parameters

<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.
--------------	--

Returns

RETURN_CODE: Values can be parsed with `errorCode.getErrorString()`

21.10.2.66 RETURN_CODE IDTechSDK.IDT_NEO2.ctls_setTerminalData (byte[] tlv, string ident = " ")

Set Terminal Data

Sets the Terminal Data for CTLS as specified by the TLV. The terminal global data is group 0. Other groups can be defined using this method (1 or greater), and those can be retrieved with `emv_getConfigurationGroup(int group)`, and deleted with `emv_removeConfigurationGroup(int group)`. You cannot delete group 0.

Parameters

<i>tlv</i>	TerminalData configuration data
------------	---------------------------------

Return values

<i>RETURN_CODE</i>	Return codes listed as typedef enum in IDTCommon:RETURN_CODE. Values can be parsed with IDT_NEO2::device_getResponseCodeString:(string ident = "")
--------------------	--

21.10.2.67 RETURN_CODE IDTechSDK.IDT_NEO2.ctls_startTransaction (double amount, double amtOther, int exponent, int type, int timeout, byte[] tags, bool forceOnline, bool isFastEMV = false, string ident = " ")

Start a CTLS Transaction Request

Authorizes the CTLS transaction for an CTLS card

The tags will be returned in the callback routine.

Parameters

<i>amount</i>	Transaction amount value (tag value 9F02, string ident = "")
<i>amtOther</i>	Other amount value, if any (tag value 9F03, string ident = "")
<i>exponent</i>	Number of characters after decimile point
<i>type</i>	Transaction type (tag value 9C).
<i>timeout</i>	Timeout value in seconds.
<i>tags</i>	Any other tags to be included in the request. Passed as TLV Data. Example, tag 9F02 with amount 0x000000000100 would be 0x9F0206000000000100 If tags 9F02 (amount), 9F03 (other amount), or 9C (transaction type) are included, they will take priority over these values supplied as individual parameters to this method.
<i>forceOnline</i>	TRUE = do not allow offline approval, FALSE = allow ICC to approve offline if terminal capable
<i>isFastEMV</i>	If TRUE, it will populate the IDTTransactionData.fastEMV with ASCII data similar to IDTech FastEMV KB output, after performing an auto-authenticate and auto-complete with ResultCode = Could Not Contact Host
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with `errorCode.getErrorString()`

NOTE ON APPLEPAY VAS: To enable ApplePay VAS, first a merchant record must be defined in one of the six available index positions (1-6) using `device_setMerchantRecord`, then container tag FFEE06 must be sent as part

of the additional tags parameter of `ctls_startTransaction`. Tag FFEE06 must contain tag 9F26 and 9F22, and can optionally contain tags 9F2B and DF01. Example FFEE06189F220201009F2604000000009F2B050100000000↵ DF010101 9F22 = two bytes = ApplePay Terminal Application Version Number. Hard defined as 0100 for now. (required, string ident = "") 9F26 = four bytes = ApplePay Terminal Capabilities Information (required, string ident = "")

- Byte 1 = RFU
- Byte 2 = Terminal Type
 - - Bit 8 = VAS Support (1=on, 0 = off, string ident = "")
 - - Bit 7 = Touch ID Required (1=on, 0 = off, string ident = "")
 - - Bit 6 = RFU
 - - Bit 5 = RFU
 - - Bit 1,2,3,4
 - - - 0 = Payment Terminal
 - - - 1 = Transit Terminal
 - - - 2 = Access Terminal
 - - - 3 = Wireless Handoff Terminal
 - - - 4 = App Handoff Terminal
 - - - 15 = Other Terminal
- Byte 3 = RFU
- Byte 4 = Terminal Mode
 - - 0 = ApplePay VAS OR ApplePay
 - - 1 = ApplePay VAS AND ApplePay
 - - 2 = ApplePay VAS ONLY
 - - 3 = ApplePay ONLY 9F2B = 5 bytes = ApplePay VAS Filter. Each byte filters for that specific merchant index (optional, string ident = "") DF01 = 1 byte = ApplePay VAS Protocol. (optional, string ident = "")
 - - Bit 1 : 1 = URL VAS, 0 = Full VAS
 - - Bit 2 : 1 = VAS Beeps, 0 = No VAS Beeps
 - - Bit 3 : 1 = Silent Comm Error, 2 = EMEA Comm Error
 - - Bit 4-8 : RFU

21.10.2.68 RETURN_CODE IDTechSDK.IDT_NEO2.`ctls_trySetTerminalData` (`byte[] tlv`, `ref byte[] rejectedTLV`, `ref byte[] convertedTLV`, `string ident = ""`)

Try to Set CTLS Terminal Data

Attempts to set the CTLS Terminal Data. Use this functions to attempt to set the terminal data to the device. This function will allow the cross-device tag compatible writing of IDTech Legacy/New tag definitions

Parameters

<i>tlv</i>	TerminalData TLV Data
<i>rejectedTLV</i>	Contains the tags (if any) that were rejected by the firmware
<i>convertedTLV</i>	Contains the tags (if any) that were converted and accepted by the firmware
<i>ident</i>	Optional identifier

Return values

<i>RETURN_CODE</i>	Return codes listed as typedef enum in IDTCommon:RETURN_CODE. Values can be parsed with IDT_BTPay::device_getResponseCodeString:()
--------------------	--

21.10.2.69 `RETURN_CODE IDTechSDK.IDT_NEO2.ctls_updateBalance (byte statusCode, byte[] authCode, byte[] date, byte[] time, string ident = " ")`

Update Balance

This command is the authorization response sent by the issuer to the terminal including the Authorization Status (OK or NOT OK).

This command is also being used in some implementations (i.e. EMEA) to communicate the results of Issuer Authentication to the reader in order to display the correct LCD messages. With this command, the POS passes the authorization result (OK/NOT OK), and possibly the Authorization Code (Auth_Code)/Date/Time to the terminal.

Parameters

<i>statusCode</i>	00: OK, 01: NOT OK, 02: (ARC response 89 for Interac, string ident = "")
<i>authCode</i>	Authorization code from host. Six bytes. Optional
<i>date</i>	Transaction date. If null, uses current terminal date. 3 bytes compressed numeric YYMMDD (tag value 9A).
<i>time</i>	Transaction time. If null, uses current terminal time. 3 bytes compressed numeric HHMMSS (tag value 9F21).
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString

21.10.2.70 `RETURN_CODE IDTechSDK.IDT_NEO2.device_activateTransaction (int timeout, byte[] tags, bool isFastEMV = false, string ident = " ")`

Start a Transaction Request

Authorizes the transaction CTLS, MSR or Contact EMV

The tags will be returned in the callback routine.

Parameters

<i>timeout</i>	Timeout value in seconds.
<i>tags</i>	The tags to be included in the request. Passed as TLV Data. Example, tag 9F02 with amount 0x000000000100 would be 0x9F0206000000000100
<i>isFastEMV</i>	If TRUE, it will populate the IDTTransactionData.fastEMV with ASCII data similar to IDTech FastEMV KB output, after performing an auto-authenticate and auto-complete with ResultCode = Could Not Contact Host
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with `errorCode.getErrorString()`

NOTE ON APPLEPAY VAS: To enable ApplePay VAS, first a merchant record must be defined in one of the six available index positions (1-6) using `device_setMerchantRecord`, then container tag FFEE06 must be sent as part of the additional tags parameter of `ctls_startTransaction`. Tag FFEE06 must contain tag 9F26 and 9F22, and can optionally contain tags 9F2B and DFO1. Example FFEE06189F220201009F2604000000009F2B050100000000↵ DF010101 9F22 = two bytes = ApplePay Terminal Application Version Number. Hard defined as 0100 for now. (required, string ident = "") 9F26 = four bytes = ApplePay Terminal Capabilities Information (required, string ident = "")

- Byte 1 = RFU
- Byte 2 = Terminal Type
 - - Bit 8 = VAS Support (1=on, 0 = off, string ident = "")
 - - Bit 7 = Touch ID Required (1=on, 0 = off, string ident = "")
 - - Bit 6 = RFU
 - - Bit 5 = RFU
 - - Bit 1,2,3,4
 - - - 0 = Payment Terminal
 - - - 1 = Transit Terminal
 - - - 2 = Access Terminal
 - - - 3 = Wireless Handoff Terminal
 - - - 4 = App Handoff Terminal
 - - - 15 = Other Terminal
- Byte 3 = RFU
- Byte 4 = Terminal Mode
 - - 0 = ApplePay VAS OR ApplePay
 - - 1 = ApplePay VAS AND ApplePay
 - - 2 = ApplePay VAS ONLY
 - - 3 = ApplePay ONLY 9F2B = 5 bytes = ApplePay VAS Filter. Each byte filters for that specific merchant index (optional, string ident = "") DF01 = 1 byte = ApplePay VAS Protocol. (optional, string ident = "")
 - - Bit 1 : 1 = URL VAS, 0 = Full VAS
 - - Bit 2 : 1 = VAS Beeps, 0 = No VAS Beeps
 - - Bit 3 : 1 = Silent Comm Error, 2 = EMEA Comm Error
 - - Bit 4-8 : RFU

21.10.2.71 RETURN_CODE IDTechSDK.IDT_NEO2.device_buzzer (string ident = " ")

Buzzer Device

Buzzer the reader. If success can hear one beep. Otherwise, returns timeout.

Parameters

<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.
--------------	--

Returns

RETURN_CODE: Values can be parsed with `errorCode.getErrorString(string ident = "")`

21.10.2.72 RETURN_CODE IDTechSDK.IDT_NEO2.device_buzzerOnOff (string *ident* = " ")

Buzzer On/Off

Cause the buzzer to beep once.

Parameters

<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.
--------------	--

Returns

RETURN_CODE: Values can be parsed with `device_getResponseCodeString`

21.10.2.73 RETURN_CODE IDTechSDK.IDT_NEO2.device_cancelTransaction (string *ident* = " ")

Cancel Transaction

Cancels the currently executing device transaction.

Parameters

<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.
--------------	--

Returns

RETURN_CODE: Values can be parsed with `errorCode.getErrorString()`

21.10.2.74 RETURN_CODE IDTechSDK.IDT_NEO2.device_certificateType (ref int *type*, string *ident* = " ")

Device Certificate Type

Returns the device certificate type

Parameters

<i>type</i>	0 = Unknown, 1 = Demo, 2 = Production
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with `device_getResponseCodeString`

21.10.2.75 RETURN_CODE IDTechSDK.IDT_NEO2.device_controlLED (byte *indexLED*, byte *control*, string *ident* = " ")

Control LED

Controls the LED for the reader. This command will only operate in pass-through mode

Parameters

<i>indexLED</i>	For LED <ul style="list-style-type: none">• 00: LED 0 (Power LED, leftmost LED)• 01: LED 1• 02: LED 2• 03: LED 3• FF: All LED's
<i>control</i>	LED Status: <ul style="list-style-type: none">• 00: LED Off• 01: LED On
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString

21.10.2.76 RETURN_CODE IDTechSDK.IDT_NEO2.device_controlUserInterface (byte[] *values*, string *ident* = " ")

Control User Interface

Controls the User Interface: Display, Beep, LED

Parameters

<i>values</i>	<p>Four bytes to control the user interface Byte[0] = LCD Message Messages 00-07 are normally controlled by the reader.</p> <ul style="list-style-type: none"> • 00h: Idle Message (Welcome, string ident = "") • 01h: Present card (Please Present Card, string ident = "") • 02h: Time Out or Transaction cancel (No Card, string ident = "") • 03h: Transaction between reader and card is in the middle (Processing...) • 04h: Transaction Pass (Thank You, string ident = "") • 05h: Transaction Fail (Fail, string ident = "") • 06h: Amount (Amount \$ 0.00 Tap Card, string ident = "") • 07h: Balance or Offline Available funds (Balance \$ 0.00) Messages 08-0B are controlled by the terminal • 08h: Insert or Swipe card (Use Chip & PIN, string ident = "") • 09h: Try Again(Tap Again, string ident = "") • 0Ah: Tells the customer to present only one card (Present 1 card only) • 0Bh: Tells the customer to wait for authentication/authorization (Wait) • FFh: indicates the command is setting the LED/Buzzer only. Byte[1] = Beep Indicator • 00h: No beep • 01h: Single beep • 02h: Double beep • 03h: Three short beeps • 04h: Four short beeps • 05h: One long beep of 200 ms • 06h: One long beep of 400 ms • 07h: One long beep of 600 ms • 08h: One long beep of 800 ms Byte[2] = LED Number • 00h: LED 0 (Power LED) 01h: LED 1 • 02h: LED 2 • 03h: LED 3 • FFh: All LEDs Byte[3] = LED Status • 00h: LED Off • 01h: LED On
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with `errorCode.getErrorString(string ident = "")`

21.10.2.77 RETURN_CODE IDTechSDK.IDT_NEO2.device_deleteDirectory (string filename, string ident = " ")**Delete Directory**

This command deletes an empty directory.

Parameters

<i>filename</i>	Complete path and file name of the directory you want to delete. You do not need to specify the root directory. Indicate subdirectories with a forward slash (/).
<i>ip</i>	Optional: IP address to execute command on (for IP connected devices, string ident = "")
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with `errorCode.getErrorString(string ident = "")`

21.10.2.78 RETURN_CODE IDTechSDK.IDT_NEO2.device_deleteFile (string filename, bool isSD = false, string ident = " ")**Delete File**

This command deletes a file or group of files.

Parameters

<i>filename</i>	Complete path and file name of the file you want to delete. You do not need to specify the root directory. Indicate subdirectories with a forward slash (/).
<i>isSD</i>	TRUE = Delete from SDCard, FALSE = Delete from Flash
<i>ip</i>	Optional: IP address to execute command on (for IP connected devices, string ident = "")
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with `errorCode.getErrorString(string ident = "")`

21.10.2.79 RETURN_CODE IDTechSDK.IDT_NEO2.device_disBlueLED (string ident = " ")**Disable Blue LED Sequence**

Stop the blue LEDs on the ViVOpay Vendi reader from flashing in left to right sequence and turn the LEDs off, and contactless function is disable at the same time.

Parameters

<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.
--------------	--

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString

21.10.2.80 RETURN_CODE IDTechSDK.IDT_NEO2.device_enableL100PassThrough (bool *enablePassThrough*, string *ident* = " ")

Enable L100 Pass Through

Enables Pass Through Mode for direct communication to L100 hooked up to NEO II device

Parameters

<i>enablePassThrough</i>	true = pass through ON, false = pass through OFF
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with errorCode.getErrorString(string ident = "")

21.10.2.81 RETURN_CODE IDTechSDK.IDT_NEO2.device_enableL80PassThrough (bool *enablePassThrough*, string *ident* = " ")

Enable L80 Pass Through

Enables Pass Through Mode for direct communication to L80 hooked up to NEO II device

Parameters

<i>enablePassThrough</i>	true = pass through ON, false = pass through OFF
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with errorCode.getErrorString(string ident = "")

21.10.2.82 RETURN_CODE IDTechSDK.IDT_NEO2.device_enablePassThrough (bool *enablePassThrough*, string *ident* = " ")

Enable Pass Through

Enables Pass Through Mode for direct communication with L1 interface (power on icc, send apdu, etc).

Parameters

<i>enablePassThrough</i>	true = pass through ON, false = pass through OFF
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with errorCode.getErrorString(string ident = "")

21.10.2.83 RETURN_CODE IDTechSDK.IDT_NEO2.device_enaBlueLED (byte[] dataCmd, string ident = " ")

Control the blue LED behavior on the Vendi reader.

Control the blue LED behavior on the Vendi reader.

Parameters

<i>dataCmd</i>	LED control. Minumum 4 bytes. Maximum 25 bytes. First byte is cycle. Next three bytes are sequence. Then sequence can repeat up to 8 times (24 bytes total for sequence, string ident = "") Byte 0 = Cycles (0 = Cycle once 1 = Repeat, string ident = "") Byte 1 = LED State Bitmap Bit - Description 8 = Left blue LED, 0 = off, 1 = on 7 = Center Blue LED, 0 = off, 1 = on 6 = Right Blue LED 0 = off, 1 = on 5 = Yellow LED, 0 = off, 1 = on 4 = Reserved for future use 3 = Reserved for future use 2 = Reserved for future use 1 = Reserved for future use Byte2~3 = Duration (Given in multiples of 10 millisecond, string ident = "") If cycle equals 1, more pairs would be after Byte 3.
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString

21.10.2.84 RETURN_CODE IDTechSDK.IDT_NEO2.device_enterStandbyMode (string ident = " ")

Enter Standby Mode

Puts unit into low power stand by mode

Parameters

<i>ip</i>	Optional: IP address to execute command on (for IP connected devices, string ident = "")
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString

21.10.2.85 RETURN_CODE IDTechSDK.IDT_NEO2.device_extendedErrorCondition (bool enable, string ident = " ")

Enable/disable Extended Error Condition

Enables/disables extended error condition for commands 02-40, 61-xx, 62-xx, 83-41 when error is 0xD0A or 0xD0B
String can be retrieved with getLastErrorString

Parameters

<i>enable</i>	TRUE = enable log, FALSE = disable log
<i>ip</i>	Optional: IP address to execute command on (for IP connected devices, string ident = "")
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString

21.10.2.86 **RETURN_CODE** IDTechSDK.IDT_NEO2.device_get1050BootloaderVersion (ref string *response*, string *ident* = " ")

Polls device for 1050 Bootloader Version

Parameters

<i>response</i>	Response returned of 1050 Bootloader Version
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with `errorCode.getErrorString(string ident = "")`

21.10.2.87 **RETURN_CODE** IDTechSDK.IDT_NEO2.device_get1050FuseStatus (ref byte[] *status*, string *ident* = " ")

RT1050 SRK Fuse Status

This command retrieves the status of the readers RT1050 SRK fuse.

Parameters

<i>status</i>	First 8 bytes of SRK Fuse, if programmed. Otherwise, empty array returned
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with `errorCode.getErrorString(string ident = "")`

21.10.2.88 **RETURN_CODE** IDTechSDK.IDT_NEO2.device_getAnyRKIStatus (bool *isMultiKey*, ref string *status*, ref Dictionary< string, RKI_KEY_TYPE > *keys*, string *ident* = " ")

Get RKI Status Polls the RKI servers to validate device RKI eligibility. Note: if device type is known in advance (production or demo device), it is more efficient to use `device_getRKIStatus` instead

Parameters

<i>isMultiKey</i>	True = MultiKey, False = SingleKey
<i>status</i>	RKI status returned as a string
<i>keys</i>	Returns all available keys
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with `errorCode.getErrorString()`. When no data is available, return code = **RETURN_CODE_NO_DATA_AVAILABLE**

21.10.2.89 **RETURN_CODE** IDTechSDK.IDT_NEO2.device_getAudioVolume (ref int *volume*, string *ident* = " ")

Get Audio Volume

Returns playback volume as represented by an integer

Parameters

<i>volume</i>	Value 0-20, where 0 is silent and 20 is full volume
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with `errorCode.getErrorString()`

21.10.2.90 RETURN_CODE IDTechSDK.IDT_NEO2.device_getBatteryVoltage (ref string *volume*, string *ident* = " ")

Polls device for Battery Voltage

Parameters

<i>volume</i>	Returns Battery Voltage string representing millivolts.
---------------	---

Return values

<i>RETURN_CODE</i>	<ul style="list-style-type: none"> • 0x0000: Success: no error - RETURN_CODE_DO_SUCCESS • 0x0001: Disconnect: no response from reader - RETURN_CODE_ERR_DISCONNECT • 0x0002: Invalid Response: invalid response data - RETURN_CODE_ERR_CMD_RESPONSE • 0x0003: Timeout: time out for task or CMD - RETURN_CODE_ERR_TIMEDOUT • 0x0004: Invalid Parameter: wrong parameter - RETURN_CODE_ERR_INVALID_PARAMETER • 0x0005: MSR Busy: SDK is doing MSR or ICC task - RETURN_CODE_SDK_BUSY_MSR • 0x0006: PINPad Busy: SDK is doing PINPad task - RETURN_CODE_SDK_BUSY_PINPAD • 0x0007: Unknown: Unknown error - RETURN_CODE_ERR_OTHER • 0x0100 through 0xFFFF refer to <code>errorCode.getErrorString(string ident = "")</code>
--------------------	---

21.10.2.91 RETURN_CODE IDTechSDK.IDT_NEO2.device_getBLEName (ref string *name*, string *ident* = " ")

Get BLE Name

Parameters

<i>response</i>	Name of BLE device
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with `errorCode.getErrorString(string ident = "")`

21.10.2.92 RETURN_CODE IDTechSDK.IDT_NEO2.device_getBootloaderVersion (ref string *response*, string *ident* = " ")

Polls device for Bootloader Version

Parameters

<i>response</i>	Response returned of Bootloader Version
<i>ip</i>	Optional: IP address to execute command on (for IP connected devices, string <i>ident</i> = "")
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with `errorCode.getErrorString(string ident = "")`

21.10.2.93 RETURN_CODE IDTechSDK.IDT_NEO2.device_getCashTranRiskPara (ref byte[] *tlv*, string *ident* = " ")

Get Cash Transaction Reader Risk Parameters Returns the TTQ and reader risk parameters that will be used for cash transactions, if enabled.

Parameters

<i>tlv</i>	TLV Data Objects
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with `errorCode.getErrorString(string ident = "")`

21.10.2.94 RETURN_CODE IDTechSDK.IDT_NEO2.device_getConfigurationFromMemory (ref string *json*, string *ident* = " ")

Read Device Configuration From Memory Retrieves the ViVOconfig configuration data from memory after a successful `device_readConfigurationToMemory`

Parameters

<i>json</i>	Returns the .json file data that can be saved to disk
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with `errorCode.getErrorString()`. When no data is available, return code = RETURN_CODE_NO_DATA_AVAILABLE

21.10.2.95 RETURN_CODE IDTechSDK.IDT_NEO2.device_getDeviceTime (ref DateTime *time*, string *ident* = " ")

Get Device Time

Parameters

<i>time</i>	Device Time
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString

21.10.2.96 RETURN_CODE IDTechSDK.IDT_NEO2.device_getDeviceTreeVersion (ref string *response*, bool *isDeviceTree*, string *ident* = " ")

Polls device for Device Tree Version

Parameters

<i>response</i>	Response returned of Device Tree Version
<i>isDeviceTree</i>	TRUE = Device Tree Version, FALSE = Device Tree 1050 Version
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with errorCode.getErrorString(string ident = "")

21.10.2.97 RETURN_CODE IDTechSDK.IDT_NEO2.device_getDRS (ref byte[] *codeDRS*, string *ident* = " ")

Get DRS Status

Gets the status of DRS(Destructive Reset).

Parameters

<i>codeDRS</i>	the data format is [DRS SourceBlk Number] [SourceBlk1] ... [SourceBlkN] [DRS SourceBlk Number] is 2 bytes, format is NumL NumH. It is Number of [SourceBlkX] [SourceBlkX] is n bytes, Format is [SourceID] [SourceLen] [SourceData] [SourceID] is 1 byte [SourceLen] is 1 byte, it is length of [SourceData]
----------------	--

[SourceID] [SourceLen] [SourceData] 00 1 01 – Application Error 01 1 01 – Application Error 02 1 0x01 – EMV L2 Configuration Check Value Error 0x02 – Future Key Check Value Error 10 1 01 – Battery Error 11 1 Bit 0 – Tamper Switch 1 (0-No, 1-Error) Bit 1– Tamper Switch 2 (0-No, 1-Error) Bit 2– Tamper Switch 3 (0-No, 1-Error) Bit 3– Tamper Switch 4 (0-No, 1-Error) Bit 4– Tamper Switch 5 (0-No, 1-Error) Bit 5– Tamper Switch 6 (0-No, 1-Error)

12 1 01 –TemperatureHigh or Low 13 1 01 –Voltage High or Low 1F 4 Reg31~24bits, Reg23~16bits, Reg15~8bits, Reg7~0bits

Returns

RETURN_CODE: Values can be parsed with errorCode.getErrorString()

21.10.2.98 RETURN_CODE IDTechSDK.IDT_NEO2.device_getFirmwareVersion (ref string *response*, string *ident* = " ")

Polls device for Firmware Version

Parameters

<i>response</i>	Response returned of Firmware Version *
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with `errorCode.getErrorString(string ident = "")`

21.10.2.99 RETURN_CODE IDTechSDK.IDT_NEO2.device_getHardwareInfor (ref string *ascii*, string *ident* = " ")

Get Hardware Information

Parameters

<i>ascii</i>	the ascii charactor
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

RETURN_CODE: Values can be parsed with `errorCode.getErrorString(string ident = "")`

ASCII | Description

HW,VPVendi<CR><LF>K21F Rev9 | Vendi HW,VP3300 Audio Jack<CR><LF>K21F Rev9 | Unipay III HW,V←
PUnipay1.5<CR><LF>K21F Rev9 | Unipay 1.5 HW,VPUniPay1.5TTK<CR><LF>K21F Rev9 | UniPay 1.5 TTK
HW,VP3300 USB<CR><LF>K21F Rev9 | VP3300 USB, VP3300 USB OEM (iBase/Cake same code, string ident
= "") HW,VP3300 USB-E<CR><LF>K21F Rev9 | VP3300 USB-E HW,VP3300 USB-C<CR><LF>K21F Rev9
| VP3300 USB-C HW,VPVP3300 Bluetooth<CR><LF>K21F Rev9 | VP3300 Bluetooth HW,.VP6300<CR><L←
F>K81F.Rev4 | VP6300

21.10.2.100 RETURN_CODE IDTechSDK.IDT_NEO2.device_getLightSensorVal (ref UInt16 *lightVal*, string *ident* = " ")

Get Light Sensor Value

Parameters

<i>lightVal</i>	Value of the light sensor
<i>ip</i>	Optional: IP address to execute command on (for IP connected devices, string ident = "")
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with `device_getResponseCodeString`

21.10.2.101 RETURN_CODE IDTechSDK.IDT_NEO2.device_getMerchantRecord (int *index*, ref byte[] *record*, string *ident* = " ")

Get Merchant Record Gets the merchant record for ApplePay VAS

Parameters

<i>index</i>	Merchant Record index, valid values 1-6
--------------	---

Parameters

<i>record</i>	Data returned containing 99 bytes: Byte 0 = Merchand Index Byte 1 = Merchant Enabled (1 = enabled, string ident = "") Byte 2 - 33 = Merchant Protocol Hash-256 value Byte 34 = Length of Merchant URL Bytes 35 - 99 = URL
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with `errorCode.getErrorString(string ident = "")`

21.10.2.102 RETURN_CODE IDTechSDK.IDT_NEO2.device_getModuleVer (ref string *moduleVer*, string *ident* = " ")

Get Module Version Information Get the 16 byte UID of MCU

Parameters

<i>uid</i>	string UID
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with `errorCode.getErrorString(string ident = "")`

21.10.2.103 RETURN_CODE IDTechSDK.IDT_NEO2.device_getMsrSecurePar (bool *b0*, bool *b1*, bool *b2*, bool *b3*, ref byte[] *tlv*, string *ident* = " ")

Get MSR Secure Parameters get parameters from flash setting

Parameters

<i>b0,b1,b2,b3</i>	Encryption Option Bit 0: T1 force encrypt Bit 1: T2 force encrypt Bit 2: T3 force encrypt Bit 3: T3 force encrypt when card type is 80
<i>tlv</i>	MSR Secure Parameters TVL objects
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with `errorCode.getErrorString(string ident = "")`

21.10.2.104 RETURN_CODE IDTechSDK.IDT_NEO2.device_getProcessorType (ref byte[] *type*, string *ident* = " ")

Get Processor Type Returns a processor type TLV

Parameters

<i>type</i>	processor type
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

RETURN_CODE: Values can be parsed with `errorCode.getErrorString(string ident = "")`

Processor Type | Description

45 00 | ARM7/ LPC21xx 4D 00 | ARM Cortex-M4/ K21 Family 4E 00 | ARM Cortex-M4/ K81 Family

21.10.2.105 **RETURN_CODE** `IDTechSDK.IDT_NEO2.device_getProductType (ref byte[] type, string ident = " ")`

Get Product Type Returns a "product type" value in a proprietary TLV

Parameters

<i>type</i>	product type
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

RETURN_CODE: Values can be parsed with `errorCode.getErrorString(string ident = "")`

Product Type | Description

42 37 00 | ViVOPay 5000 43 33 00 | ViVOPay 4500 43 35 00 | ViVOPay Vend 43 36 00 | Vendi (NEO, string ident = "") 43 37 00 | ViVOPay Kiosk1 (ATM1, string ident = "") 43 38 00 | Kiosk2 43 39 00 | Kiosk3 (NEO, string ident = "") 55 31 00 | UniPay 1.5 (NEO, string ident = "") 55 33 00 | UniPay III (NEO) 55 33 31 | VP3300, VP3300 OEM (NEO) (iBase/Cake same code, string ident = "") 55 33 32 | VP3300E(NEO, string ident = "") 55 33 33 | VP3300C(NEO, string ident = "") 55 33 34 | BTPay Mini (NEO) (UniPayIII + BLE, string ident = "") 56 31 00 | VP3600 56 32 00 | VP5200 56 33 00 | VP5300 56 34 00 | VP6300 56 35 00 | VP6800 56 36 00 | VP8300 56 37 00 | VP8310 56 38 00 | VP8800 56 39 00 | VP8810 56 40 00 | VP9000 44 30 00 | QX120 44 31 00 | Mx8Series 44 32 00 | NETs 44 33 00 | Magtek 44 35 00 | ICP

21.10.2.106 **RETURN_CODE** `IDTechSDK.IDT_NEO2.device_getRemoteKeyInjectionTO (ref int timeout, string ident = " ")`

Get Remote Key Injection Timeout

Parameters

<i>timeout</i>	Timeout is in seconds, value scope is [120, 3600]
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with `errorCode.getErrorString(string ident = "")`

21.10.2.107 **string** `IDTechSDK.IDT_NEO2.device_getResponseCodeString (RETURN_CODE eCode)`

Get the description of response result.

Parameters

<i>eCode</i>	the response result.
--------------	----------------------

Return values

<i>the</i>	string for description of response result
------------	---

- case 0: "no error, beginning task";
- case 1: "no response from reader";
- case 2: "invalid response data";
- case 3: "time out for task or CMD";
- case 4: "wrong parameter";
- case 5: "SDK is doing MSR or ICC task";
- case 6: "SDK is doing PINPad task";
- case 7: "SDK is doing CTLS task";
- case 8: "SDK is doing EMV task";
- case 9: "SDK is doing Other task";
- case 10: "err response or data";
- case 11: "no reader attached";
- case 12: "mono audio is enabled";
- case 13: "did connection";
- case 14: "audio volume is too low";
- case 15: "task or CMD be canceled";
- case 16: "UF wrong string format";
- case 17: "UF file not found";
- case 18: "UF wrong file format";
- case 19: "Attempt to contact online host failed";
- case 20: "Attempt to perform RKI failed";
- case 0x300: "Key Type(TDES) of Session Key is not same as the related Master Key.";
- case 0x400: "Related Key was not loaded.";
- case 0x500: "Key Same.";
- case 0x501: "Key is all zero";
- case 0x502: "TR-31 format error";
- case 0x702: "PAN is Error Key.";
- case 0x705: "No Internal MSR PAN (or Internal MSR PAN is erased timeout)";
- case 0X0C01: "Incorrect Frame Tag";
- case 0X0C02: "Incorrect Frame Type";
- case 0X0C03: "Unknown Frame Type";
- case 0X0C04: "Unknown Command";

- case 0X0C05: "Unknown Sub-Command";
- case 0X0C06: "CRC Error";
- case 0X0C07: "Failed";
- case 0X0C08: "Timeout";
- case 0X0C0A: "Incorrect Parameter";
- case 0X0C0B: "Command Not Supported";
- case 0X0C0C: "Sub-Command Not Supported";
- case 0X0C0D: "Parameter Not Supported / Status Abort Command";
- case 0X0C0F: "Sub-Command Not Allowed";
- case 0X0D01: "Incorrect Header Tag";
- case 0X0D02: "Unknown Command";
- case 0X0D03: "Unknown Sub-Command";
- case 0X0D04: "CRC Error in Frame";
- case 0X0D05: "Incorrect Parameter";
- case 0X0D06: "Parameter Not Supported";
- case 0X0D07: "Mal-formatted Data";
- case 0X0D08: "Timeout";
- case 0X0D0A: "Failed / NACK";
- case 0X0D0B: "Command not Allowed";
- case 0X0D0C: "Sub-Command not Allowed";
- case 0X0D0D: "Buffer Overflow (Data Length too large for reader buffer)";
- case 0X0D0E: "User Interface Event";
- case 0X0D11: "Communication type not supported, VT-1, burst, etc.";
- case 0X0D12: "Secure interface is not functional or is in an intermediate state.";
- case 0X0D13: "Data field is not mod 8";
- case 0X0D14: "Pad 0x80 not found where expected";
- case 0X0D15: "Specified key type is invalid";
- case 0X0D1: "Could not retrieve key from the SAM(InitSecureComm)";
- case 0X0D17: "Hash code problem";
- case 0X0D18: "Could not store the key into the SAM(InstallKey)";
- case 0X0D19: "Frame is too large";
- case 0X0D1A: "Unit powered up in authentication state but POS must resend the InitSecureComm command";
- case 0X0D1B: "The EEPROM may not be initialized because SecCommInterface does not make sense";
- case 0X0D1C: "Problem encoding APDU";
- case 0X0D20: "Unsupported Index(ILM) SAM Transceiver error – problem communicating with the SAM(Key Mgr)";

- case 0X0D2: "Unexpected Sequence Counter in multiple frames for single bitmap(ILM) Length error in data returned from the SAM(Key Mgr)";
- case 0X0D22: "Improper bit map(ILM)";
- case 0X0D23: "Request Online Authorization";
- case 0X0D24: "ViVOCard3 raw data read successful";
- case 0X0D25: "Message index not available(ILM) ViVOcomm activate transaction card type(ViVOcomm)";
- case 0X0D26: "Version Information Mismatch(ILM)";
- case 0X0D27: "Not sending commands in correct index message index(ILM)";
- case 0X0D28: "Time out or next expected message not received(ILM)";
- case 0X0D29: "ILM languages not available for viewing(ILM)";
- case 0X0D2A: "Other language not supported(ILM)";
- case 0X0D41: "Unknown Error from SAM";
- case 0X0D42: "Invalid data detected by SAM";
- case 0X0D43: "Incomplete data detected by SAM";
- case 0X0D44: "Reserved";
- case 0X0D45: "Invalid key hash algorithm";
- case 0X0D46: "Invalid key encryption algorithm";
- case 0X0D47: "Invalid modulus length";
- case 0X0D48: "Invalid exponent";
- case 0X0D49: "Key already exists";
- case 0X0D4A: "No space for new RID";
- case 0X0D4B: "Key not found";
- case 0X0D4C: "Crypto not responding";
- case 0X0D4D: "Crypto communication error";
- case 0X0D4E: "Module-specific error for Key Manager";
- case 0X0D4F: "All key slots are full (maximum number of keys has been installed)";
- case 0X0D50: "Auto-Switch OK";
- case 0X0D51: "Auto-Switch failed";
- case 0X0D90: "Account DUKPT Key not exist";
- case 0X0D91: "Account DUKPT Key KSN exhausted";
- case 0x0D00: "This Key had been loaded.";
- case 0x0E00: "Base Time was loaded.";
- case 0x0F00: "Encryption Or Decryption Failed.";
- case 0x1000: "Battery Low Warning (It is High Priority Response while Battery is Low.)";
- case 0x1800: "Send "Cancel Command" after send "Get Encrypted PIN" &"Get Numeric "&"Get Amount"";
- case 0x1900: "Press "Cancel" key after send "Get Encrypted PIN" &"Get Numeric "&"Get Amount"";

- case 0x30FF: "Security Chip is not connect";
- case 0x3000: "Security Chip is deactivation & Device is In Removal Legally State.";
- case 0x3101: "Security Chip is activation & Device is In Removal Legally State.";
- case 0x5500: "No Admin DUKPT Key.";
- case 0x5501: "Admin DUKPT Key STOP.";
- case 0x5502: "Admin DUKPT Key KSN is Error.";
- case 0x5503: "Get Authentication Code1 Failed.";
- case 0x5504: "Validate Authentication Code Error.";
- case 0x5505: "Encrypt or Decrypt data failed.";
- case 0x5506: "Not Support the New Key Type.";
- case 0x5507: "New Key Index is Error.";
- case 0x5508: "Step Error.";
- case 0x5509: "KSN Error";
- case 0x550A: "MAC Error.";
- case 0x550B: "Key Usage Error.";
- case 0x550C: "Mode Of Use Error.";
- case 0x550F: "Other Error.";
- case 0x6000: "Save or Config Failed / Or Read Config Error.";
- case 0x6200: "No Serial Number.";
- case 0x6900: "Invalid Command - Protocol is right, but task ID is invalid.";
- case 0x6A01: "Unsupported Command – Protocol and task ID are right, but command is invalid – In this State";
- case 0x6A00: "Unsupported Command - Protocol and task ID are right, but command is invalid.";
- case 0x6B00: "Unknown parameter in command - Protocol task ID and command are right, but parameter is invalid.";
- case 0x6C00: "Unknown parameter in command – Protocol task ID and command are right, but length is out of the requirement.";
- case 0x7200: "Device is suspend (MKSK suspend or press password suspend).";
- case 0x7300: "PIN DUKPT is STOP (21 bit 1).";
- case 0x7400: "Device is Busy.";
- case 0xE100: "Can not enter sleep mode";
- case 0xE200: "File has existed";
- case 0xE300: "File has not existed";
- case 0xE313: "IO line low -- Card error after session start";
- case 0xE400: "Open File Error";
- case 0xE500: "SmartCard Error";
- case 0xE600: "Get MSR Card data is error";

- case 0xE700: "Command time out";
- case 0xE800: "File read or write is error";
- case 0xE900: "Active 1850 error!";
- case 0xEA00: "Load bootloader error";
- case 0xEF00: "Protocol Error- STX or ETX or check error.";
- case 0xEB00: "Picture is not exist";
- case 0x2C02: "No Microprocessor ICC seated";
- case 0x2C06: "no card seated to request ATR";
- case 0x2D01: "Card Not Supported,";
- case 0x2D03: "Card Not Supported, wants CRC";
- case 0x690D: "Command not supported on reader without ICC support";
- case 0x8100: "ICC error time out on power-up";
- case 0x8200: "invalid TS character received - Wrong operation step";
- case 0x8300: "Decode MSR Error";
- case 0x8400: "TriMagII no Response";
- case 0x8500: "No Swipe MSR Card";
- case 0x8510: "No Financial Card";
- case 0x8600: "Unsupported F, D, or combination of F and D";
- case 0x8700: "protocol not supported EMV TD1 out of range";
- case 0x8800: "power not at proper level";
- case 0x8900: "ATR length too long";
- case 0x8B01: "EMV invalid TA1 byte value";
- case 0x8B02: "EMV TB1 required";
- case 0x8B03: "EMV Unsupported TB1 only 00 allowed";
- case 0x8B04: "EMV Card Error, invalid BWI or CWI";
- case 0x8B06: "EMV TB2 not allowed in ATR";
- case 0x8B07: "EMV TC2 out of range";
- case 0x8B08: "EMV TC2 out of range";
- case 0x8B09: "per EMV96 TA3 must be > 0xF";
- case 0x8B10: "ICC error on power-up";
- case 0x8B11: "EMV T=1 then TB3 required";
- case 0x8B12: "Card Error, invalid BWI or CWI";
- case 0x8B13: "Card Error, invalid BWI or CWI";
- case 0x8B17: "EMV TC1/TB3 conflict-";
- case 0x8B20: "EMV TD2 out of range must be T=1";
- case 0x8C00: "TCK error";

- case 0xA304: "connector has no voltage setting";
- case 0xA305: "ICC error on power-up invalid (SBLK(IFSD) exchange";
- case 0xE301: "ICC error after session start";
- case 0xFF00: "Request to go online";
- case 0xFF01: "EMV: Accept the offline transaction";
- case 0xFF02: "EMV: Decline the offline transaction";
- case 0xFF03: "EMV: Accept the online transaction";
- case 0xFF04: "EMV: Decline the online transaction";
- case 0xFF05: "EMV: Application may fallback to magstripe technology";
- case 0xFF06: "EMV: ICC detected tah the conditions of use are not satisfied";
- case 0xFF07: "EMV: ICC didn't accept transaction";
- case 0xFF08: "EMV: Transaction was cancelled";
- case 0xFF09: "EMV: Application was not selected by kernel or ICC format error or ICC missing data error";
- case 0xFF0A: "EMV: Transaction is terminated";
- case 0xFF0B: "EMV: Other EMV Error";
- case 0xFFFF: "NO RESPONSE";
- case 0xF002: "ICC communication timeout";
- case 0xF003: "ICC communication Error";
- case 0xF00F: "ICC Card Seated and Highest Priority, disable MSR work request";
- case 0xF200: "AID List / Application Data is not exist";
- case 0xF201: "Terminal Data is not exist";
- case 0xF202: "TLV format is error";
- case 0xF203: "AID List is full";
- case 0xF204: "Any CA Key is not exist";
- case 0xF205: "CA Key RID is not exist";
- case 0xF206: "CA Key Index it not exist";
- case 0xF207: "CA Key is full";
- case 0xF208: "CA Key Hash Value is Error";
- case 0xF209: "Transaction format error";
- case 0xF20A: "The command will not be processing";
- case 0xF20B: "CRL is not exist";
- case 0xF20C: "CRL number exceed max number";
- case 0xF20D: "Amount,Other Amount,Trasaction Type are missing";
- case 0xF20E: "The Identification of algorithm is mistake";
- case 0xF20F: "No Financial Card";
- case 0xF210: "In Encrypt Result state, TLV total Length is greater than Max Length";

- case 0x1001: "INVALID ARG";
- case 0x1002: "FILE_OPEN_FAILED";
- case 0x1003: "FILE OPERATION_FAILED";
- case 0x2001: "MEMORY_NOT_ENOUGH";
- case 0x3002: "SMARTCARD_FAIL";
- case 0x3003: "SMARTCARD_INIT_FAILED";
- case 0x3004: "FALLBACK_SITUATION";
- case 0x3005: "SMARTCARD_ABSENT";
- case 0x3006: "SMARTCARD_TIMEOUT";
- case 0x5001: "EMV_PARSING_TAGS_FAILED";
- case 0x5002: "EMV_DUPLICATE_CARD_DATA_ELEMENT";
- case 0x5003: "EMV_DATA_FORMAT_INCORRECT";
- case 0x5004: "EMV_NO_TERM_APP";
- case 0x5005: "EMV_NO_MATCHING_APP";
- case 0x5006: "EMV_MISSING_MANDATORY_OBJECT";
- case 0x5007: "EMV_APP_SELECTION_RETRY";
- case 0x5008: "EMV_GET_AMOUNT_ERROR";
- case 0x5009: "EMV_CARD_REJECTED";
- case 0x5010: "EMV_AIP_NOT_RECEIVED";
- case 0x5011: "EMV_AFL_NOT_RECEIVED";
- case 0x5012: "EMV_AFL_LEN_OUT_OF_RANGE";
- case 0x5013: "EMV_SFI_OUT_OF_RANGE";
- case 0x5014: "EMV_AFL_INCORRECT";
- case 0x5015: "EMV_EXP_DATE_INCORRECT";
- case 0x5016: "EMV_EFF_DATE_INCORRECT";
- case 0x5017: "EMV_ISS_COD_TBL_OUT_OF_RANGE";
- case 0x5018: "EMV_CRYPTOGAM_TYPE_INCORRECT";
- case 0x5019: "EMV_PSE_NOT_SUPPORTED_BY_CARD";
- case 0x5020: "EMV_USER_SELECTED_LANGUAGE";
- case 0x5021: "EMV_SERVICE_NOT_ALLOWED";
- case 0x5022: "EMV_NO_TAG_FOUND";
- case 0x5023: "EMV_CARD_BLOCKED";
- case 0x5024: "EMV_LEN_INCORRECT";
- case 0x5025: "CARD_COM_ERROR";
- case 0x5026: "EMV_TSC_NOT_INCREASED";
- case 0x5027: "EMV_HASH_INCORRECT";

- case 0x5028: "EMV_NO_ARC";
- case 0x5029: "EMV_INVALID_ARC";
- case 0x5030: "EMV_NO_ONLINE_COMM";
- case 0x5031: "TRAN_TYPE_INCORRECT";
- case 0x5032: "EMV_APP_NO_SUPPORT";
- case 0x5033: "EMV_APP_NOT_SELECT";
- case 0x5034: "EMV_LANG_NOT_SELECT";
- case 0x5035: "EMV_NO_TERM_DATA";
- case 0x6001: "CVM_TYPE_UNKNOWN";
- case 0x6002: "CVM_AIP_NOT_SUPPORTED";
- case 0x6003: "CVM_TAG_8E_MISSING";
- case 0x6004: "CVM_TAG_8E_FORMAT_ERROR";
- case 0x6005: "CVM_CODE_IS_NOT_SUPPORTED";
- case 0x6006: "CVM_COND_CODE_IS_NOT_SUPPORTED";
- case 0x6007: "NO_MORE_CVM";
- case 0x6008: "PIN_BYPASSED_BEFORE";
- case 0x7001: "PK_BUFFER_SIZE_TOO_BIG";
- case 0x7002: "PK_FILE_WRITE_ERROR";
- case 0x7003: "PK_HASH_ERROR";
- case 0x8001: "NO_CARD_HOLDER_CONFIRMATION";
- case 0x8002: "GET_ONLINE_PIN";
- case 0xD000: "Data not exist";
- case 0xD001: "Data access error";
- case 0xD100: "RID not exist";
- case 0xD101: "RID existed";
- case 0xD102: "Index not exist";
- case 0xD200: "Maximum exceeded";
- case 0xD201: "Hash error";
- case 0xD205: "System Busy";
- case 0x0E01: "Unable to go online";
- case 0x0E02: "Technical Issue";
- case 0x0E03: "Declined";
- case 0x0E04: "Issuer Referral transaction";
- case 0x0F01: "Decline the online transaction";
- case 0x0F02: "Request to go online";
- case 0x0F03: "Transaction is terminated";

- case 0x0F05: "Application was not selected by kernel or ICC format error or ICC missing data error";
- case 0x0F07: "ICC didn't accept transaction";
- case 0x0F0A: "Application may fallback to magstripe technology";
- case 0x0F0C: "Transaction was cancelled";
- case 0x0F0D: "Timeout";
- case 0x0F0F: "Other EMV Error";
- case 0x0F10: "Accept the offline transaction";
- case 0x0F11: "Decline the offline transaction";
- case 0x0F21: "ICC detected tah the conditions of use are not satisfied";
- case 0x0F22: "No app were found on card matching terminal configuration";
- case 0x0F23: "Terminal file does not exist";
- case 0x0F24: "CAPK file does not exist";
- case 0x0F25: "CRL Entry does not exist";
- case 0x0FFE: "code when blocking is disabled";
- case 0x0FFF: "code when command is not applicable on the selected device";
- case 0xF005: "ICC Encrypted C-APDU Data Structure Length Error Or Format Error.";
- case 0xBBE0: "CM100 Success";
- case 0xBBE1: "CM100 Parameter Error";
- case 0xBBE2: "CM100 Low Output Buffer";
- case 0xBBE3: "CM100 Card Not Found";
- case 0xBBE4: "CM100 Collision Card Exists";
- case 0xBBE5: "CM100 Too Many Cards Exist";
- case 0xBBE6: "CM100 Saved Data Does Not Exist";
- case 0xBBE8: "CM100 No Data Available";
- case 0xBBE9: "CM100 Invalid CID Returned";
- case 0xBBEA: "CM100 Invalid Card Exists";
- case 0xBBEC: "CM100 Command Unsupported";
- case 0xBBED: "CM100 Error In Command Process";
- case 0xBBEE: "CM100 Invalid Command";
- case 0X9031: "Unknown command";
- case 0X9032: "Wrong parameter (such as the length of the command is incorrect)";
- case 0X9038: "Wait (the command couldn't be finished in BWT)";
- case 0X9039: "Busy (a previously command has not been finished)";
- case 0X903A: "Number of retries over limit";
- case 0X9040: "Invalid Manufacturing system data";
- case 0X9041: "Not authenticated";

- case 0X9042: "Invalid Master DUKPT Key";
- case 0X9043: "Invalid MAC Key";
- case 0X9044: "Reserved for future use";
- case 0X9045: "Reserved for future use";
- case 0X9046: "Invalid DATA DUKPT Key";
- case 0X9047: "Invalid PIN Pairing DUKPT Key";
- case 0X9048: "Invalid DATA Pairing DUKPT Key";
- case 0X9049: "No nonce generated";
- case 0X9949: "No GUID available. Perform getVersion first.";
- case 0X9950: "MAC Calculation unsuccessful. Check BDK value.";
- case 0X904A: "Not ready";
- case 0X904B: "Not MAC data";
- case 0X9050: "Invalid Certificate";
- case 0X9051: "Duplicate key detected";
- case 0X9052: "AT checks failed";
- case 0X9053: "TR34 checks failed";
- case 0X9054: "TR31 checks failed";
- case 0X9055: "MAC checks failed";
- case 0X9056: "Firmware download failed";
- case 0X9060: "Log is full";
- case 0X9061: "Removal sensor unengaged";
- case 0X9062: "Any hardware problems";
- case 0X9070: "ICC communication timeout";
- case 0X9071: "ICC data error (such check sum error)";
- case 0X9072: "Smart Card not powered up";

21.10.2.108 RETURN_CODE IDTechSDK.IDT_NEO2.device_getRKIStatus (bool *isProd*, bool *isMultiKey*, ref string *status*, ref Dictionary< string, RKI_KEY_TYPE > *keys*, string *ident* = " ")

Get RKI Status From Specified Server Polls the RKI server to validate device RKI eligibility

Parameters

<i>isProd</i>	TRUE = poll production RKI server, FALSE = poll demo RKI server
<i>isMultiKey</i>	True = MultiKey, False = SingleKey
<i>status</i>	RKI status returned as a string
<i>keys</i>	Returns dictionary of available keys
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with `errorCode.getErrorString()`. When no data is available, return code = RETURN_CODE_NO_DATA_AVAILABLE

21.10.2.109 RETURN_CODE IDTechSDK.IDT_NEO2.device_getRT1050FirmwareVersion (ref string *response*, string *ident* = " ")

Get RT1050 Firmware Version

Parameters

<i>response</i>	Response returned of Firmware Version
<i>ip</i>	Optional: IP address to execute command on (for IP connected devices, string ident = "")
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with `errorCode.getErrorString(string ident = "")`

21.10.2.110 RETURN_CODE IDTechSDK.IDT_NEO2.device_getSelfCheckTime (ref byte *hour*, ref byte *minutes*, string *ident* = " ")

Get Self-Check Time

Gest the specific time for 24hrs self-check in Coordinated Universal Time

Parameters

<i>Hours,00-23h</i>	
<i>Minutes</i>	00-59h
<i>ip</i>	Optional: IP address to execute command on (for IP connected devices, string ident = "")
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with `device_getResponseCodeString`

21.10.2.111 RETURN_CODE IDTechSDK.IDT_NEO2.device_getTransactionResults (ref IDTTransactionData *results*, string *ident* = " ")

Get Transaction Results Gets the transaction results when the reader is functioning in "Auto Poll" mode

Parameters

<i>results</i>	The transaction results
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with `errorCode.getErrorString()`. When no data is available, return code = RETURN_CODE_NO_DATA_AVAILABLE

21.10.2.112 **RETURN_CODE** IDTechSDK.IDT_NEO2.device_getTransArmorID (ref string *TID*, string *ident* = " ")

Get TransArmor ID

Parameters

<i>TID</i>	TransArmor ID
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with `errorCode.getErrorString(string ident = "")`

21.10.2.113 **RETURN_CODE** IDTechSDK.IDT_NEO2.device_getUIDofMCU (ref string *uid*, string *ident* = " ")

Get UID of MCU

Parameters

<i>moduleVer</i>	module version information
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with `errorCode.getErrorString(string ident = "")`

21.10.2.114 **RETURN_CODE** IDTechSDK.IDT_NEO2.device_getUsbBootLoader (ref string *bootLoader*, string *ident* = " ")

Get USB Boot Loader Version Get the version of the USB Boot Loader

Parameters

<i>bootLoader</i>	USB boot loader information
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with `errorCode.getErrorString(string ident = "")`

21.10.2.115 **RETURN_CODE** IDTechSDK.IDT_NEO2.device_listDirectory (string *directoryName*, bool *recursive*, bool *onSD*, ref string *directory*, string *ident* = " ")

List Directory

This command retrieves a directory listing of user accessible files from the reader.

Parameters

<i>directoryName</i>	Directory Name. If null, root directory is listed
<i>recursive</i>	Included sub-directories
<i>onSD</i>	TRUE = use flash storage
<i>directory</i>	The returned directory information

Parameters

<i>ip</i>	Optional: IP address to execute command on (for IP connected devices, string <i>ident</i> = "")
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with `errorCode.getErrorString(string ident = "")`

21.10.2.116 `void IDTechSDK.IDT_NEO2.device_listenForNotifications (bool enable, string ident = " ")`

Listen for Notifications

Instructs SDK to listen for unsolicited data

Parameters

<i>enable</i>	TRUE = Listen, FALSE = Don't Listen
---------------	-------------------------------------

21.10.2.117 `RETURN_CODE IDTechSDK.IDT_NEO2.device_loadCertCA (byte certType, byte[] certData, string ident = " ")`

CA Cert (Intermediate Certificate) Loading

This command load the intermediate certificate for Application and TLS

Parameters

<i>certType</i>	<ul style="list-style-type: none"> • 0 = Application CA • 1 = TLS CA
<i>certData</i>	CA cert data
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with `device_getResponseCodeString`

21.10.2.118 `RETURN_CODE IDTechSDK.IDT_NEO2.device_logClear (string ident = " ")`

Clear Log

Instructs device to delete all log data

Parameters

<i>ip</i>	Optional: IP address to execute command on (for IP connected devices, string <i>ident</i> = "")
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString

21.10.2.119 RETURN_CODE IDTechSDK.IDT_NEO2.device_logEnable (bool *enable*, string *ident* = " ")

Enable Log

Instructs device to enable log

Parameters

<i>enable</i>	TRUE = enable log, FALSE = disable log
<i>ip</i>	Optional: IP address to execute command on (for IP connected devices, string <i>ident</i> = "")
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString

21.10.2.120 RETURN_CODE IDTechSDK.IDT_NEO2.device_logRead (DeviceLogCallback *callback*, string *ident* = " ")

Read Log

Instructs device to output all log data

Parameters

<i>callback</i>	DeviceLogCallback that will receive all log entries. If null, will be sent to MessageCallback, DeviceState.LogEvent
<i>ip</i>	Optional: IP address to execute command on (for IP connected devices, string <i>ident</i> = "")
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString

21.10.2.121 RETURN_CODE IDTechSDK.IDT_NEO2.device_lowPowerMode (bool *stopMode*, bool *wakeOnTrans*, string *ident* = " ")

Enter Low Power Mode

Puts terminal in sleep mor stop mode, with the option to wak on swipe/tap

Parameters

<i>stopMode</i>	TRUE = Stop Mode (POR required), FALSE = Sleep Mode (resume from last instruction, string <i>ident</i> = "")
<i>wakeOnTrans</i>	TRUE = Swipe/Tap will wake from low power, FALSE = will not wake on power from swipe/tap

Return values

<i>RETURN_CODE</i>	Return codes listed as typedef enum in IDTCommon:RETURN_CODE. Values can be parsed with IDT_NEO2::device_getResponseCodeString :(string ident = "")
--------------------	--

21.10.2.122 RETURN_CODE IDTechSDK.IDT_NEO2.device_offYellowLED (string *ident* = " ")

Turn Off Yellow LED

Turn off the ViVOpay Vendi reader yellow LED. This LED is located below the three blue LEDs.

Parameters

<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.
--------------	--

Returns

RETURN_CODE: Values can be parsed with [device_getResponseCodeString](#)

21.10.2.123 RETURN_CODE IDTechSDK.IDT_NEO2.device_onYellowLED (string *ident* = " ")

Turn On Yellow LED

Turn On the ViVOpay Vendi reader yellow LED. This LED is located below the three blue LEDs.

Parameters

<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.
--------------	--

Returns

RETURN_CODE: Values can be parsed with [device_getResponseCodeString](#)

21.10.2.124 RETURN_CODE IDTechSDK.IDT_NEO2.device_pingDevice (string *ident* = " ")

Ping Device

Pings the reader. If connected, returns success. Otherwise, returns timeout.

Parameters

<i>ip</i>	Optional: The callback function will only be applicable to the provided IP.
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with [errorCode.getErrorString](#)(string ident = "")

21.10.2.125 RETURN_CODE IDTechSDK.IDT_NEO2.device_playAudio (string *name*, int *type*, string *ident* = " ")

Play Audio

This command plays an audio file loaded from the inserted SD card. The VP6800 supports 16bit PCM format .WAV files.

Parameters

<i>name</i>	Name of file on SD Card
<i>type</i>	0 = Flash, 1 = SD Card
<i>ip</i>	Optional IP Address
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with `errorCode.getErrorString()`

21.10.2.126 RETURN_CODE IDTechSDK.IDT_NEO2.device_pollForToken (byte seconds, ref byte card, ref byte[] serialNumber, string ident = " ")

Poll for Token

Once Pass-Through Mode is started, ViVOpay will not poll for any cards until the "Poll for Token" command is received. This command tells ViVOpay to start polling for a Type A or Type B PICC until a PICC is detected or a timeout occurs.

This command automatically turns the RF Antenna on.

If a PICC is detected within the specified time limit, ViVOpay activates it and responds back to the terminal with card related data such as the Serial Number. If no PICC is detected within the specified time limit, ViVOpay stops polling and responds back indicating that no card was found. No card related data is returned in this case

Parameters

<i>timeout</i>	Timeout, in seconds to wait for card to be detected
<i>card</i>	Card Type: <ul style="list-style-type: none"> • 00h None (Card Not Detected or Could not Activate) • 01h ISO 14443 Type A (Supports ISO 14443-4 Protocol) • 02h ISO 14443 Type B (Supports ISO 14443-4 Protocol) • 03h Mifare Type A (Standard) • 04h Mifare Type A (Ultralight) • 05h ISO 14443 Type A (Does not support ISO 14443-4 Protocol) • 06h ISO 14443 Type B (Does not support ISO 14443-4 Protocol) • 07h ISO 14443 Type A and Mifare (NFC phone)
<i>serialNumber</i>	Serial Number or the UID of the PICC
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with `device_getResponseCodeString`

21.10.2.127 `RETURN_CODE IDTechSDK.IDT_NEO2.device_queryFile (string directory, string filename, bool isSD, ref bool exists, ref string timestamp, ref int fileSize, string ident = " ")`

Query File

Reports if the file exists, and if so will report the file timestamp and the file size

Parameters

<i>directory</i>	File directory. If blank/null, root directory is used
<i>filename</i>	Name of file to retrieve
<i>isSD</i>	TRUE = query SD card, FALSE = query internal memory
<i>isSD</i>	TRUE = query SD card, FALSE = query internal memory
<i>timestamp</i>	If file exists, reports the timestamp from the file in the form of YYYYMMDDMMSS
<i>fileSize</i>	If file exists, reports the size of the file

Return values

<i>RETURN_CODE</i>	<ul style="list-style-type: none"> • 0x0000: Success: no error - RETURN_CODE_DO_SUCCESS • 0x0001: Disconnect: no response from reader - RETURN_CODE_ERR_DISCONNECT • 0x0002: Invalid Response: invalid response data - RETURN_CODE_ERR_CMD_RESPONSE • 0x0003: Timeout: time out for task or CMD - RETURN_CODE_ERR_TIMEDOUT • 0x0004: Invalid Parameter: wrong parameter - RETURN_CODE_ERR_INVALID_PARAMETER • 0x0005: MSR Busy: SDK is doing MSR or ICC task - RETURN_CODE_SDK_BUSY_MSR • 0x0006: PINPad Busy: SDK is doing PINPad task - RETURN_CODE_SDK_BUSY_PINPAD • 0x0007: Unknown: Unknown error - RETURN_CODE_ERR_OTHER • 0x0100 through 0xFFFF refer to errorCode.getErrorString(string ident = "")
--------------------	--

21.10.2.128 `RETURN_CODE IDTechSDK.IDT_NEO2.device_readConfigurationToMemory (string memo, Configs.InstallRules rules, List< Configs.DeviceCommand > cmds, string ident = " ", bool isForeground = false)`

Read Device Configuration To Memory Executes a ViVOconfig read of device settings and stores the results in memory

Once a ViVOconfig read starts, all messages will be returned to the callback as DeviceState.ViVOconfig. The ViVOconfig operation is complete when the return code from a DeviceState.ViVOconfig message is one of the following values:

- RETURN_CODE_SDK_VIVOCONFIG_READ_CONFIG_SUCCESS = Read process completed successfully. Execute device_getConfigurationFromMemory to retrieve the configuration data
- RETURN_CODE_SDK_VIVOCONFIG_READ_CONFIG_FAILED = Read process FAILED.

Parameters

<i>memo</i>	Optional: populates the memo field of the captured .json file
<i>rules</i>	Optional: populates the .json file with available install rules
<i>cmds</i>	Optional: populates the .json file with commands to execute
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.
<i>isForeground</i>	If TRUE, will block program until update complete. Otherwise, FALSE performs update on background.

Returns

RETURN_CODE: Values can be parsed with `errorCode.getErrorString()`. When no data is available, return code = RETURN_CODE_NO_DATA_AVAILABLE

21.10.2.129 RETURN_CODE IDTechSDK.IDT_NEO2.device_readFileFromSD (string *directory*, string *filename*, ref byte[] *fileData*, string *ident* = " ")

Reads a file from the SD card

Parameters

<i>directory</i>	File directory. If blank/null, root directory is used
<i>filename</i>	Name of file to retrieve
<i>fileData</i>	the returned file data, if exists

Return values

<i>RETURN_CODE</i>	<ul style="list-style-type: none"> • 0x0000: Success: no error - RETURN_CODE_DO_SUCCESS • 0x0001: Disconnect: no response from reader - RETURN_CODE_ERR_DISCONNECT • 0x0002: Invalid Response: invalid response data - RETURN_CODE_ERR_CMD_RESPONSE • 0x0003: Timeout: time out for task or CMD - RETURN_CODE_ERR_TIMEDOUT • 0x0004: Invalid Parameter: wrong parameter - RETURN_CODE_ERR_INVALID_PARAMETER • 0x0005: MSR Busy: SDK is doing MSR or ICC task - RETURN_CODE_SDK_BUSY_MSR • 0x0006: PINPad Busy: SDK is doing PINPad task - RETURN_CODE_SDK_BUSY_PINPAD • 0x0007: Unknown: Unknown error - RETURN_CODE_ERR_OTHER • 0x0100 through 0xFFFF refer to <code>errorCode.getErrorString(string ident = "")</code>
--------------------	---

21.10.2.130 RETURN_CODE IDTechSDK.IDT_NEO2.device_rebootDevice (string *ident* = " ")

Reboot Device

Performs a reboot of the device

Parameters

<i>ip</i>	Optional: The callback function will only be applicable to the provided IP.
-----------	---

21.10.2.131 `RETURN_CODE IDTechSDK.IDT_NEO2.device_RemoteKeyInjection (RKL_KEY_TYPE type, string keyName, string ident = " ", bool performOnForeground = false)`

Remote Key Injection Performs a remote key injection for the device

Parameters

<i>type</i>	Remote Key Injection Type
<i>keyName</i>	Name of key (optional)
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.
<i>performOnForeground</i>	TRUE = block during process (default is false)

Returns

RETURN_CODE: Values can be parsed with `errorCode.getErrorString()`. When no data is available, return code = RETURN_CODE_NO_DATA_AVAILABLE

21.10.2.132 `RETURN_CODE IDTechSDK.IDT_NEO2.device_resetConfigurationGroup (int group, string ident = " ")`

Reset Configuration Group

This command allows resetting a dataset to its default configuration. If the file exists, it will be overwritten. If not, it will be created.

Parameters

<i>group</i>	Configuration Group
--------------	---------------------

Return values

<i>RETURN_CODE</i>	Return codes listed as typedef enum in IDTCommon:RETURN_CODE. Values can be parsed with IDT_VP8800::device_getResponseCodeString:(string ident = "")
--------------------	--

21.10.2.133 `RETURN_CODE IDTechSDK.IDT_NEO2.device_resetNVM (string ident = " ")`

Reset non-volatile memory.

- The Set Configuration Defaults and Keep Encryption Key command provides an external method for resetting parameters • in non-volatile memory (NVM) to their default values. When the reader receives this serial command it • erases EEPROM (but retains encryption keys). After completing initialization, the reader reboots.

Returns

RETURN_CODE: Values can be parsed with `device_getResponseCodeString`

21.10.2.134 RETURN_CODE IDTechSDK.IDT_NEO2.device_resetTransaction (string *ident* = " ")

Reset Transaction

Abruptly terminates the currently executing device transaction. Does not produce any further transaction data or notifications after executing.

Parameters

<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.
--------------	--

Returns

RETURN_CODE: Values can be parsed with `errorCode.getErrorString()`

21.10.2.135 RETURN_CODE IDTechSDK.IDT_NEO2.device_retrieveAIDList (ref byte *response*[], string *ident* = " ")

Retrieve AID list

Returns all the AID names and their assigned groups installed on the terminal for CTLS/CONTACT.

Parameters

<i>response</i>	array of 2-tag TLV data objects: DFEE2D (group name) followed by 9F06 (AID, string <i>ident</i> = "")
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with `errorCode.getErrorString(string ident = "")`

21.10.2.136 RETURN_CODE IDTechSDK.IDT_NEO2.device_retrieveTerminalData (ref byte[] *tlv*, string *ident* = " ")

Retrieve Terminal Data

Retrieves the Terminal Data for CTLS. This is configuration group 0 (Tag FFEE - > FFEE0100). The terminal data can also be retrieved by `ctls_getConfiguraitonGroup(0)`.

Parameters

<i>tlv</i>	Response returned as a TLV
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with `errorCode.getErrorString()`

21.10.2.137 RETURN_CODE IDTechSDK.IDT_NEO2.device_rrcConnect (string *ident* = " ")

RRC Connect

The RRC Connect command allows a host to establish an RRC connection to a reader. A host must first establish an RRC connection to the reader before issuing other RRC IDG commands.

Parameters

<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.
--------------	--

Return values

<i>RETURN_CODE</i>	Return codes listed as typedef enum in IDTCommon:RETURN_CODE. Values can be parsed with IDT_NEO2::device_getResponseCodeString :(string ident = "")
--------------------	--

21.10.2.138 **RETURN_CODE** IDTechSDK.IDT_NEO2.device_rrcDisconnect (string *ident* = " ")

RRC Disconnect

The RRC Disonnect command allows a host to terminate it's existing RRC connection with a reader.

Parameters

<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.
--------------	--

Return values

<i>RETURN_CODE</i>	Return codes listed as typedef enum in IDTCommon:RETURN_CODE. Values can be parsed with IDT_NEO2::device_getResponseCodeString :(string ident = "")
--------------------	--

21.10.2.139 **RETURN_CODE** IDTechSDK.IDT_NEO2.device_rrcDownloadApp (string *appName*, byte[] *appData*, string *ident* = " ")

RRC Download Application

The RRC Download Application command allows the transfer of a compressed application file from a host to the reader, extracts it, and performs signature verification on its contents.

Parameters

<i>appName</i>	This will be the name of the application that will appear in Application Manager application list.
<i>appData</i>	Application Data
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Return values

<i>RETURN_CODE</i>	Return codes listed as typedef enum in IDTCommon:RETURN_CODE. Values can be parsed with IDT_NEO2::device_getResponseCodeString :(string ident = "")
--------------------	--

21.10.2.140 **RETURN_CODE** IDTechSDK.IDT_NEO2.device_rrcInstallApp (string *appName*, string *ident* = " ")

RRC Install Application

The RRC Install Application command installs a downloaded application to a reader. Only installed applications can run on the reader.

Parameters

<i>appName</i>	This will be the name of the application that will appear in Application Manager application list.
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Return values

<i>RETURN_CODE</i>	Return codes listed as typedef enum in IDTCommon:RETURN_CODE. Values can be parsed with IDT_NEO2::device_getResponseCodeString :(string ident = "")
--------------------	---

21.10.2.141 **RETURN_CODE** IDTechSDK.IDT_NEO2.device_rrcRunApp (string *appName*, string *ident* = " ")

RRC Run Application

The RRC Run Application command allows the reader to run an installed application.

Parameters

<i>appName</i>	This will be the name of the application that will appear in Application Manager application list.
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Return values

<i>RETURN_CODE</i>	Return codes listed as typedef enum in IDTCommon:RETURN_CODE. Values can be parsed with IDT_NEO2::device_getResponseCodeString :(string ident = "")
--------------------	---

21.10.2.142 **RETURN_CODE** IDTechSDK.IDT_NEO2.device_rrcUninstallApp (string *appName*, string *ident* = " ")

RRC Uninstall Application

The RRC Uninstall Application command uninstalls an application on a reader device. The application will remain in the file system of the reader but cannot be executed.

Parameters

<i>appName</i>	This will be the name of the application that will appear in Application Manager application list.
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Return values

<i>RETURN_CODE</i>	Return codes listed as typedef enum in IDTCommon:RETURN_CODE. Values can be parsed with IDT_NEO2::device_getResponseCodeString :(string ident = "")
--------------------	---

21.10.2.143 **RETURN_CODE** IDTechSDK.IDT_NEO2.device_sendConfiguration (string *filename*, VC_OPERATION_TYPE *type*, bool *matchFW*, string *ident* = " ", bool *isForeground* = false)

Send Configuration Executes a ViVOconfig update or verify to a device

Once a ViVOconfig write or verify starts, all messages will be returned to the callback as DeviceState.ViVOconfig.

The ViVOconfig operation is complete when the return code from a DeviceState.ViVOconfig message is one of the following values:

- RETURN_CODE_SDK_VIVOCONFIG_VERIFY_SUCCESS = Verification process completed successfully. No differences found.
- RETURN_CODE_SDK_VIVOCONFIG_VERIFY_SUCCESS_WITH_WARNING = Verification process completed with warnings.
- RETURN_CODE_SDK_VIVOCONFIG_VERIFY_FAILED = Verification process FAILED
- RETURN_CODE_SDK_VIVOCONFIG_WRITE_SUCCESS = Write process completed successfully.
- RETURN_CODE_SDK_VIVOCONFIG_WRITE_SUCCESS_WITH_WARNING = Write process completed with warnings
- RETURN_CODE_SDK_VIVOCONFIG_WRITE_FAILED = Write process FAILED

Parameters

<i>filename</i>	The .json configuration file to update the device with NOTE: You can also use this parameter to pass the path of where the .json file is located at on local storage instead
<i>filename</i>	The name of the .json configuration file
<i>type</i>	VC_OPERATION_TYPE

- VC_OPERATION_TYPE_WRITE = Write File To Device, Hash must validate
- VC_OPERATION_TYPE_VERIFY = Verify Device With File
- VC_OPERATION_TYPE_WRITE_IGNORE_HASH = Write File To Device, Ignore Hash
- VC_OPERATION_TYPE_WRITE_FIX_HASH = Write File To Device, Fix Hash If Necessary

Parameters

<i>matchFW</i>	TRUE = Device FW must match file FW, FALSE = Don't validate device FW against file
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.
<i>isForeground</i>	If TRUE, will block program until update complete. Otherwise, FALSE performs update on background.

Returns

RETURN_CODE: Values can be parsed with `errorCode.getErrorString()`. When no data is available, return code = RETURN_CODE_NO_DATA_AVAILABLE

21.10.2.144 RETURN_CODE IDTechSDK.IDT_NEO2.device_sendConfigurationFromZip (byte[] zip, string filename, VC_OPERATION_TYPE type, bool matchFW, string ident = " ", bool isForeground = false)

Send Configuration From Zip Executes a ViVOconfig update or verify to a device using files from a .zip archive

Once a ViVOconfig write or update starts, all messages will be returned to the callback as DeviceState.ViVOconfig. The ViVOconfig operation is complete when the return code from a DeviceState.ViVOconfig message is one of the following values:

- RETURN_CODE_SDK_VIVOCONFIG_VERIFY_SUCCESS = Verification process completed successfully. No differences found.
- RETURN_CODE_SDK_VIVOCONFIG_VERIFY_SUCCESS_WITH_WARNING = Verification process completed with warnings.

- RETURN_CODE_SDK_VIVOCONFIG_VERIFY_FAILED = Verification process FAILED
- RETURN_CODE_SDK_VIVOCONFIG_WRITE_SUCCESS = Write process completed successfully.
- RETURN_CODE_SDK_VIVOCONFIG_WRITE_SUCCESS_WITH_WARNING = Write process completed with warnings
- RETURN_CODE_SDK_VIVOCONFIG_WRITE_FAILED = Write process FAILED

Parameters

<i>zip</i>	.Zip archive containing the .json configuration file and also any associated image assets NOTE: You can also use this parameter to pass the path of where the .zip file is located at on local storage instead
<i>filename</i>	The name of the .json configuration file
<i>type</i>	VC_OPERATION_TYPE

- VC_OPERATION_TYPE_WRITE = Write File To Device, Hash must validate
- VC_OPERATION_TYPE_VERIFY = Verify Device With File
- VC_OPERATION_TYPE_WRITE_IGNORE_HASH = Write File To Device, Ignore Hash
- VC_OPERATION_TYPE_WRITE_FIX_HASH = Write File To Device, Fix Hash If Necessary

Parameters

<i>matchFW</i>	TRUE = Device FW must match file FW, FALSE = Don't validate device FW against file
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.
<i>isForeground</i>	If TRUE, will block program until update complete. Otherwise, FALSE performs update on background.

Returns

RETURN_CODE: Values can be parsed with `errorCode.getErrorString()`. When no data is available, return code = RETURN_CODE_NO_DATA_AVAILABLE

21.10.2.145 RETURN_CODE IDTechSDK.IDT_NEO2.device_sendDataCommand (string *cmd*, bool *calcLRC*, ref byte[] *response*, string *ident* = " ")

Send a data command to the device

Sends a command to the device.

Parameters

<i>cmd</i>	String representation of command to execute
<i>calcLRC</i>	If TRUE, this will wrap command with start/length/lrc/sum/end: '{STX}{Len_Low}{Len_High} data {CheckLRC} {CheckSUM} {ETX}'
<i>response</i>	Response data
<i>ip</i>	Optional IP parameter
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString

21.10.2.146 RETURN_CODE IDTechSDK.IDT_NEO2.device_sendDataCommand_ext (string *cmd*, bool *calcLRC*, ref byte[] *response*, int *timeout*, bool *noResponse*, string *ident* = " ")

Send a data command to the device - extended

Sends a command to the device.

Parameters

<i>cmd</i>	String representation of command to execute
<i>calcLRC</i>	If TRUE, this will wrap command as NGA with start/length/lrc/sum/end: '{STX}{Len_Low}{Len_High} data {CheckLRC} {CheckSUM} {ETX}'
<i>response</i>	Response data
<i>timeout</i>	Timeout value waiting for response, in milliseconds (1000 = 1 second, string <i>ident</i> = "")
<i>noResponse</i>	if TRUE, this will not wait for a response and immediately return SUCCESS
<i>calcITP</i>	If TRUE, this will wrap command as ITP with start/end/lrc: '{STX} data {ETX}{CheckLRC}'
<i>ip</i>	Optional IP parameter
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString

21.10.2.147 RETURN_CODE IDTechSDK.IDT_NEO2.device_sendPAE (string *command*, ref string *response*, int *timeout*, string *ident* = " ")

Send Payment Application Engine Command

Executes a PAE command

Parameters

<i>command</i>	ASCII command string, should start with "*PAE"
<i>response</i>	command response
<i>timeout</i>	timeout waiting for PAE response
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with errorCode.getErrorString(string *ident* = "")

21.10.2.148 RETURN_CODE IDTechSDK.IDT_NEO2.device_sendVivoCommandP2 (byte *command*, byte *subCommand*, byte[] *data*, ref byte[] *response*, string *ident* = " ")

Send Vivo Command Protocol 2

Sends a protocol 2 command to Vivo readers (IDG/NEO, string *ident* = "")

Parameters

<i>command</i>	Command
<i>subCommand</i>	Sub-Command
<i>data</i>	Data. May be null
<i>response</i>	Response
<i>ip</i>	Optional IP
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with `errorCode.getErrorString(string ident = "")`

21.10.2.149 RETURN_CODE IDTechSDK.IDT_NEO2.device_sendVivoCommandP2_ext (byte *command*, byte *subCommand*, byte[] *data*, ref byte[] *response*, int *timeout*, bool *noResponse*, string *ident* = " ")

Send Vivo Command Protocol 2 Extended

Sends a protocol 2 command to Vivo readers (IDG/NEO, string *ident* = "")

Parameters

<i>command</i>	Command
<i>subCommand</i>	Sub-Command
<i>data</i>	Data. May be null
<i>response</i>	Response
<i>timeout</i>	Timeout, in milliseconds (3000 = 3 seconds, string <i>ident</i> = "")
<i>noResponse</i>	TRUE = don't wait for response, FALSE = wait for response defined by timeout
<i>ip</i>	Optional IP
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with `errorCode.getErrorString(string ident = "")`

21.10.2.150 RETURN_CODE IDTechSDK.IDT_NEO2.device_sendVivoCommandP3 (byte *command*, byte *subCommand*, byte[] *data*, ref byte[] *response*, string *ident* = " ")

Send Vivo Command Protocol 3

Sends a protocol 3 command to Vivo readers (IDG/NEO, string *ident* = "")

Parameters

<i>command</i>	Command
<i>subCommand</i>	Sub-Command
<i>data</i>	Data. May be null
<i>response</i>	Response
<i>ip</i>	Optional IP
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with `errorCode.getErrorString(string ident = "")`

21.10.2.151 RETURN_CODE IDTechSDK.IDT_NEO2.device_sendVivoCommandP3_ext (byte *command*, byte *subCommand*, byte[] *data*, ref byte[] *response*, int *timeout*, bool *noResponse*, string *ident* = " ")

Send Vivo Command Protocol 3 Extended

Sends a protocol 3 command to Vivo readers (IDG/NEO, string ident = "")

Parameters

<i>command</i>	Command
<i>subCommand</i>	Sub-Command
<i>data</i>	Data. May be null
<i>response</i>	Response
<i>timeout</i>	Timeout, in milliseconds (3000 = 3 seconds, string ident = "")
<i>noResponse</i>	TRUE = don't wait for response, FALSE = wait for response defined by timeout
<i>ip</i>	Optional IP
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with `errorCode.getErrorString(string ident = "")`

21.10.2.152 RETURN_CODE IDTechSDK.IDT_NEO2.device_sendVivoCommandP4 (byte *command*, byte *subCommand*, byte[] *data*, ref byte[] *response*, string *ident* = " ")

Send Vivo Command Protocol 4

Sends a protocol 4 command to Vivo readers (IDG/NEO, string ident = "")

Parameters

<i>command</i>	Command
<i>subCommand</i>	Sub-Command
<i>data</i>	Data. May be null
<i>response</i>	Response
<i>ip</i>	Optional IP
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with `errorCode.getErrorString(string ident = "")`

21.10.2.153 RETURN_CODE IDTechSDK.IDT_NEO2.device_sendVivoCommandP4_ext (byte *command*, byte *subCommand*, byte[] *data*, ref byte[] *response*, int *timeout*, bool *noResponse*, string *ident* = " ")

Send Vivo Command Protocol 4 Extended

Sends a protocol 4 command to Vivo readers (IDG/NEO, string ident = "")

Parameters

<i>command</i>	Command
<i>subCommand</i>	Sub-Command
<i>data</i>	Data. May be null
<i>response</i>	Response
<i>timeout</i>	Timeout, in milliseconds (3000 = 3 seconds, string ident = "")
<i>noResponse</i>	TRUE = don't wait for response, FALSE = wait for response defined by timeout
<i>ip</i>	Optional IP
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with `errorCode.getErrorString(string ident = "")`

21.10.2.154 RETURN_CODE IDTechSDK.IDT_NEO2.device_setAudioVolume (int *volume*, string *ident* = " ")

Set Audio Volume

Sets the audio playback volume

Parameters

<i>volume</i>	Value 0-20, where 0 is silent and 20 is full volume
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with `errorCode.getErrorString()`

21.10.2.155 RETURN_CODE IDTechSDK.IDT_NEO2.device_setBurstMode (byte *mode*, string *ident* = " ")

Send Burst Mode

Sets the burst mode for the device.

Parameters

<i>mode</i>	0 = OFF, 1 = Always On, 2 = Auto Exit
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with `errorCode.getErrorString(string ident = "")`

21.10.2.156 RETURN_CODE IDTechSDK.IDT_NEO2.device_setLED (byte *indexLED*, byte *control*, string *ident* = " ")

Set LED

Control the reader. If connected, returns success. Otherwise, returns timeout.

Parameters

<i>indexLED</i>	description as follows: 00h: LED 0 (Power LED, string ident = "") 01h: LED 1 02h: LED 2 03h: LED 3 10h: Single Tri-Color LED (Unipay III used, string ident = "") FFh: All 4 LEDs + Single Tri-Color LED Where the LEDs are numbered 0, 1, 2, 3 counting from the left. Note: If you are using pass-through mode to control the Power LED (LED 0), it is your responsibility to make sure that it behaves correctly.
<i>control</i>	description as follows: 00h: LED Off (LED 0~4 + Tri-Color LED, string ident = "") 01h: LED On (LED 0~4, string ident = "") 02h: Green Color (Tri-Color LED, string ident = "") 03h: Red Color (Tri-Color LED, string ident = "") 04h: Amber Color(Tri-Color LED, string ident = "")
<i>ip</i>	Optional: The callback function will only be applicable to the provided IP.
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with `errorCode.getErrorString(string ident = "")`

21.10.2.157 RETURN_CODE IDTechSDK.IDT_NEO2.device_setMerchantRecord (int *index*, bool *enabled*, string *merchantID*, string *merchantURL*, string *ident* = " ")

Set Merchant Record Sets the merchant record for ApplePay VAS

Parameters

<i>index</i>	Merchant Record index, valid values 1-6
<i>enabled</i>	Merchant Enabled/Valid flag
<i>merchantID</i>	Merchant unique identifier registered with Apple. Example com.idtechproducts.applePay
<i>merchantURL</i>	Merchant URL, when applicable
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with `errorCode.getErrorString(string ident = "")`

21.10.2.158 RETURN_CODE IDTechSDK.IDT_NEO2.device_setPollMode (byte *mode*, string *ident* = " ")

Send Poll Mode

Sets the poll mode for the device. Auto Poll keeps reader active, Poll On Demand only polls when requested by terminal

Parameters

<i>mode</i>	0 = Auto Poll, 1 = Poll On Demand
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with `errorCode.getErrorString(string ident = "")`

21.10.2.159 **RETURN_CODE** IDTechSDK.IDT_NEO2.device_setSelfCheckTime (byte *hour*, byte *minutes*, string *ident* = " ")

Set Self-Check Time

Set a specific time for 24hrs self-check in Coordinated Universal Time

Parameters

<i>Hours</i>	00-23h
<i>Minutes</i>	00-59h
<i>ip</i>	Optional: IP address to execute command on (for IP connected devices, string <i>ident</i> = "")
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString

21.10.2.160 **RETURN_CODE** IDTechSDK.IDT_NEO2.device_setTerminalData (byte[] *tlv*, string *ident* = " ")

Set Terminal Data

Sets the Terminal Data for CTLS as specified by the TLV. The terminal global data is group 0. Other groups can be defined using this method (1 or greater), and those can be retrieved with emv_getConfigurationGroup(int group), and deleted with emv_removeConfigurationGroup(int group). You cannot delete group 0.

Parameters

<i>tlv</i>	TerminalData configuration data
------------	---------------------------------

Return values

RETURN_CODE	Return codes listed as typedef enum in IDTCommon:RETURN_CODE. Values can be parsed with IDT_NEO2::device_getResponseCodeString :(string <i>ident</i> = "")
--------------------	---

21.10.2.161 **RETURN_CODE** IDTechSDK.IDT_NEO2.device_setTransArmorEncryption (byte[] *cert*, string *ident* = " ")

Set TransArmor Encryption

Parameters

<i>cert</i>	Certificate in PEM format or DER format. PEM format must be string data (converted to binary) starting with "----". DER format is binary data
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with errorCode.getErrorString(string *ident* = "")

21.10.2.162 **RETURN_CODE** IDTechSDK.IDT_NEO2.device_setTransArmorID (string *TID*, string *ident* = " ")

Set TransArmor ID

Parameters

<i>TID</i>	TransArmor ID
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with `errorCode.getErrorString(string ident = "")`

21.10.2.163 RETURN_CODE IDTechSDK.IDT_NEO2.device_StartRKI (int type, string ident = " ")

Start Remote Key Injection

Starts a remote key injection request with IDTech RKI servers. Set/Get RKI url with `IDT_Device.RKI_URL`.

Parameters

<i>type</i>	0 = Symmetric RKI Demo Unit 1 = Symmetric RKI Production Unit 2 = PKI-RKI Demo Unit 3 = PKI-RKI Production Unit
<i>ip</i>	Optional IP address
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with `device_getResponseCodeString`

21.10.2.164 RETURN_CODE IDTechSDK.IDT_NEO2.device_startRKI (bool isTest, string ident = " ")

Start Remote Key Injection

Starts a remote key injection request with IDTech RKI servers.

Parameters

<i>isTest</i>	TRUE = Demo Device, FALSE = Production Device
<i>ip</i>	Optional IP address
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with `device_getResponseCodeString`

21.10.2.165 RETURN_CODE IDTechSDK.IDT_NEO2.device_startTransaction (double amount, double amtOther, int exponent, int type, int timeout, byte[] tags, bool isFastEMV = false, string ident = " ")

Start a Transaction Request

Authorizes the transaction CTLS, MSR or Contact EMV

The tags will be returned in the callback routine.

Parameters

<i>amount</i>	Transaction amount value (tag value 9F02, string ident = "")
<i>amtOther</i>	Other amount value, if any (tag value 9F03, string ident = "")
<i>exponent</i>	Number of characters after decimile point
<i>type</i>	Transaction type (tag value 9C).
<i>timeout</i>	Timeout value in seconds.
<i>tags</i>	Any other tags to be included in the request. Passed as TLV Data. Example, tag 9F02 with amount 0x000000000100 would be 0x9F0206000000000100 If tags 9F02 (amount), 9F03 (other amount), or 9C (transaction type) are included, they will take priority over these values supplied as individual parameters to this method.
<i>isFastEMV</i>	If TRUE, it will populate the IDTTransactionData.fastEMV with ASCII data similar to IDTech FastEMV KB output, after performing an auto-authenticate and auto-complete with ResultCode = Could Not Contact Host
<i>ip</i>	Optional IP parameter
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with `errorCode.getErrorString()`

NOTE ON APPLEPAY VAS: To enable ApplePay VAS, first a merchant record must be defined in one of the six available index positions (1-6) using `device_setMerchantRecord`, then container tag FFEE06 must be sent as part of the additional tags parameter of `ctls_startTransaction`. Tag FFEE06 must contain tag 9F26 and 9F22, and can optionally contain tags 9F2B and DFO1. Example FFEE06189F220201009F26040000000009F2B050100000000↵DF010101 9F22 = two bytes = ApplePay Terminal Application Version Number. Hard defined as 0100 for now. (required, string ident = "") 9F26 = four bytes = ApplePay Terminal Capabilities Information (required, string ident = "")

- Byte 1 = RFU
- Byte 2 = Terminal Type
- - Bit 8 = VAS Support (1=on, 0 = off, string ident = "")
- - Bit 7 = Touch ID Required (1=on, 0 = off, string ident = "")
- - Bit 6 = RFU
- - Bit 5 = RFU
- - Bit 1,2,3,4
- - - 0 = Payment Terminal
- - - 1 = Transit Terminal
- - - 2 = Access Terminal
- - - 3 = Wireless Handoff Terminal
- - - 4 = App Handoff Terminal
- - - 15 = Other Terminal
- Byte 3 = RFU
- Byte 4 = Terminal Mode
- - 0 = ApplePay VAS OR ApplePay
- - 1 = ApplePay VAS AND ApplePay

- - 2 = ApplePay VAS ONLY
- - 3 = ApplePay ONLY 9F2B = 5 bytes = ApplePay VAS Filter. Each byte filters for that specific merchant index (optional, string ident = "") DF01 = 1 byte = ApplePay VAS Protocol. (optional, string ident = "")
- - Bit 1 : 1 = URL VAS, 0 = Full VAS
- - Bit 2 : 1 = VAS Beeps, 0 = No VAS Beeps
- - Bit 3 : 1 = Silent Comm Error, 2 = EMEA Comm Error
- - Bit 4-8 : RFU

21.10.2.166 `RETURN_CODE IDTechSDK.IDT_NEO2.device_startTransactionCB (double amount, double amtOther, int exponent, int type, int timeout, byte[] tags, CallbackIP callback, string ident = "", bool isFastEMV = false)`

Start a Transaction Request

Authorizes the transaction CTLS, MSR or Contact EMV with results returned to the specified callback

The tags will be returned in the callback routine.

Parameters

<i>amount</i>	Transaction amount value (tag value 9F02, string ident = "")
<i>amtOther</i>	Other amount value, if any (tag value 9F03, string ident = "")
<i>exponent</i>	Number of characters after decimile point
<i>type</i>	Transaction type (tag value 9C).
<i>timeout</i>	Timeout value in seconds.
<i>tags</i>	Any other tags to be included in the request. Passed as TLV Data. Example, tag 9F02 with amount 0x000000000100 would be 0x9F0206000000000100 If tags 9F02 (amount), 9F03 (other amount), or 9C (transaction type) are included, they will take priority over these values supplied as individual parameters to this method.
<i>callback</i>	Callback for returning transaction results
<i>isFastEMV</i>	If TRUE, it will populate the IDTTransactionData.fastEMV with ASCII data similar to IDTech FastEMV KB output, after performing an auto-authenticate and auto-complete with ResultCode = Could Not Contact Host
<i>ip</i>	Optional IP parameter
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

`RETURN_CODE`: Values can be parsed with `errorCode.getErrorString()`

NOTE ON APPLEPAY VAS: To enable ApplePay VAS, first a merchant record must be defined in one of the six available index positions (1-6) using `device_setMerchantRecord`, then container tag FFEE06 must be sent as part of the additional tags parameter of `ctls_startTransaction`. Tag FFEE06 must contain tag 9F26 and 9F22, and can optionally contain tags 9F2B and DF01. Example FFEE06189F220201009F2604000000009F2B050100000000DF010101 9F22 = two bytes = ApplePay Terminal Application Version Number. Hard defined as 0100 for now. (required, string ident = "") 9F26 = four bytes = ApplePay Terminal Capabilities Information (required, string ident = "")

- Byte 1 = RFU
- Byte 2 = Terminal Type
- - Bit 8 = VAS Support (1=on, 0 = off, string ident = "")

- - Bit 7 = Touch ID Required (1=on, 0 = off, string ident = "")
- - Bit 6 = RFU
- - Bit 5 = RFU
- - Bit 1,2,3,4
- - - 0 = Payment Terminal
- - - 1 = Transit Terminal
- - - 2 = Access Terminal
- - - 3 = Wireless Handoff Terminal
- - - 4 = App Handoff Terminal
- - - 15 = Other Terminal
- Byte 3 = RFU
- Byte 4 = Terminal Mode
- - 0 = ApplePay VAS OR ApplePay
- - 1 = ApplePay VAS AND ApplePay
- - 2 = ApplePay VAS ONLY
- - 3 = ApplePay ONLY 9F2B = 5 bytes = ApplePay VAS Filter. Each byte filters for that specific merchant index (optional, string ident = "") DF01 = 1 byte = ApplePay VAS Protocol. (optional, string ident = "")
- - Bit 1 : 1 = URL VAS, 0 = Full VAS
- - Bit 2 : 1 = VAS Beeps, 0 = No VAS Beeps
- - Bit 3 : 1 = Silent Comm Error, 2 = EMEA Comm Error
- - Bit 4-8 : RFU

21.10.2.167 **RETURN_CODE** IDTechSDK.IDT_NEO2.device_stopAudio (string *ident* = " ")

Stop Audio

This command stop playing audio started with lcd_playAudio.

Parameters

<i>ip</i>	Optional IP Address
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with `errorCode.getErrorString()`

21.10.2.168 **RETURN_CODE** IDTechSDK.IDT_NEO2.device_terminalInfo (ref byte[] *tlv*, string *ident* = " ")

Retrieve Terminal Info

Parameters

<i>tlv</i>	TLV Tags assigned as follows: <ul style="list-style-type: none"> • 0x01 = Date Time, 6 bytes Y M D H M S • 0x02 = Device Model Number • 0x03 = Firmware Version • 0x04 = hardware Version • 0x05 = Serial Number • 0x06 Last swipe UTC/RTC • 0x07 Life time total swipe • 0x08 Last Dip (ICC) UTC/RTC • 0x09 Lifetime Total Dip (ICC, string ident = "") • 0x0A Last Tap (CL) UTC/RTC • 0x0B Lifetime Total Tap (CL, string ident = "") • 0x0C Reboot Count • 0x0D Device Uptime since Last Reboot • 0x0E Device Lifetime Uptime • 0x0F Tamper Status
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString

21.10.2.169 RETURN_CODE IDTechSDK.IDT_NEO2.device_transferFile (string *fileName*, byte[] *file*, bool *isSD* = false, string *ident* = " ")

Transfer File

This command transfers a data file to the reader.

Parameters

<i>fileName</i>	Filename. The data for this command is a ASCII string with the complete path and file name you want to create. You do not need to specify the root directory. Indicate subdirectories with a forward slash (/).
<i>file</i>	The data file.
<i>isSD</i>	TRUE = tranfer to SD Card, FALSE = transfer to Flash.
<i>ip</i>	Optional: IP address to execute command on (for IP connected devices, string ident = "")
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with errorCode.getErrorString(string ident = "")

21.10.2.170 `static RETURN_CODE IDTechSDK.IDT_NEO2.device_updateDeviceFirmware (byte[] firmwareData, string ident = "") [static]`

Update K81 Firmware

Updates the firmware .

Parameters

<i>firmwareData</i>	Signed binary data of a firmware file provided by IDTech
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with `errorCode.getErrorString(string ident = "")`

After you pass the `firmwareData` file, a new thread will start to execute the firmware download. You will receive status of the progress through callbacks to the `IDTechSDK.Callback()` delegate. The following parameters will be passed back:

- `sender = IDT_DEVICE_Types.IDT_DEVICE_AUGUSTA`
- `state = DeviceState.FirmwareUpdate`
- `transactionResultCode =` status of the firmware update (starting, entering bootloader, applying update, block success, firmware success, string `ident = ""`)
- `data =` File Progress. Four bytes, with bytes `[0][1] =` current block, and bytes `[2][3] =` total blocks. `0x00030010 =` block 3 of 16

Example code starting a firmware update

```
OpenFileDialog diag = new OpenFileDialog( ident);
diag.Filter = "NGA FW Files|*.fm";

if (diag.ShowDialog() == DialogResult.OK, string ident = "")
{
    byte[] file = File.ReadAllBytes(diag.FileName, ident);
    RETURN_CODE rt = IDT_Device.SharedController.device_updateDeviceFirmware(file, ident);
    if (rt == RETURN_CODE.RETURN_CODE_DO_SUCCESS, string ident = "")
    {
        //Was a success
    }
    else
    {
        //Error starting firmware download
    }
}
```

Example monitoring firmware update status / success

```
private void MessageCallBack(IDTechSDK.IDT_DEVICE_Types type, DeviceState state, byte[] data,
    IDTTransactionData cardData, EMV_Callback emvCallback, RETURN_CODE transactionResultCode, string ident = "")
{
    switch (state, string ident = "")
    {
        case DeviceState.FirmwareUpdate:
            switch (transactionResultCode, string ident = "")
            {
                case RETURN_CODE.RETURN_CODE_FW_STARTING_UPDATE:
                    SetOutputText("Starting Firmware Update\n", ident);
                    break;
                case RETURN_CODE.RETURN_CODE_DO_SUCCESS:
                    SetOutputText("Firmware Update Successful\n", ident);
                    break;
                case RETURN_CODE.RETURN_CODE_APPLYING_FIRMWARE_UPDATE:
                    break;
            }
        break;
    }
}
```



```

        SetOutputText("Applying Firmware Update...\n", ident);
        break;
    case RETURN_CODE.RETURN_CODE_ENTERING_BOOTLOADER_MODE:
        SetOutputText("Entering Bootloader Mode...\n", ident);
        break;
    case RETURN_CODE.RETURN_CODE_BLOCK_TRANSFER_SUCCESS:

        int start = data[0] * 0x100 + data[1];
        int end = data[2] * 0x100 + data[3];

        SetOutputText("Sent block " + start.ToString() + " of " + end.ToString() + "\n", ident)
;
        break;
    default:
        SetOutputText("Firmware Update Error Code: " + "0x" + String.Format("{0:X}", (ushort)
transactionResultCode) + ": " + IDTechSDK.errorCode.getErrorString(transactionResultCode) + "\r\n",
ident);
        break;
    }
    break;
}
}

```

21.10.2.171 RETURN_CODE IDTechSDK.IDT_NEO2.device_updateDeviceFromManifest (string filepath, string ident = "", bool isForeground = false)

Update Device From Manifest

Initiates a device update using a manifest file.

Parameters

<i>filepath</i>	File path of the manifest file. It must be a valid .json manifest file, and must be located in the same folder as all the needed firmware update assets
<i>ip</i>	Optional Device Identifier
<i>isForeground</i>	Optional. If TRUE, function blocks until complete, other if FALSE, runs on background thread
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString

21.10.2.172 RETURN_CODE IDTechSDK.IDT_NEO2.device_updateFirmwareFromZip (byte[] zipfile, string ident = "", bool isForeground = false)

Update Firmware From Zip

Performs one or more device firmware updates from firmware files passed as a compressed archive with a valid configuration file (.json format)

Parameters

<i>zipfile</i>	Zip file containing on or more firmware update files and a .json configuration file
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.
<i>isForeground</i>	If TRUE, will block program until firmware update complete. Otherwise, FALSE performs FW update on background.

Returns

RETURN_CODE: Values can be parsed with errorCode.getErrorString(string ident = "")

21.10.2.173 **RETURN_CODE** IDTechSDK.IDT_NEO2.device_updateFirmwareIP (string *path*, int *type*, FirmwareUpdateCallback *callback*, string *ip*, string *ident* = " ", bool *performOnForeground* = false)

Update Firmware

Updates the firmware over IP .

Parameters

<i>path</i>	Local filepath to the signed binary data of a firmware file provided by IDTech
<i>type</i>	0 = K81, 1 = 1050
<i>callback</i>	Callback to receive the status updates
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with `errorCode.getErrorString(string ident = "")`

After you pass the filename file, a new thread will start to execute the firmware download. You will receive status of the progress through callback to the `IDTechSDK.FirmwareUpdateCallback()` delegate. The following parameters will be passed back:

- `transactionResultCode` = status of the firmware update (starting, entering bootloader, applying update, block success, firmware success, string `ident` = "")
- `data` = File Progress. Four bytes, with bytes `[0][1]` = current block, and bytes `[2][3]` = total blocks. `0x00030010` = block 3 of 16
- `IP`. Ip address of the device

Example code starting a firmware update

```
OpenFileDialog diag = new OpenFileDialog( ident);
diag.Filter = "FW Files|*.fm";

if (diag.ShowDialog() == DialogResult.OK, string ident = "")
{
    RETURN_CODE rt = IDT_Device.SharedController.device_updateFirmwareIP(diag.FileName,0,callback,"
    10.12.34.96", ident);
    if (rt == RETURN_CODE.RETURN_CODE_DO_SUCCESS, string ident = "")
    {
        //Was a success
    }
    else
    {
        //Error starting firmware download
    }
}
```

Example monitoring firmware update status / success

```
private void callback( byte[] data, RETURN_CODE transactionResultCode, string IP, string ident = "")
{
    switch (transactionResultCode, string ident = "")
    {
        case RETURN_CODE.RETURN_CODE_FW_STARTING_UPDATE:
            SetOutputText("Starting Firmware Update\n", ident);
            break;
        case RETURN_CODE.RETURN_CODE_DO_SUCCESS:
            SetOutputText("Firmware Update Successful\n", ident);
            break;
        case RETURN_CODE.RETURN_CODE_APPLYING_FIRMWARE_UPDATE:
            SetOutputText("Applying Firmware Update...\n", ident);
            break;
    }
}
```

```

        break;
    case RETURN_CODE.RETURN_CODE_ENTERING_BOOTLOADER_MODE:
        SetOutputText("Entering Bootloader Mode...\n", ident);
        break;
    case RETURN_CODE.RETURN_CODE_BLOCK_TRANSFER_SUCCESS:

        int start = data[0] * 0x100 + data[1];
        int end = data[2] * 0x100 + data[3];

        SetOutputText("Sent block " + start.ToString() + " of " + end.ToString() + "\n", ident)
;
        break;
    default:
        SetOutputText("Firmware Update Error Code: " + "0x" + String.Format("{0:X}", (ushort)
transactionResultCode) + ": " + IDTechSDK.errorCode.getErrorString(transactionResultCode) + "\r\n",
ident);
        break;
}

```

21.10.2.174 RETURN_CODE IDTechSDK.IDT_NEO2.device_updateFirmwareKernels (string path, FirmwareUpdateCallback callback, string ip = "", UInt32 address = 0, string ident = "")

Update Firmware Kernels

Updates all the recognized firmware kernels from the provided folder path .

Parameters

<i>path</i>	Local folder that contains the kernel files. NOTE: The files must start with the following names to be properly recognized <ul style="list-style-type: none"> • app_ct_l2 • app_nfcrd_l2_amex • app_nfcrd_l2_apvas • app_nfcrd_l2_cup • app_nfcrd_l2_dpass • app_nfcrd_l2_interac • app_nfcrd_l2_jcb • app_nfcrd_l2_mchip • app_nfcrd_l2_ppse • app_nfcrd_l2_smarttap • app_nfcrd_l2_vcps
<i>callback</i>	Optional firmware callback. Otherwise, will use main callback
<i>ip</i>	Optional ip address of device
<i>address</i>	Required for ADF_SDK and ADF_APP. Start address
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with errorCode.getErrorString(string ident = "")

After you pass the folder name, a new thread will start to execute the kernel download. You will receive status of the progress through callback to the IDTechSDK.FirmwareUpdateCallback() delegate. The following parameters will be passed back:

- transactionResultCode = status of the firmware update (starting, entering bootloader, applying update, block success, firmware success, string ident = "")
- data = File Progress. Four bytes, with bytes [0][1] = current block, and bytes [2][3] = total blocks. 0x00030010 = block 3 of 16
- IP. Ip address of the device

21.10.2.175 static RETURN_CODE IDTechSDK.IDT_NEO2.device_updateFirmwareType (FIRMWARE_TYPE type, byte[] firmwareData, string ident = " ", UInt32 address = 0, bool performOnForeground = false, int manifestScriptsCount = 0, int processedScriptCount = 0, string DT_PRJ_filename = " ") [static]

Update App Firmware

Updates the firmware

Parameters

<i>type</i>	FIRMWARE_TYPE. <ul style="list-style-type: none"> • FIRMWARE_TYPE_External_EMV_CTL2 • FIRMWARE_TYPE_EMV_CLL2_PPSE • FIRMWARE_TYPE_EMV_CLL2_VCPS • FIRMWARE_TYPE_EMV_CLL2_MChip • FIRMWARE_TYPE_EMV_CLL2_DPass • FIRMWARE_TYPE_EMV_CLL2_Amex • FIRMWARE_TYPE_EMV_CLL2_Interac • FIRMWARE_TYPE_EMV_CLL2_CUP • FIRMWARE_TYPE_EMV_CLL2_JCB • FIRMWARE_TYPE_APVAS • FIRMWARE_TYPE_SmartTap • FIRMWARE_TYPE_SCRP • FIRMWARE_TYPE_ADF_SDK • FIRMWARE_TYPE_ADF_App • FIRMWARE_TYPE_1050 • FIRMWARE_TYPE_1050_DEVICE_TREE FIRMWARE_TYPE_DEVICE_TREE_DEF FIRMWARE_TYPE_DEVICE_TREE_PRJ • FIRMWARE_TYPE_NEO3_BOOTLOADER • FIRMWARE_TYPE_K81_BOOTLOADER_A • FIRMWARE_TYPE_K81_BOOTLOADER_B • FIRMWARE_TYPE_K81
<i>firmwareData</i>	Signed binary data of a firmware file provided by IDTech
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.
<i>address</i>	Required for ADF_SDK and ADF_APP. Start address
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with `errorCode.getErrorString(string ident = "")`

After you pass the `firmwareData` file, a new thread will start to execute the firmware download. You will receive status of the progress through callbacks to the `IDTechSDK.Callback()` delegate. The following parameters will be passed back:

- `sender = IDT_DEVICE_Types.IDT_DEVICE_AUGUSTA`
- `state = DeviceState.FirmwareUpdate`
- `transactionResultCode = status of the firmware update (starting, entering bootloader, applying update, block success, firmware success, string ident = "")`
- `data = File Progress. Four bytes, with bytes [0][1] = current block, and bytes [2][3] = total blocks. 0x00030010 = block 3 of 16`

Example code starting a firmware update

```
OpenFileDialog diag = new OpenFileDialog( ident);
diag.Filter = "NGA FW Files|*.fm";

if (diag.ShowDialog() == DialogResult.OK, string ident = "")
{
    byte[] file = File.ReadAllBytes(diag.FileName, ident);
    RETURN_CODE rt = IDT_Device.SharedController.device_updateFirmware(FIRMWARE_TYPE.FIRMWARE_TYPE_1050,
        file, ident);
    if (rt == RETURN_CODE.RETURN_CODE_DO_SUCCESS, string ident = "")
    {
        //Was a success
    }
    else
    {
        //Error starting firmware download
    }
}
```

Example monitoring firmware update status / success

```
private void MessageCallBack(IDTechSDK.IDT_DEVICE_Types type, DeviceState state, byte[] data,
    IDTTransactionData cardData, EMV_Callback emvCallback, RETURN_CODE transactionResultCode, string ident = "")
{
    switch (state, string ident = "")
    {
        case DeviceState.FirmwareUpdate:
            switch (transactionResultCode, string ident = "")
            {
                case RETURN_CODE.RETURN_CODE_FW_STARTING_UPDATE:
                    SetOutputText("Starting Firmware Update\n", ident);
                    break;
                case RETURN_CODE.RETURN_CODE_DO_SUCCESS:
                    SetOutputText("Firmware Update Successful\n", ident);
                    break;
                case RETURN_CODE.RETURN_CODE_APPLYING_FIRMWARE_UPDATE:
                    SetOutputText("Applying Firmware Update...\n", ident);
                    break;
                case RETURN_CODE.RETURN_CODE_ENTERING_BOOTLOADER_MODE:
                    SetOutputText("Entering Bootloader Mode...\n", ident);
                    break;
                case RETURN_CODE.RETURN_CODE_BLOCK_TRANSFER_SUCCESS:
                    int start = data[0] * 0x100 + data[1];
                    int end = data[2] * 0x100 + data[3];

                    SetOutputText("Sent block " + start.ToString() + " of " + end.ToString() + "\n", ident);
                    break;
                default:
                    SetOutputText("Firmware Update Error Code: " + "0x" + String.Format("{0:X}", (ushort)
                        transactionResultCode) + ": " + IDTechSDK.errorCode.getErrorString(transactionResultCode) + "\r\n",
                        ident);
                    break;
            }
        break;
    }
}
```

21.10.2.176 **RETURN_CODE** IDTechSDK.IDT_NEO2.device_updateFirmwareType (string *path*, FIRMWARE_TYPE *type*, FirmwareUpdateCallback *callback*, string *ident* = " ", UInt32 *address* = 0)

Update Firmware

Updates the firmware through path .

Parameters

<i>path</i>	Local filepath to the signed binary data of a firmware file provided by IDTech
<i>type</i>	FIRMWARE_TYPE. <ul style="list-style-type: none"> • FIRMWARE_TYPE_External_EMV_CTL2 • FIRMWARE_TYPE_EMV_CLL2_PPSE • FIRMWARE_TYPE_EMV_CLL2_VCPS • FIRMWARE_TYPE_EMV_CLL2_MChip • FIRMWARE_TYPE_EMV_CLL2_DPass • FIRMWARE_TYPE_EMV_CLL2_Amex • FIRMWARE_TYPE_EMV_CLL2_Interac • FIRMWARE_TYPE_EMV_CLL2_CUP • FIRMWARE_TYPE_EMV_CLL2_JCB • FIRMWARE_TYPE_APVAS • FIRMWARE_TYPE_SmartTap • FIRMWARE_TYPE_SCRP • FIRMWARE_TYPE_ADF_SDK • FIRMWARE_TYPE_ADF_App • FIRMWARE_TYPE_1050 • FIRMWARE_TYPE_1050_DEVICE_TREE FIRMWARE_TYPE_DEVICE_TREE_DEF FIRMWARE_TYPE_DEVICE_TREE_PRJ • FIRMWARE_TYPE_NEO3_BOOTLOADER • FIRMWARE_TYPE_K81_BOOTLOADER_A • FIRMWARE_TYPE_K81_BOOTLOADER_B • FIRMWARE_TYPE_K81
<i>callback</i>	Optional firmware callback. Otherwise, will use main callback
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.
<i>address</i>	Required for ADF_SDK and ADF_APP. Start address

Returns

RETURN_CODE: Values can be parsed with `errorCode.getErrorString(string ident = "")`

After you pass the filename file, a new thread will start to execute the firmware download. You will receive status of the progress through callback to the `IDTechSDK.FirmwareUpdateCallback()` delegate. The following parameters will be passed back:

- transactionResultCode = status of the firmware update (starting, entering bootloader, applying update, block success, firmware success, string ident = "")
- data = File Progress. Four bytes, with bytes [0][1] = current block, and bytes [2][3] = total blocks. 0x00030010 = block 3 of 16
- IP. Ip address of the device

Example code starting a firmware update

```
OpenFileDialog diag = new OpenFileDialog( ident);
diag.Filter = "FW Files|*.fm";

if (diag.ShowDialog() == DialogResult.OK, string ident = "")
{
    RETURN_CODE rt = IDT_Device.SharedController.device_updateFirmwareIP(diag.FileName,0,callback,"
    10.12.34.96", ident);
    if (rt == RETURN_CODE.RETURN_CODE_DO_SUCCESS, string ident = "")
    {
        //Was a success
    }
    else
    {
        //Error starting firmware download
    }
}
```

Example monitoring firmware update status / success

```
private void callback( byte[] data, RETURN_CODE transactionResultCode, string IP, string ident = "")
{
    switch (transactionResultCode, string ident = "")
    {
        case RETURN_CODE.RETURN_CODE_FW_STARTING_UPDATE:
            SetOutputText("Starting Firmware Update\n", ident);
            break;
        case RETURN_CODE.RETURN_CODE_DO_SUCCESS:
            SetOutputText("Firmware Update Successful\n", ident);
            break;
        case RETURN_CODE.RETURN_CODE_APPLYING_FIRMWARE_UPDATE:
            SetOutputText("Applying Firmware Update...\n", ident);
            break;
        case RETURN_CODE.RETURN_CODE_ENTERING_BOOTLOADER_MODE:
            SetOutputText("Entering Bootloader Mode...\n", ident);
            break;
        case RETURN_CODE.RETURN_CODE_BLOCK_TRANSFER_SUCCESS:
            int start = data[0] * 0x100 + data[1];
            int end = data[2] * 0x100 + data[3];

            SetOutputText("Sent block " + start.ToString() + " of " + end.ToString() + "\n", ident);
            break;
        default:
            SetOutputText("Firmware Update Error Code: " + "0x" + String.Format("{0:X}", (ushort)
            transactionResultCode) + ": " + IDTechSDK.errorCode.getErrorString(transactionResultCode) + "\r\n",
            ident);
            break;
    }
}
```

21.10.2.177 static RETURN_CODE IDTechSDK.IDT_NEO2.device_wakeDevice (string macAddress = "", string ipAddress = "") [static]

Wake Device From Standby Mode

If an IP connected device, the device can be located by it's MAC Address, or IP Address (if the SDK has previ
When a IP device is first connected, the MacAddress and IP Address are stored in a look up table.
Only one of the two parameter are required for Wake on WAN. If IP address is provided, it will take priority

Parameters

<i>macAddress</i>	Optional: Mac Address of IP connected device to wake
<i>ipAddress</i>	Optional: IP Address of IP connected device to wake
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with `device_getResponseCodeString`

21.10.2.178 **RETURN_CODE** IDTechSDK.IDT_NEO2.emv_activateTransaction (int *timeout*, byte[] *tags*, bool *forceOnline*, bool *isFastEMV* = false, string *ident* = " ")

Start EMV Transaction Request

Authorizes the EMV transaction for an ICC card

The tags will be returned in the callback routine.

Parameters

<i>timeout</i>	Timeout value in seconds.
<i>tags</i>	The tags to be included in the request. Passed as TLV Data. Example, tag 9F02 with amount 0x000000000100 would be 0x9F0206000000000100
<i>forceOnline</i>	TRUE = do not allow offline approval, FALSE = allow ICC to approve offline if terminal capable Note: To request tags to be included in default response, use tag DFEE1A, and specify tag list. Example four tags 9F02, 9F36, 95, 9F37 to be included in response = DFEE1A079F029F369F37
<i>isFastEMV</i>	If TRUE, it will populate the IDTTransactionData.fastEMV with ASCII data similar to IDTech FastEMV KB output, after performing an auto-authenticate and auto-complete with ResultCode = Could Not Contact Host
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with `device_getResponseCodeString`

21.10.2.179 **RETURN_CODE** IDTechSDK.IDT_NEO2.emv_addTerminalData (byte[] *tlv*, string *ident* = " ")

Add Terminal Data

Adds to the exosting Terminal Data

Parameters

<i>tlv</i>	TerminalData TLV Data
<i>ident</i>	Optional identifier

Return values

RETURN_CODE	Return codes listed as typedef enum in IDTCommon:RETURN_CODE. Values can be parsed with <code>IDT_BTPay::device_getResponseCodeString:()</code>
--------------------	---

21.10.2.180 `static void IDTechSDK.IDT_NEO2.emv_allowFallback (bool allow, string ident = " ") [static]`

Allow fallback for EMV transactions. Default is TRUE

Parameters

<i>allow</i>	TRUE = allow fallback, FALSE = don't allow fallback
--------------	---

21.10.2.181 `RETURN_CODE IDTechSDK.IDT_NEO2.emv_authenticateTransaction (byte[] updatedTLV, string ident = " ")`

Authenticate EMV Transaction Request

Authenticates the EMV transaction for an ICC card. Execute this after receiving response with result code 0x10 to `emv_startTransaction`

The tags will be returned in the callback routine.

@param *updatedTLV* TLV stream that can be used to update the following values:

- 9F02: Amount
- 9F03: Other amount
- 9C: Transaction type
- 5F57: Account type

In addition tag DFEE1A can be sent to specify tag list to include in results. Example four tags 9F02, 9F36, 95

@param *ident* Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with `device_getResponseCodeString`

21.10.2.182 `RETURN_CODE IDTechSDK.IDT_NEO2.emv_authenticateTransactionCB (byte[] updatedTLV, CallBackIP callback, string ident = " ")`

Authenticate EMV Transaction Request

Authenticates the EMV transaction for an ICC card. Execute this after receiving response with result code 0x10 to `emv_startTransaction`

The tags will be returned in the callback routine.

@param *updatedTLV* TLV stream that can be used to update the following values:

- 9F02: Amount
- 9F03: Other amount
- 9C: Transaction type
- 5F57: Account type

In addition tag DFEE1A can be sent to specify tag list to include in results. Example four tags 9F02, 9F36, 95

@param *callback* Callback to receive transaction results

@param *ident* Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with `device_getResponseCodeString`

21.10.2.183 `static void IDTechSDK.IDT_NEO2.emv_autoAuthenticate (bool authenticate, string ident = " ") [static]`

Enables authenticate for EMV transactions. If a `startEMVTransaction` results in code 0x0010 (start transaction success), then `emv_authenticateEMVTransaction` can automatically execute if parameter is set to TRUE

Parameters

<i>authenticate</i>	TRUE = auto authenticate, FALSE = manually authenticate
---------------------	---

21.10.2.184 `static void IDTechSDK.IDT_NEO2.emv_autoAuthenticateTags (bool authenticate, byte[] tags, string ident = " ")`
`[static]`

Enables authenticate for EMV transactions. If a startEMVTransaction results in code 0x0010 (start transaction success), then emv_authenticateEMVTransaction can automatically execute if parameter is set to TRUE

Parameters

<i>authenticate</i>	TRUE = auto authenticate, FALSE = manually authenticate
<i>tags</i>	Tags to pass during authentication stage;

21.10.2.185 `RETURN_CODE IDTechSDK.IDT_NEO2.emv_callbackResponseKSN (byte[] KSN, string ident = " ")`

Callback Response Get ETC DUKPT key KSN

Provides a status code to device request of DUKPT IK Loaded Status, from emvCallback.callbackType = EMV_CALLBACK_TYPE.EMV_CALLBACK_TYPE_GET_KSN

Parameters

<i>KSN</i>	KSN
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString

21.10.2.186 `RETURN_CODE IDTechSDK.IDT_NEO2.emv_callbackResponseLCD (EMV_LCD_DISPLAY_MODE type, byte selection, string ident = " ")`

Callback Response LCD Display

Provides menu selection responses to the kernel after a callback was received with DeviceState.EMVCallback, and callbackType = EMV_CALLBACK_TYPE.EMV_CALLBACK_TYPE_LCD, and lcd_displayMode = EMV_LCD_DISPLAY_MODE_MENU, EMV_LCD_DISPLAY_MODE_PROMPT, or EMV_LCD_DISPLAY_MODE_LANGUAGE_SELECT

Parameters

<i>type</i>	If Cancel key pressed during menu selection, then value is EMV_LCD_DISPLAY_MODE_CANCEL. Otherwise, value can be EMV_LCD_DISPLAY_MODE_MENU, EMV_LCD_DISPLAY_MODE_PROMPT, or EMV_LCD_DISPLAY_MODE_LANGUAGE_SELECT
<i>selection</i>	If type = EMV_LCD_DISPLAY_MODE_MENU or EMV_LCD_DISPLAY_MODE_LANGUAGE_SELECT, provide the selection ID line number. Otherwise, if type = EMV_LCD_DISPLAY_MODE_PROMPT supply either 0x43 ('C') for Cancel, or 0x45 ('E') for Enter/accept
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with `errorCode.getErrorString()`

21.10.2.187 RETURN_CODE IDTechSDK.IDT_NEO2.emv_callbackResponseMSR (byte[] MSR, string ident = " ")

Callback Response MSR Entry

Provides MSR information to kernel after a callback was received with DeviceState.EMVCallback, and callbackType = EMV_CALLBACK_MSR

Parameters

<i>MSR</i>	Swiped track data
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with `errorCode.getErrorString()`

21.10.2.188 RETURN_CODE IDTechSDK.IDT_NEO2.emv_callbackResponsePIN (EMV_PIN_MODE type, byte[] KSN, byte[] PIN, string ident = " ")

Callback Response PIN Entry

Provides (or cancels) PIN entry information to kernel after a callback was received with DeviceState.EMVCallback, and callbackType = EMV_CALLBACK_TYPE_PINPAD

Parameters

<i>type</i>	If Cancel key pressed during PIN entry, then value is EMV_PIN_MODE_CANCEL. Otherwise, value can be EMV_PIN_MODE_ONLINE_DUKPT, EMV_PIN_MODE_ONLINE_MKSK, or EMV_PIN_MODE_OFFLINE
<i>KSN</i>	If enciphered PIN, this is either the PIN DUKPT Key (EMV_PIN_MODE_ONLINE_DUKPT) or PIN Session Key (EMV_PIN_MODE_ONLINE_MKSK), or PIN Pairing DUKPT key (EMV_PIN_MODE_OFFLINE, string ident = "")
<i>PIN</i>	If enciphered PIN, this is encrypted PIN block. If device does not implement pairing function, this is plaintext PIN
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with `device_getResponseCodeString`

21.10.2.189 RETURN_CODE IDTechSDK.IDT_NEO2.emv_callbackResponsePIN_ETC (EMV_PIN_MODE type, byte[] KSN, byte[] PIN, string ident = " ")

Callback Response PIN Entry for ETC

Provides (or cancels) PIN entry information to kernel after a callback was received with DeviceState.EMVCallback, and callbackType = EMV_CALLBACK_TYPE_PINPAD_ETC

Parameters

<i>type</i>	If Cancel key pressed during PIN entry, then value is EMV_PIN_MODE_CANCEL. If PIN Bypass during PIN entry, then value is EMV_PIN_MODE_BYPASS. Otherwise, value can be EMV_PIN_MODE_ONLINE_DUKPT, EMV_PIN_MODE_ONLINE_MKSK, or EMV_PIN_MODE_OFFLINE
<i>KSN</i>	If enciphered PIN, this is either the PIN DUKPT Key (EMV_PIN_MODE_ONLINE_DUKPT) or PIN Session Key (EMV_PIN_MODE_ONLINE_MKSK), or PIN Pairing DUKPT key (EMV_PIN_MODE_OFFLINE, string ident = "")
<i>PIN</i>	If enciphered PIN, this is encrypted PIN block. If device does not implement pairing function, this is plaintext PIN
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString

21.10.2.190 RETURN_CODE IDTechSDK.IDT_NEO2.emv_cancelTransaction (string *ident* = " ")

Cancel Transaction

Cancels the currently executing EMV or CTLS transaction.

Parameters

<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.
--------------	--

Returns

RETURN_CODE: Values can be parsed with errorCode.getErrorString()

21.10.2.191 RETURN_CODE IDTechSDK.IDT_NEO2.emv_completeTransaction (bool *commError*, byte[] *authCode*, byte[] *iad*, byte[] *tlvScripts*, byte[] *tlv*, string *ident* = " ")

Complete EMV Transaction Request

Completes the EMV transaction for an ICC card when online authorization request is received from emv_↔ authenticateTransaction

The tags will be returned in the callback routine.

Parameters

<i>commError</i>	Communication error with host. Set to TRUE if host was unreachable, or FALSE if host response received. If Communication error, authCode, iad, tlvScripts can be null.
<i>authCode</i>	Authorization code from host. Two bytes. Example 0x3030. (Tag value 8A). Required
<i>iad</i>	Issuer Authentication Data, if any. Example 0x11223344556677883030 (tag value 91).
<i>tlvScripts</i>	71/72 scripts, if any
<i>tlv</i>	Additional TVL data to return with transaction results
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString

NOTE: There are three possible outcomes for Authorization Code: Approval, Refer To Bank, Decline. The kernel maps these three outcomes to valid authorization response codes using tag DFEE1B through 8 bytes: {2 bytes Approval Code}{2 bytes Referral Code}{2 bytes Decline Code}{2 bytes RFU} If your gateway uses "00" for Approval, "01" for Referral, and "05" for Decline, then DFEE1B 08 3030 3031 3035 0000 If you use an authorization code value that is not defined in DFEE1B, the kernel will use the DECLINE value of DFEE1B by default.

21.10.2.192 RETURN_CODE IDTechSDK.IDT_NEO2.emv_completeTransactionCB (bool *commError*, byte[] *authCode*, byte[] *iad*, byte[] *tlvScripts*, byte[] *tlv*, CallbackP *callback*, string *ident* = " ")

Complete EMV Transaction Request

Completes the EMV transaction for an ICC card when online authorization request is received from emv_↵ authenticateTransaction

The tags will be returned in the callback routine.

Parameters

<i>commError</i>	Communication error with host. Set to TRUE if host was unreachable, or FALSE if host response received. If Communication error, authCode, iad, tlvScripts can be null.
<i>authCode</i>	Authorization code from host. Two bytes. Example 0x3030. (Tag value 8A). Required
<i>iad</i>	Issuer Authentication Data, if any. Example 0x11223344556677883030 (tag value 91).
<i>tlvScripts</i>	71/72 scripts, if any
<i>tlv</i>	Additional TVL data to return with transaction results
<i>callback</i>	Callback that will receive transaction results
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString

NOTE: There are three possible outcomes for Authorization Code: Approval, Refer To Bank, Decline. The kernel maps these three outcomes to valid authorization response codes using tag DFEE1B through 8 bytes: {2 bytes Approval Code}{2 bytes Referral Code}{2 bytes Decline Code}{2 bytes RFU} If your gateway uses "00" for Approval, "01" for Referral, and "05" for Decline, then DFEE1B 08 3030 3031 3035 0000 If you use an authorization code value that is not defined in DFEE1B, the kernel will use the DECLINE value of DFEE1B by default.

21.10.2.193 RETURN_CODE IDTechSDK.IDT_NEO2.emv_exchangeCerts (ref byte[] *cert*, ref byte[] *nonce*, ref byte[] *signature*, string *ident* = " ")

Exchange Certificates, Nonces, and Keys

Use this command to send the ETC certificate, nonce, and signature. The return data is the NEO2 certificate, nonce, and signature.

Parameters

<i>cert</i>	SEND: ETC Certificate for signature verification, RECEIVE: NEO2 Certificate for signature verification
<i>nonce</i>	SEND: ETC random nonce, RECEIVE: NEO2 random nonce
<i>signature</i>	SEND: ETC Signature, RECEIVE: NEO2 Signature. Signature of (CertETC_SV NONCE_ETC) with PKCS1-v1_5 padding
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString

21.10.2.194 RETURN_CODE IDTechSDK.IDT_NEO2.emv_generateDUKPT (byte[] *cert*, byte[] *signature*, ref byte[] *key*, string *ident* = " ")

Generate DUKPT IK Using KEK

Use this command to send the encrypted KEK and signature generated by the ETC. NEO2 returns the DUKPT IK in TR-31 format encrypted with the KEK and signature

Parameters

<i>cert</i>	ETC certificate for signature verification
<i>signature</i>	Signature of (KEK NONCE_ETC) with PKCS1-v1_5 padding
<i>key</i>	ASN.1 structure of DUKPT IK used between NEO2 and ETC
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString

21.10.2.195 RETURN_CODE IDTechSDK.IDT_NEO2.emv_getEMVConfigurationCheckValue (ref string *response*, string *ident* = " ")

Polls device for EMV Configuration Check Value

Parameters

<i>response</i>	Response returned of Check Value of Configuration
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with errorCode.getErrorString(string ident = "")

21.10.2.196 RETURN_CODE IDTechSDK.IDT_NEO2.emv_getEMVKernelCheckValue (ref string *response*, string *ident* = " ")

Polls device for EMV Kernel Check Value

Parameters

<i>response</i>	Response returned of Check Value of Kernel
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with errorCode.getErrorString(string ident = "")

21.10.2.197 **RETURN_CODE** IDTechSDK.IDT_NEO2.emv_getEMVKernelVersion (ref string *response*, string *ident* = " ")

Polls device for EMV Kernel Version

Parameters

<i>response</i>	Response returned of Kernel Version
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with `errorCode.getErrorString(string ident = "")`

21.10.2.198 **RETURN_CODE** IDTechSDK.IDT_NEO2.emv_getEMVKernelVersionExt (ref string *response*, string *ident* = " ")

Polls device for Extended EMV Kernel Version

Parameters

<i>response</i>	Response returned of Kernel Version
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with `errorCode.getErrorString(string ident = "")`

21.10.2.199 **RETURN_CODE** IDTechSDK.IDT_NEO2.emv_getTerminalMajorConfiguration (ref int *configuration*, string *ident* = " ")

Get Terminal Major Configuration

Gets the Terminal Data Major Configuration setting

Parameters

<i>configuration</i>	The configuration that is set (1-5, string <i>ident</i> = "")
----------------------	---

Return values

RETURN_CODE	Return codes listed as typedef enum in IDTCommon:RETURN_CODE. Values can be parsed with IDT_NEO2::device_getResponseCodeString :(string <i>ident</i> = "")
--------------------	---

21.10.2.200 **RETURN_CODE** IDTechSDK.IDT_NEO2.emv_removeAllApplicationData (string *ident* = " ")

Remove All Application Data

Removes all the Application Data for EMV Kernel

Parameters

<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.
--------------	--

Returns

RETURN_CODE: Values can be parsed with `errorCode.getErrorString(string ident = "")`

21.10.2.201 RETURN_CODE IDTechSDK.IDT_NEO2.emv_removeAllCAPK (string *ident* = " ")

Remove All Certificate Authority Public Key

Removes all the CAPK for EMV Kernel

Parameters

<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.
--------------	--

Returns

RETURN_CODE: Values can be parsed with `errorCode.getErrorString(string ident = "")`

21.10.2.202 RETURN_CODE IDTechSDK.IDT_NEO2.emv_removeAllCRL (string *ident* = " ")

Remove All Certificate Revocation List Entries

Removes all CRLEntry entries

Parameters

<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.
--------------	--

Returns

RETURN_CODE: Values can be parsed with `errorCode.getErrorString(string ident = "")`

21.10.2.203 RETURN_CODE IDTechSDK.IDT_NEO2.emv_removeApplicationData (byte[] *AID*, string *ident* = " ")

Remove Application Data by AID

Removes the Application Data for EMV Kernel as specified by the AID name passed as a parameter

Parameters

<i>AID</i>	Name of ApplicationID Must be between 5 and 16 bytes
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with `errorCode.getErrorString(string ident = "")`

21.10.2.204 RETURN_CODE IDTechSDK.IDT_NEO2.emv_removeCAPK (byte[] *capk*, string *ident* = " ")

Remove Certificate Authority Public Key

Removes the CAPK for EMV Kernel as specified by the RID/Index

Parameters

<i>capk</i>	6 byte CAPK = 5 bytes RID + 1 byte INDEX
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with `errorCode.getErrorString(string ident = "")`

21.10.2.205 RETURN_CODE IDTechSDK.IDT_NEO2.emv_removeCRL (byte[] *crlList*, string *ident* = " ")

Remove Certificate Revocation List Entries

Removes CRL entries as specified by the RID and Index and serial number passed as 9 bytes

Parameters

<i>crlList</i>	containing the list of CRL to remove: [CRL1][CRL2]...[CRLn] where each [CRL] is 9 bytes: [5 bytes RID][1 byte CAPK Index][3 bytes serial number]
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with `device_getResponseCodeString`

21.10.2.206 RETURN_CODE IDTechSDK.IDT_NEO2.emv_removeTerminalData (string *ident* = " ")

Remove Terminal Data

Removes the Terminal Data

Parameters

<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.
--------------	--

Returns

RETURN_CODE: Values can be parsed with `device_getResponseCodeString`

21.10.2.207 RETURN_CODE IDTechSDK.IDT_NEO2.emv_resetConfigurationGroup (int *group*, string *ident* = " ")

Reset Configuration Group

This command allows resetting a dataset to its default configuration. If the file exists, it will be overwritten. If not, it will be created.

Parameters

<i>group</i>	Configuration Group
--------------	---------------------

Return values

<i>RETURN_CODE</i>	Return codes listed as typedef enum in IDTechCommon:RETURN_CODE. Values can be parsed with IDT_VP8800::device_getResponseCodeString :(string ident = "")
--------------------	---

21.10.2.208 **RETURN_CODE** IDTechSDK.IDT_NEO2.emv_retrieveAIDList (ref byte *response*[[], string *ident* = " ")

Retrieve AID list

Returns all the AID names and their assigned groups installed on the terminal for CONTACT.

Parameters

<i>response</i>	array of 2-tag TLV data objects: DFEE2D (group name) followed by 9F06 (AID, string ident = "")
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with `errorCode.getErrorString(string ident = "")`

21.10.2.209 **RETURN_CODE** IDTechSDK.IDT_NEO2.emv_retrieveApplicationData (byte[] *AID*, ref byte[] *tlv*, string *ident* = " ")

Retrieve Application Data by AID

Retrieves the Application Data for EMV Kernel as specified by the AID name passed as a parameter.

Parameters

<i>AID</i>	Name of ApplicationID. Must be between 5 and 16 bytes
<i>tlv</i>	The TLV elements of the requested AID
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with `errorCode.getErrorString()`

21.10.2.210 **RETURN_CODE** IDTechSDK.IDT_NEO2.emv_retrieveCAPK (byte[] *capk*, ref byte[] *key*, string *ident* = " ")

Retrieve Certificate Authority Public Key

Retrieves the CAPK for EMV Kernel as specified by the RID/Index passed as a parameter.

Parameters

<i>capk</i>	6 bytes CAPK = 5 bytes RID + 1 byte Index
-------------	---

Parameters

<i>key</i>	<p>Response returned as a CAKey format: [5 bytes RID][1 byte Index][1 byte Hash Algorithm][1 byte Encryption Algorithm][20 bytes HashValue][4 bytes Public Key Exponent][2 bytes Modulus Length][Variable bytes Modulus] Where:</p> <ul style="list-style-type: none"> • Hash Algorithm: The only algorithm supported is SHA-1. The value is set to 0x01 • Encryption Algorithm: The encryption algorithm in which this key is used. Currently support only one type: RSA. The value is set to 0x01. • HashValue: Which is calculated using SHA-1 over the following fields: RID & Index & Modulus & Exponent • Public Key Exponent: Actually, the real length of the exponent is either one byte or 3 bytes. It can have two values: 3 (Format is 0x00 00 00 03), or 65537 (Format is 0x00 01 00 01, string ident = "") • Modulus Length: LenL LenH Indicated the length of the next field. • Modulus: This is the modulus field of the public key. Its length is specified in the field above.
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with `errorCode.getErrorString()`

21.10.2.211 RETURN_CODE IDTechSDK.IDT_NEO2.emv_retrieveCAPKList (ref byte[] keys, string ident = " ")

Retrieve the Certificate Authority Public Key list

Returns all the CAPK RID and Index installed on the terminal for EMV Kernel.

Parameters

<i>keys</i>	[key1][key2]...[keyn], each key 6 bytes where key = 5 bytes RID + 1 byte index
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with `errorCode.getErrorString()`

21.10.2.212 RETURN_CODE IDTechSDK.IDT_NEO2.emv_retrieveCRLList (ref byte[] list, string ident = " ")

Retrieve the Certificate Revocation List

Returns the CRL entries on the terminal.

Parameters

<i>list</i>	[CRL1][CRL2]...[CRLn], each CRL 9 bytes where CRL = 5 bytes RID + 1 byte index + 3 bytes serial number
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with `errorCode.getErrorString()`

21.10.2.213 RETURN_CODE IDTechSDK.IDT_NEO2.emv_retrieveTerminalData (ref byte[] *tlv*, string *ident* = " ")

Retrieve Terminal Data

Retrieves the Terminal Data for EMV Kernel.

Parameters

<i>tlv</i>	Response returned as a TLV
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with `errorCode.getErrorString()`

21.10.2.214 RETURN_CODE IDTechSDK.IDT_NEO2.emv_retrieveTransactionResult (byte[] *tags*, ref IDTTransactionData *tlv*, string *ident* = " ")

Retrieve Transaction Results

Retrieves specified EMV tags from the currently executing transaction.

Parameters

<i>tags</i>	Tags to be retrieved. Example 0x9F028A will retrieve tags 9F02 and 8A
<i>tlv</i>	All requested tags returned as unencrypted, encrypted and masked TLV data in IDTTransactionData object
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with `device.getResponseCodeString`

21.10.2.215 RETURN_CODE IDTechSDK.IDT_NEO2.emv_setApplicationData (byte[] *name*, byte[] *tlv*, string *ident* = " ")

Set Application Data by AID

Sets the Application Data as specified by TLV data

Parameters

<i>tlv</i>	Application data in TLV format
------------	--------------------------------

Tags:

- DFEE4B : Partial AID Matching. 01 = allowed, 00 = Disabled. Mandatory for Visa
- DFEE4C : Application Flow, System: Never, User: Mandatory – 0x01 = MasterCard – 0x02 = AMEX – 0x03 = MasterCard w/Strip Application – 0x06 = Visa – 0x0D = Discover – 0x0E = JCB – 0x15 = Reserved – 0x16 = Reserved – 0x17 = Reserved

- DFEE4D = PPSE Disable, Optional
- DFEE2E = Max AID Length, Mandatory if DFEE4B included. Visa must be set to 16
- DFEE2F = Disable System Aid (no effect on User AID). 0x80 = Disable, 0x00 = Enable
- DFEE4F = Interface Support. 01 = CTLS, 02 = Contact. If missing, defaults to CTLS

Parameters

<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.
--------------	--

Returns

RETURN_CODE: Values can be parsed with `errorCode.getErrorString()`

21.10.2.216 RETURN_CODE IDTechSDK.IDT_NEO2.emv_setApplicationData (byte[] *tlv*, string *ident* = " ")

Set Application Data by AID

Sets the Application Data as specified by TLV data

Parameters

t/v	Application data in TLV format The first tag of the TLV data must be the group number (DFEE2D). The second tag of the TLV data must be the AID (9F06) Tag DFEE4F must be included, and it must have a value of 0x02 to be and EMV AID
-----	---

Example valid TLV, for Group #8, AID a0000000045010: "dfee2d01089f0607a0000000045010dfee4b0101dfee2e0110dfee4c0101dfee

Tags:

- DFEE2D : Group Number. Mandatory First Tag. 0 = System, 1-255 = User
- 9F06 : AID. Mandatory Second Tag
- DFEE4B : Partial AID Matching. 01 = allowed, 00 = Disabled. Mandatory for Visa
- DFEE4C : Application Flow, System: Never, User: Mandatory – 0x01 = MasterCard – 0x02 = AMEX – 0x03 = MasterCard w/Strip Application – 0x06 = Visa – 0x0D = Discover – 0x0E = JCB – 0x15 = Reserved – 0x16 = Reserved – 0x17 = Reserved
- DFEE4D = PPSE Disable, Optional
- DFEE2E = Max AID Length, Mandatory if DFEE4B included. Visa must be set to 16
- DFEE2F = Disable System Aid (no effect on User AID). 0x80 = Disable, 0x00 = Enable
- DFEE4F = Interface Support. 01 = CTLS, 02 = Contact. If missing, defaults to CTLS

Parameters

<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.
--------------	--

Returns

RETURN_CODE: Values can be parsed with `errorCode.getErrorString()`

21.10.2.217 RETURN_CODE IDTechSDK.IDT_NEO2.emv_setCAPK (byte[] key, string ident = " ")

Set Certificate Authority Public Key

Sets the CAPK for EMV Kernel as specified by the CAKey structure

Parameters

<i>key</i>	CAKey format: [5 bytes RID][1 byte Index][1 byte Hash Algorithm][1 byte Encryption Algorithm][20 bytes HashValue][4 bytes Public Key Exponent][2 bytes Modulus Length][Variable bytes Modulus] Where: <ul style="list-style-type: none"> • Hash Algorithm: The only algorithm supported is SHA-1. The value is set to 0x01 • Encryption Algorithm: The encryption algorithm in which this key is used. Currently support only one type: RSA. The value is set to 0x01. • HashValue: Which is calculated using SHA-1 over the following fields: RID & Index & Modulus & Exponent • Public Key Exponent: Actually, the real length of the exponent is either one byte or 3 bytes. It can have two values: 3 (Format is 0x00 00 00 03), or 65537 (Format is 0x00 01 00 01, string ident = "") • Modulus Length: LenL LenH Indicated the length of the next field. • Modulus: This is the modulus field of the public key. Its length is specified in the field above.
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with `errorCode.getErrorString()`

21.10.2.218 RETURN_CODE IDTechSDK.IDT_NEO2.emv_setCRL (byte[] list, string ident = " ")

Set Certificate Revocation List

Sets the CRL

Parameters

<i>list</i>	CRL Entries containing the RID, Index, and serial numbers to set [CRL1][CRL2]...[CRLn] where each [CRL] is 9 bytes: [5 bytes RID][1 byte CAPK Index][3 bytes serial number]
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with `errorCode.getErrorString()`

21.10.2.219 RETURN_CODE IDTechSDK.IDT_NEO2.emv_setTerminalData (byte[] tlv, string ident = " ")

Set Terminal Data

Sets the Terminal Data

Parameters

<i>tlv</i>	TerminalData TLV Data
<i>ident</i>	Optional identifier

Return values

<i>RETURN_CODE</i>	Return codes listed as typedef enum in IDTCommon:RETURN_CODE. Values can be parsed with IDT_BTPay::device_getResponseCodeString:()
--------------------	--

21.10.2.220 **RETURN_CODE** IDTechSDK.IDT_NEO2.emv_setTerminalMajorConfiguration (int *configuration*, string *ident* = " ")

Set Terminal Major Configuration

Sets the Terminal Data Major Configuration setting

Parameters

<i>configuration</i>	The configuration to set (1-5, string <i>ident</i> = "")
----------------------	--

Return values

<i>RETURN_CODE</i>	Return codes listed as typedef enum in IDTCommon:RETURN_CODE. Values can be parsed with IDT_NEO2::device_getResponseCodeString:(string ident = "")
--------------------	--

21.10.2.221 **RETURN_CODE** IDTechSDK.IDT_NEO2.emv_startTransaction (double *amount*, double *amtOther*, int *exponent*, int *type*, int *timeout*, byte[] *tags*, bool *forceOnline*, bool *isFastEMV* = false, string *ident* = " ")

Start EMV Transaction Request

Authorizes the EMV transaction for an ICC card

The tags will be returned in the callback routine.

Parameters

<i>amount</i>	Transaction amount value (tag value 9F02, string <i>ident</i> = "")
<i>amtOther</i>	Other amount value, if any (tag value 9F03, string <i>ident</i> = "")
<i>exponent</i>	Number of characters after decimile point
<i>type</i>	Transaction type (tag value 9C).
<i>timeout</i>	Timeout value in seconds.
<i>tags</i>	Any other tags to be included in the request. Passed as TLV Data. Example, tag 9F02 with amount 0x000000000100 would be 0x9F0206000000000100 If tags 9F02 (amount), 9F03 (other amount), or 9C (transaction type) are included, they will take priority over these values supplied as individual parameters to this method.
<i>forceOnline</i>	TRUE = do not allow offline approval, FALSE = allow ICC to approve offline if terminal capable Note: To request tags to be included in default response, use tag DFEE1A, and specify tag list. Example four tags 9F02, 9F36, 95, 9F37 to be included in response = DFEE1A079F029F369f9F37
<i>isFastEMV</i>	If TRUE, it will populate the IDTTransactionData.fastEMV with ASCII data similar to IDTech FastEMV KB output, after performing an auto-authenticate and auto-complete with ResultCode = Could Not Contact Host
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString

21.10.2.222 **RETURN_CODE** IDTechSDK.IDT_NEO2.emv_startTransactionCB (double *amount*, double *amtOther*, int *exponent*, int *type*, int *timeout*, byte[] *tags*, bool *forceOnline*, CallbackP *callback*, bool *isFastEMV* = false, string *ident* = " ")

Start EMV Transaction Request

Authorizes the EMV transaction for an ICC card

The tags will be returned in the specified callback routine.

Parameters

<i>amount</i>	Transaction amount value (tag value 9F02, string ident = "")
<i>amtOther</i>	Other amount value, if any (tag value 9F03, string ident = "")
<i>exponent</i>	Number of characters after decimile point
<i>type</i>	Transaction type (tag value 9C).
<i>timeout</i>	Timeout value in seconds.
<i>tags</i>	Any other tags to be included in the request. Passed as TLV Data. Example, tag 9F02 with amount 0x000000000100 would be 0x9F0206000000000100 If tags 9F02 (amount), 9F03 (other amount), or 9C (transaction type) are included, they will take priority over these values supplied as individual parameters to this method.
<i>forceOnline</i>	TRUE = do not allow offline approval, FALSE = allow ICC to approve offline if terminal capable
<i>callback</i>	Callback for the returned data Note: To request tags to be included in default response, use tag DFEE1A, and specify tag list. Example four tags 9F02, 9F36, 95, 9F37 to be included in response = DFEE1A079F029F369F959F37
<i>isFastEMV</i>	If TRUE, it will populate the IDTTransactionData.fastEMV with ASCII data similar to IDTech FastEMV KB output, after performing an auto-authenticate and auto-complete with ResultCode = Could Not Contact Host
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString

21.10.2.223 **RETURN_CODE** IDTechSDK.IDT_NEO2.emv_trySetTerminalData (byte[] *tlv*, ref byte[] *rejectedTLV*, ref byte[] *convertedTLV*, bool *overwrite* = false, string *ident* = " ")

Try to Set Terminal Data

Attempts to set the Terminal Data. Use this functions to attempt to set the terminal data to the device. This function will allow the cross-device tag compatible writing of IDTech Legacy/New tag definitions

Parameters

<i>tlv</i>	TerminalData TLV Data
<i>rejectedTLV</i>	Contains the tags (if any) that were rejected by the firmware
<i>convertedTLV</i>	Contains the tags (if any) that were converted and accepted by the firmware
<i>overwrite</i>	TRUE = add TLV to existing tags, FALSE = replace existing tags with TLV
<i>ident</i>	Optional identifier

Return values

RETURN_CODE	Return codes listed as typedef enum in IDTCommon:RETURN_CODE. Values can be parsed with IDT_BTPay::device_getResponseCodeString:()
--------------------	--

21.10.2.224 **RETURN_CODE** IDTechSDK.IDT_NEO2.emv_verifyDUKPTLoaded (byte[] *KCV*, string *ident* = " ")

7.3 Verify DUKPT IK Loaded on ETC

Use this command to verify the DUKPT IK is loaded into the ETC. NEO2 is activated and it can request PIN from ETC after this command.

Parameters

<i>KCV</i>	ASN.1 structure of KCV
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString

21.10.2.225 **RETURN_CODE** IDTechSDK.IDT_NEO2.felica_authentication (byte[] *key*, string *ident* = " ")

FeliCa Authentication

Provides a key to be used in a follow up FeliCa Read with MAC (3 blocks max) or Write with MAC (1 block max). This command must be executed before each Read w/MAC or Write w/MAC command

NOTE: The reader must be in Pass Through Mode for FeliCa commands to work.

Parameters

<i>key</i>	16 byte key used for MAC generation of Read or Write with MAC
<i>ip</i>	IP Address of target device (optional, string <i>ident</i> = "")
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with errorCode.getErrorString(string *ident* = "")

21.10.2.226 **RETURN_CODE** IDTechSDK.IDT_NEO2.felica_read (byte[] *serviceCode*, int *numBlocks*, byte[] *blockList*, ref byte[] *blocks*, string *ident* = " ")

FeliCa Read

Reads up to 4 blocks.

NOTE: The reader must be in Pass Through Mode for FeliCa commands to work.

Parameters

<i>serviceCode</i>	Service Code List. Each service code in Service Code List = 2 bytes of data
<i>numBlocks</i>	Number of blocks
<i>blockList</i>	Blocks to read. Maximum 4 block requests
<i>blocks</i>	Blocks read. Each block 16 bytes.
<i>ip</i>	IP Address of target device (optional, string <i>ident</i> = "")
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with `errorCode.getErrorString(string ident = "")`

21.10.2.227 RETURN_CODE IDTechSDK.IDT_NEO2.felica_readWithMac (int *numBlocks*, byte[] *blockList*, ref byte[] *blocks*, string *ident* = " ")

FeliCa Read with MAC Generation

Reads up to 3 blocks with MAC Generation. FeliCa Authentication must be performed first

NOTE: The reader must be in Pass Through Mode for FeliCa commands to work.

Parameters

<i>numBlocks</i>	Number of blocks
<i>blockList</i>	Block to read. Each block in blockList Maximum 3 block requests
<i>blocks</i>	Blocks read. Each block 16 bytes.
<i>ip</i>	IP Address of target device (optional, string ident = "")
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with `errorCode.getErrorString(string ident = "")`

21.10.2.228 RETURN_CODE IDTechSDK.IDT_NEO2.felica_requestService (byte[] *nodeCode*, ref byte[] *response*, string *ident* = " ")

FeliCa Request Service

Perform functions a Felica Request Service

NOTE: The reader must be in Pass Through Mode for FeliCa commands to work.

Parameters

<i>nodeCode</i>	Node Code
<i>response</i>	Response as explained in FeliCA Lite-S User's Manual
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with `errorCode.getErrorString(string ident = "")`

21.10.2.229 RETURN_CODE IDTechSDK.IDT_NEO2.felica_SendCommand (byte[] *command*, ref byte[] *response*, string *ident* = " ")

FeliCa Send Command

Send a Felica Command

Parameters

<i>command</i>	Command data from settlement center to be sent to felica card
<i>response</i>	Response data from felica card.

Returns

RETURN_CODE: Values can be parsed with device_getIDGStatusCodeString(string ident = "")

21.10.2.230 RETURN_CODE IDTechSDK.IDT_NEO2.felica_write (byte[] *serviceCode*, int *blockCount*, byte[] *blockList*, byte[] *data*, ref byte[] *statusFlag*, string *ident* = " ")

FeliCa Write

Writes a block

NOTE: The reader must be in Pass Through Mode for FeliCa commands to work.

Parameters

<i>serviceCode</i>	Service Code list. Each service code must be 2 bytes
<i>blockCount</i>	Block Count
<i>blockList</i>	Block list.
<i>data</i>	Block to write. Must be 16 bytes.
<i>statusFlag</i>	Status flag response as explained in FeliCA Lite-S User's Manual, Section 4.5
<i>ip</i>	IP Address of target device (optional, string ident = "")
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with errorCode.getErrorString(string ident = "")

21.10.2.231 RETURN_CODE IDTechSDK.IDT_NEO2.felica_writeWithMac (int *blockNumber*, byte[] *data*, string *ident* = " ")

FeliCa Write with MAC Generation

Writes a block with MAC Generation. FeliCa Authentication must be performed first

NOTE: The reader must be in Pass Through Mode for FeliCa commands to work.

Parameters

<i>blockNumber</i>	Number of block
<i>data</i>	Block to write. Must be 16 bytes.
<i>ip</i>	IP Address of target device (optional, string ident = "")
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with errorCode.getErrorString(string ident = "")

21.10.2.232 static int IDTechSDK.IDT_NEO2.getCommandTimeout (string *ident* = " ") [static]

Get Command Timeout

Gets the default timeout (in milliseconds) waiting for a blocking command response

Return values

<i>time</i>	Time
-------------	------

21.10.2.233 `static string IDTechSDK.IDT_NEO2.getlastErrorString (string ident = " ") [static]`

Last Error String

Returns the last firmware reported error string to RETURN_CODE_P2_FAILED and RETURN_CODE_P2_COMMAND_MAND_NOT_ALLOWED if error reporting is enabled

Parameters

<i>ip</i>	IP address of device
-----------	----------------------

Return values

<i>string</i>	Last Error String Message
---------------	---------------------------

21.10.2.234 `RETURN_CODE IDTechSDK.IDT_NEO2.icc_exchangeAPDU (string c_APDU, ref byte[] response, string ident = " ")`

Exchange APDU

Sends an APDU packet to the ICC. If successful, response is returned in APDUResult class instance in response parameter.

Parameters

<i>c_APDU</i>	APDU data packet
<i>response</i>	Unencrypted/encrypted parsed APDU response
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with `errorCode.getErrorString(string ident = "")`

21.10.2.235 `RETURN_CODE IDTechSDK.IDT_NEO2.icc_getICCRReaderStatus (ref byte status, string ident = " ")`

Get Reader Status

Returns the reader status

Parameters

<i>status</i>	Pointer that will return with the ICCReaderStatus results. bit 0: 0 = ICC Power Not Ready, 1 = ICC Powered bit 1: 0 = Card not seated, 1 = card seated
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with `device_getResponseCodeString`

21.10.2.236 `RETURN_CODE IDTechSDK.IDT_NEO2.icc_powerOffICC (string ident = " ")`

Power Off ICC

Powers down the ICC

Parameters

<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.
--------------	--

Returns

RETURN_CODE: Values can be parsed with `errorCode.getErrorString(string ident = "")`

If Success, empty If Failure, ASCII encoded data of error string

21.10.2.237 RETURN_CODE IDTechSDK.IDT_NEO2.icc_powerOnICC (ref byte[] *ATR*, byte *interfaces*, string *ident* = " ")

Power On ICC

Power up the currently selected microprocessor card in the ICC reader

Parameters

<i>interfaces</i>	For NEO2 devices allowed interfaces for which to get the ATR. 0x20h = retrieve last ATR received from PICC 0x21h = SAM1 (SRED version only, string ident = "") 0x22h = SAM2 (SRED version only, string ident = "") For other devices interfaces euquals to 0s
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with `errorCode.getErrorString(string ident = "")`

21.10.2.238 void IDTechSDK.IDT_NEO2.ip_autoConnectToSocket ()

Auto Connect To Socket

Instructs SDK to attempt to connect to all the IP addresses as specified in NEO2_Devices.xml. The NEO2_Devices.xml must be located in the same root directory as IDTechSDK.dll. It is used to specify NEO2 devices for USB connectivity and IP connectivity. It is made up of DEVICE entries example as follows:

```
<Device>
  <Product>VP6800</Product>
  <Name>VP6800</Name>
  <VID>0x0acd</VID>
  <PID>0x4460</PID>
  <UsageID>0x0001</UsageID>
  <UsagePage>0xFF00</UsagePage>
  <BaseIP>10.12.34.98</BaseIP>
  <IP_Count>2</IP_Count>
  <IP_Port></IP_Port>
</Device>
```

- BaseIP = IP address of first device to look for
- IP_Count - # of consecutive IP address to search, up to an IP address of xx.xx.xx.255
- IP_Port - Optional. Defaults to port 1025

Multiple device connections can be defined by a single Device entry, if those devices will use consecutive IP addresses, or multiple device connections can be defined by multiple Device entries, each with unique IP.

Every IP connection attempt is reported to a callback as connection successful, or connection could not be established

21.10.2.239 `bool IDTechSDK.IDT_NEO2.ip_connectToSocket (string IP, bool isSecure = false)`

Connect to Socket

Instructs SDK to attempt to use TCP/IP for communication with [IDT_NEO2](#)

Parameters

<i>IP</i>	Valid IP address, with optional port and label. Examples <ul style="list-style-type: none"> • IP Address: "10.12.34.98" • IP Address + Port: "10.12.34.98:1024" • IP Address + Label: "10.12.34.98#Neo Device #1" • IP Address + Port + Label: "10.12.34.98:1025#Neo Device #1"
<i>isSecure</i>	TRUE = Use TLS 1.2

Return values

<i>success</i>	TRUE = device found, FALSE = device not found
----------------	---

NOTE: If the device is found, it will automatically become the currently selected SDK device unless there is another currently connected IP device the SDK is using. If you would like to change between multiple IP connected devices, use `ip_switchToSocket` instead

21.10.2.240 `bool IDTechSDK.IDT_NEO2.ip_firstConnectToSocket (string IP)`

First Connect to Socket

Instructs SDK to attempt to use TCP/IP for communication with [IDT_NEO2](#), without any retries for reboot delay

Parameters

<i>IP</i>	Valid IP address, with optional port and label. Examples <ul style="list-style-type: none"> • IP Address: "10.12.34.98" • IP Address + Port: "10.12.34.98:1024" • IP Address + Label: "10.12.34.98#Neo Device #1" • IP Address + Port + Label: "10.12.34.98:1025#Neo Device #1"
<i>isSecure</i>	TRUE = Use TLS 1.2

Return values

<i>success</i>	TRUE = device found, FALSE = device not found
----------------	---

NOTE: If the device is found, it will automatically become the currently selected SDK device unless there is another currently connected IP device the SDK is using. If you would like to change between multiple IP connected devices, use `ip_switchToSocket` instead

21.10.2.241 `static List<string> IDTechSDK.IDT_NEO2.ip_getSocketList () [static]`

Get IP Socket List

Returns a list of all the established socket connections with the SDK. Each entry is the original full IP address specified.

Return values

<i>list</i>	Socket Connections
-------------	--------------------

21.10.2.242 `static bool IDTechSDK.IDT_NEO2.ip_isConnected (string ip, int attempts = 1, bool isSecure = false) [static]`

Is IP Connected

Validates if IP channel is open and device responsive. If the channel is open and device not responsive, this will close the channel.

Parameters

<i>IP</i>	address to check
<i>attempts</i>	Number of connection attempts to try before returning result
<i>isSecure</i>	TRUE = TLS 1.2 connection

Return values

<i>status</i>	<ul style="list-style-type: none"> • TRUE = channel is open and device is responsive • FALSE = channel is open and device is unresponsive (channel will close, string ident = "") • FALSE = channel is closed
---------------	--

21.10.2.243 `void IDTechSDK.IDT_NEO2.ip_monitorSocketConnectionStatus (bool enable, bool monitorConnect, int interval, int retryCount)`

Monitor Socket Connection Status

Instructs SDK to monitor currently connected devices for network disconnect, and also to automatically re-connect when plugged back into the network. Disconnect monitors is done by pinging the SDK connected stream list on an interval, and if ping unsuccessful, that device stream is closed and removed from SDK device list, followed by a notification of the device disconnect.

If the option to monitor connection, then the NEO2_Devices.xml must contain the auto-connect information for the SDK to run on the specified interval.

Parameters

<i>enable</i>	TRUE = enable polling for device disconnect and optional connect, at specified interval and retryCount
<i>monitorConnect</i>	If TRUE, an auto-connect attempt will be made at the specified interval
<i>interval</i>	Interval, in seconds, to monitor IP. Minimum value 5 seconds
<i>retryCount</i>	Number of retries trying to communicate before determining device not available and issue disconnect notification.

21.10.2.244 **static bool** IDTechSDK.IDT_NEO2.ip_switchToSocket (**string** *IP*) [static]

Switch to Socket

Instructs SDK to direct communication to the specified device (the device must have already established a connection through auto-connect or ip_connectToSocket, string ident = "")

Parameters

<i>IP</i>	Valid established IP address of the existing device. Must match original IP string exactly. Example: Connect to device as 192.168.1.155:50#Device_1. You must use that whole string, not just 192.168.1.155, or 192.168.1.155:50.
-----------	---

21.10.2.245 **RETURN_CODE** IDTechSDK.IDT_NEO2.lcd_addButton (**string** *screenName*, **string** *buttonName*, **byte** *type*, **byte** *alignment*, **UInt16** *xCord*, **UInt16** *yCord*, **string** *label*, **ref** *lcdItem* *returnItem*, **buttonCallback** *callback*, **string** *ident* = " ")

Add Button

Adds a button to a selected screen. Must execute lcd_createScreen first to establish a screen to draw on.

Parameters

<i>screenName</i>	Screen name that will be the target of add button
<i>buttonName</i>	Button name that will be the target of add button
<i>type</i>	Button Type <ul style="list-style-type: none"> • Large = 0x01 • Medium = 0x02 • Invisible = 0x03 (70px by 60 px, string ident = "")
<i>alignment</i>	Position for Button <ul style="list-style-type: none"> • 0 = Display object at the horizon center of specified y, while x ignored • 1 = Display object at specified x and y • 2 = Display object at center of screen, x, y are both ignored • 3 = Display object at left of the screen of specified y, while x ignored • 4 = Display object at right of the screen of specified y, while x ignored
<i>xCord</i>	x-coordinate for Button, range 0-271
<i>yCord</i>	y-coordinate for Button, range 0-479
<i>label</i>	Label to show on the button. Required for Large/Medium buttons. Not used for Invisible buttons.
<i>returnItem</i>	The item includes screen ID, object ID, top-left x-coordinate, top-left y-coordinate, bottom-right x-coordinate, bottom-left y-coordinate, which are all assigned to the created button
<i>callback</i>	buttonCallback to receive button presses. Passing NULL will return button presses to main messageCallback.
<i>ip</i>	Optional: IP address to execute command on (for IP connected devices, string ident = "")
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString

NOTE ON BUTTON PRESS EVENTS: A button press consists of uint16 specifying the screen, and uint16 specifying the button ID. If buttonCallback is used, it will have the following signature:

```
public delegate void buttonCallback(IDT_DEVICE_Types sender, UInt16 screenID, UInt16 itemID, string IP);
```

If buttonCallback is NULL, then the button presses will be passed to the messageCallback, with DeviceState, ButtonEvent, and data = data[0], data[1] = uint16 screenID, and data[2], data[3] = uint16 buttonID

21.10.2.246 RETURN_CODE IDTechSDK.IDT_NEO2.lcd_addEthernet (string *screenName*, string *objectName*, byte *alignment*, UInt16 *xCord*, UInt16 *yCord*, ref lcdItem *returnItem*, string *ip*, string *ident* = " ")

Add Ethernet Settings

Adds an Ethernet settings to a selected screen. Must execute lcd_createScreen first to establish a screen to draw on.

Parameters

<i>screenName</i>	Screen name that will be the target of add widget
<i>objectName</i>	Object name that will be the target of add widget
<i>alignment</i>	Position for widget <ul style="list-style-type: none"> • 0 = Display object at the horizon center of specified y, while x ignored • 1 = Display object at specified x and y • 2 = Display object at center of screen, x, y are both ignored • 3 = Display object at left of the screen of specified y, while x ignored • 4 = Display object at right of the screen of specified y, while x ignored
<i>xCord</i>	x-coordinate for widget, range 0-271
<i>yCord</i>	y-coordinate for widget, range 0-479
<i>returnItem</i>	The item includes screen ID, object ID, top-left x-coordinate, top-left y-coordinate, bottom-right x-coordinate, bottom-left y-coordinate, which are all assigned to the created widget
<i>ip</i>	Optional: IP address to execute command on (for IP connected devices, string ident = "")
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString

Item	Maximum can be created for each screen
Text Area	20
Large Button	8
Medium Button	16
Invisible Button	16
Numeric Entry	1
Ethernet Setting	1
Led widget	1
image	20

21.10.2.247 **RETURN_CODE** IDTechSDK.IDT_NEO2.lcd_addImage (string *screenName*, string *objectName*, byte *alignment*, UInt16 *xCord*, UInt16 *yCord*, string *filename*, ref lcdItem *returnItem*, string *ident* = " ")

Add Image

Adds a image to a selected screen. Must execute lcd_createScreen first to establish a screen to draw on.

Parameters

<i>screenName</i>	Screen name that will be the target of add image
<i>objectName</i>	Object name that will be the target of add image
<i>alignment</i>	Position for Image <ul style="list-style-type: none"> • 0 = Display object at the horizon center of specified y, while x ignored • 1 = Display object at specified x andy • 2 = Display object at center of screen, x, y are both ignored • 3 = Display object at left of the screen of specified y, while x ignored • 4 = Display object at right of the screen of specified y, while x ignored
<i>xCord</i>	x-coordinate for Image, range 0-271
<i>yCord</i>	y-coordinate for Image, range 0-479
<i>filename</i>	Filename of the image. Must be available in device memory.
<i>returnItem</i>	The item includes screen ID, object ID, top-left x-coordinate, top-left y-coordinate, bottom-right x-coordinate, bottom-left y-coordinate, which are all assigned to the created image
<i>ip</i>	Optional: IP address to execute command on (for IP connected devices, string <i>ident</i> = "")
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString

Item	Maximum can be created for each screen
Text Area	20
Large Button	8
Medium Button	16
Invisible Button	16
Numeric Entry	1
Ethernet Setting	1
Led widget	1
image	20

21.10.2.248 **RETURN_CODE** IDTechSDK.IDT_NEO2.lcd_addLED (string *screenName*, string *objectName*, byte *alignment*, UInt16 *xCord*, UInt16 *yCord*, ref lcdItem *returnItem*, byte[] *LED*, string *ident* = " ")

Add LED

Adds a LED widget to a selected screen. Must execute lcd_createScreen first to establish a screen to draw on.

Parameters

<i>screenName</i>	Screen name that will be the target of add LED
<i>objectName</i>	Object name that will be the target of add LED

Parameters

<i>alignment</i>	Position for LED <ul style="list-style-type: none"> • 0 = Display object at the horizon center of specified y, while x ignored • 1 = Display object at specified x and y • 2 = Display object at center of screen, x, y are both ignored • 3 = Display object at left of the screen of specified y, while x ignored • 4 = Display object at right of the screen of specified y, while x ignored
<i>xCord</i>	x-coordinate for LED, range 0-271
<i>yCord</i>	y-coordinate for LED, range 0-479
<i>returnItem</i>	The item includes screen ID, object ID, top-left x-coordinate, top-left y-coordinate, bottom-right x-coordinate, bottom-left y-coordinate, which are all assigned to the created widget
<i>LED</i>	Must be 4 bytes, LED 0 = byte 0, LED 1 = byte 1, LED 2 = byte 2, LED 3 = byte 3 <ul style="list-style-type: none"> • Value 0 = LED OFF • Value 1 = LED Green • Value 2 = LED Yellow • Value 3 = LED Red
<i>ip</i>	Optional: IP address to execute command on (for IP connected devices, string ident = "")
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString

Item	Maximum can be created for each screen
Text Area	20
Large Button	8
Medium Button	16
Invisible Button	16
Numeric Entry	1
Ethernet Setting	1
Led widget	1
image	20

21.10.2.249 RETURN_CODE IDTechSDK.IDT_NEO2.lcd_addText (string *screenName*, string *objectName*, byte *alignment*, UInt16 *xCord*, UInt16 *yCord*, UInt16 *width*, UInt16 *height*, byte *fontID*, byte[] *color*, string *label*, ref lcdItem *returnItem*, string *ident* = " ")

Add text

Adds a text component to a selected screen. Must execute lcd_createScreen first to establish a screen to draw on.

Parameters

<i>screenName</i>	Screen name that will be the target of add text
<i>objectName</i>	Object name that will be the target of add text

Parameters

<i>alignment</i>	Position for Text <ul style="list-style-type: none"> • 0 = Display object at the horizon center of specified y, while x ignored • 1 = Display object at specified x andy • 2 = Display object at center of screen, x, y are both ignored • 3 = Display object at left of the screen of specified y, while x ignored • 4 = Display object at right of the screen of specified y, while x ignored
<i>xCord</i>	x-coordinate for Text, range 0-271
<i>yCord</i>	y-coordinate for Text, range 0-479
<i>width</i>	Width of text area
<i>height</i>	Height of text area
<i>fontID</i>	Font ID
<i>color</i>	Four bytes for color, example, Blue = 0xFF000000, Black = 0x00000000 <ul style="list-style-type: none"> • Byte 0 = B • Byte 1 = G • Byte 2 = R • Byte 3 = Reserved
<i>label</i>	Label to show on the text
<i>returnItem</i>	The item includes screen ID, object ID, top-left x-coordinate, top-left y-coordinate, bottom-right x-coordinate, bottom-left y-coordinate, which are all assigned to the created text area
<i>ip</i>	Optional: IP address to execute command on (for IP connected devices, string ident = "")
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString

Font ID	Typography Name	Font	Size
0	RoundBold_12	RoundBold.ttf	12
1	RoundBold_18	RoundBold.ttf	18
2	RoundBold_24	RoundBold.ttf	24
3	RoundBold_36	RoundBold.ttf	36
4	RoundBold_48	RoundBold.ttf	48
5	RoundBold_60	RoundBold.ttf	60
6	RoundBold_72	RoundBold.ttf	72
7	RoundCondensedBold_12	RoundCondensedBold.ttf	12
8	RoundCondensedBold_18	RoundCondensedBold.ttf	18
9	RoundCondensedBold_24	RoundCondensedBold.ttf	24
10	RoundCondensedBold_36	RoundCondensedBold.ttf	36
11	RoundCondensedBold_48	RoundCondensedBold.ttf	48
12	RoundCondensedBold_60	RoundCondensedBold.ttf	60
13	RoundCondensedBold_72	RoundCondensedBold.ttf	72
14	RoundCondensedMedium_12	RoundCondensedMedium_↔ 0.ttf	12
15	RoundCondensedMedium_18	RoundCondensedMedium_↔ 0.ttf	18

Font ID	Typography Name	Font	Size
16	RoundCondensedMedium_24	RoundCondensedMedium_↔ 0.ttf	24
17	RoundCondensedMedium_36	RoundCondensedMedium_↔ 0.ttf	36
18	RoundCondensedMedium_48	RoundCondensedMedium_↔ 0.ttf	48
19	RoundCondensedMedium_60	RoundCondensedMedium_↔ 0.ttf	60
20	RoundCondensedMedium_72	RoundCondensedMedium_↔ 0.ttf	72
21	RoundCondensedSemibold_12	RoundCondensedSemibold.ttf	12
22	RoundCondensedSemibold_18	RoundCondensedSemibold.ttf	18
23	RoundCondensedSemibold_24	RoundCondensedSemibold.ttf	24
24	RoundCondensedSemibold_36	RoundCondensedSemibold.ttf	36
25	RoundCondensedSemibold_48	RoundCondensedSemibold.ttf	48
26	RoundCondensedSemibold_60	RoundCondensedSemibold.ttf	60
27	RoundCondensedSemibold_72	RoundCondensedSemibold.ttf	72
28	RoundMedium_12	RoundMedium.ttf	12
29	RoundMedium_18	RoundMedium.ttf	18
30	RoundMedium_24	RoundMedium.ttf	24
31	RoundMedium_36	RoundMedium.ttf	36
32	RoundMedium_48	RoundMedium.ttf	48
33	RoundMedium_60	RoundMedium.ttf	60
34	RoundMedium_72	RoundMedium.ttf	72
35	RoundSemibold_12	RoundSemibold.ttf	12
36	RoundSemibold_18	RoundSemibold.ttf	18
37	RoundSemibold_24	RoundSemibold.ttf	24
38	RoundSemibold_36	RoundSemibold.ttf	36
39	RoundSemibold_48	RoundSemibold.ttf	48
40	RoundSemibold_60	RoundSemibold.ttf	60
41	RoundSemibold_72	RoundSemibold.ttf	72

Item	Maximum can be created for each screen
Text Area	20
Large Button	8
Medium Button	16
Invisible Button	16
Numeric Entry	1
Ethernet Setting	1
Led widget	1
image	20

21.10.2.250 RETURN_CODE IDTechSDK.IDT_NEO2.lcd_clearAllLines (string *ident* = " ")

Clear LCD Display

Clears all lines of the LCD Display.

Parameters

<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.
--------------	--

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString

21.10.2.251 **static RETURN_CODE IDTechSDK.IDT_NEO2.lcd_clearDisplay (string *ident* = " ") [static]**

Clear Display

Command to clear the display screen on the reader. It returns the display to the currently defined background color and terminates all events

Parameters

<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.
--------------	--

Returns

RETURN_CODE: Values can be parsed with errorCode.getErrorString(string ident = "")

21.10.2.252 **RETURN_CODE IDTechSDK.IDT_NEO2.lcd_clearScreenInfo (string *ident* = " ")**

Clear Screen Info

Clear all current screen information in RAM and flash. And then show 'power-on screen'

Parameters

<i>ip</i>	Optional: IP address to execute command on (for IP connected devices, string ident = "")
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString

21.10.2.253 **RETURN_CODE IDTechSDK.IDT_NEO2.lcd_cloneScreen (string *screenName*, string *cloneName*, ref UInt16 *cloneID*, string *ident* = " ")**

Clone Screen

Clones an existing screen.

Parameters

<i>screenID</i>	Screen number to clone.
<i>cloneID</i>	Screen ID of the cloned screen
<i>ip</i>	Optional: IP address to execute command on (for IP connected devices, string ident = "")
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString

21.10.2.254 **RETURN_CODE** IDTechSDK.IDT_NEO2.lcd_createScreen (string *screenName*, ref UInt16 *screenID*, string *ident* = " ")

Create Screen

Creates a new screen.

Parameters

<i>screenName</i>	Screen name to use.
<i>screenID</i>	Screen ID that was created.
<i>ip</i>	Optional: IP address to execute command on (for IP connected devices, string <i>ident</i> = "")
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString

21.10.2.255 **RETURN_CODE** IDTechSDK.IDT_NEO2.lcd_destroyScreen (string *screenName*, string *ident* = " ")

Destroy Screen

Destroys a previously created inactive screen. The screen cannot be active

Parameters

<i>screenName</i>	Screen name to destroy. The screen number is assigned with lcd_createScreen.
<i>ip</i>	Optional: IP address to execute command on (for IP connected devices, string <i>ident</i> = "")
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString

21.10.2.256 **RETURN_CODE** IDTechSDK.IDT_NEO2.lcd_displayMessage (int *lineNumber*, string *message*, string *ident* = " ")

Display Message on Line

Displays a message on a display line. 16 characters per line. You can display up to 32 characters for line 1, and it will flow to line 2, or you can display up to 16 characters on just line 2

Parameters

<i>lineNumber</i>	Line number to display message on (1 or 2, string <i>ident</i> = "")
<i>message</i>	Message to display
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString

21.10.2.257 **RETURN_CODE** IDTechSDK.IDT_NEO2.lcd_getActiveScreen (ref string *screenName*, string *ident* = " ")

Get Active Screen

Returns the active screen ID.

Parameters

<i>screenName</i>	Screen name this is active.
<i>ip</i>	Optional: IP address to execute command on (for IP connected devices, string <i>ident</i> = "")
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString

21.10.2.258 **RETURN_CODE** IDTechSDK.IDT_NEO2.lcd_getAllObjects (string *screenName*, ref byte *objectNumbers*, ref Dictionary< UInt16, string > *returnObjects*, string *ident* = " ")

Get All Objects on Screen

Get all created objects' name on certain screen

Parameters

<i>objectNumbers</i>	Number of created objects
<i>returnObjects</i>	Dictionary all created objects -key ID of a created screen -value Name of a created screen
<i>ip</i>	Optional: IP address to execute command on (for IP connected devices, string <i>ident</i> = "")
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString

21.10.2.259 **RETURN_CODE** IDTechSDK.IDT_NEO2.lcd_getAllScreens (ref byte *screenNumbers*, ref Dictionary< UInt16, string > *returnScreens*, string *ident* = " ")

Get All Screens

Get all created screens' name

Parameters

<i>screenNumbers</i>	Number of created screens
<i>returnScreens</i>	Dictionary all created screens -key ID of a created screen -value Name of a created screen
<i>ip</i>	Optional: IP address to execute command on (for IP connected devices, string <i>ident</i> = "")
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString

21.10.2.260 **RETURN_CODE** IDTechSDK.IDT_NEO2.Icd_getAudioVolume (ref int *volume*, string *ident* = " ")

Get Audio Volume

Returns playback volume as represented by an integer

Parameters

<i>volume</i>	Value 0-20, where 0 is silent and 20 is full volume
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with `errorCode.getErrorString()`

21.10.2.261 **RETURN_CODE** IDTechSDK.IDT_NEO2.Icd_getButtonEvent (ref UInt16 *screenID*, ref UInt16 *objectID*, ref string *screenName*, ref string *objectName*, ref bool *isLongPress*, string *ident* = " ")

Get Button Event

Reports back the ID of the button that encountered a click event after the last Get Button Event.

Parameters

<i>screenID</i>	Screen ID of the last clicked button
<i>objectID</i>	Button ID of the last clicked button
<i>screenName</i>	Screen Name of the last clicked button
<i>objectName</i>	Button Name of the last clicked button
<i>isLongPress</i>	TRUE = Long Press, FALSE = Short Press
<i>ip</i>	Optional: IP address to execute command on (for IP connected devices, string <i>ident</i> = "")
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with `device.getResponseCodeString`

21.10.2.262 **RETURN_CODE** IDTechSDK.IDT_NEO2.Icd_linkUIWithTransactionMessageId (byte *messageID*, string *screenName*, string *ident* = " ")

Link UI with Transaction Message ID

Link an existing Customer UI ID with a specified transaction message ID. During transaction, the linked System UI will be replaced by the linked Customer UI

Parameters

<i>messageID</i>	Transaction Message ID: Refer to Doc "EMV Display Message ID Assignment" Selection by Customer
<i>screenName</i>	Name of the screen (No longer than 31 characters)

Returns

RETURN_CODE: Values can be parsed with device_getIDGStatusCodeString()

21.10.2.263 RETURN_CODE IDTechSDK.IDT_NEO2.lcd_loadScreenInfo (string *ident* = " ")

Load Screen Info

Load all current screen information from RAM to flash

Parameters

<i>ip</i>	Optional: IP address to execute command on (for IP connected devices, string <i>ident</i> = "")
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString

21.10.2.264 RETURN_CODE IDTechSDK.IDT_NEO2.lcd_playAudio (string *name*, int *type*, string *ident* = " ")

Play Audio

This command plays an audio file loaded from the inserted SD card. The VP6800 supports 16bit PCM format .WAV files.

Parameters

<i>name</i>	Name of file on SD Card
<i>type</i>	0 = Flash, 1 = SD Card
<i>ip</i>	Optional IP Address
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with errorCode.getErrorString()

21.10.2.265 RETURN_CODE IDTechSDK.IDT_NEO2.lcd_queryObjectbyID (UInt16 *objectID*, ref byte *objectNumbers*, ref List< string > *returnItems*, string *ident* = " ")

Queery Object by ID

Check if the given object exists or not. If exists, return all screen names which contains the object of the given object ID

Parameters

<i>objectID</i>	ID of the checked object
<i>objectNumber</i>	Number of the checked object
<i>returnItems</i>	screens containing the item
<i>ip</i>	Optional: IP address to execute command on (for IP connected devices, string <i>ident</i> = "")
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString

21.10.2.266 RETURN_CODE IDTechSDK.IDT_NEO2.lcd_queryObjectbyName (string *objectName*, ref byte *objectNumbers*, ref List< string > *returnItems*, string *ident* = " ")

Query Object by Name

Check if the given object exists or not. If exists, return all screen names which contains the object of the given object name

Parameters

<i>objectName</i>	Name of the checked object
<i>objectNumber</i>	Number of the checked object
<i>returnItems</i>	List of all the screens that contain the object
<i>ip</i>	Optional: IP address to execute command on (for IP connected devices, string <i>ident</i> = "")
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString

21.10.2.267 RETURN_CODE IDTechSDK.IDT_NEO2.lcd_queryScreenbyID (UInt16 *screenID*, ref byte *result*, ref string *screenName*, string *ident* = " ")

Query Screen by ID

Check if the given screen exists or not

Parameters

<i>screenID</i>	ID of the checked screen
<i>result</i>	the checking result
<i>screenName</i>	Name of the checked screen
<i>ip</i>	Optional: IP address to execute command on (for IP connected devices, string <i>ident</i> = "")
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString

21.10.2.268 RETURN_CODE IDTechSDK.IDT_NEO2.lcd_queryScreenbyName (string *screenName*, ref byte *result*, string *ident* = " ")

Query Screen by Name

Check if the given screen exists or not

Parameters

<i>screenName</i>	Name of the checked screen
-------------------	----------------------------

Parameters

<i>result</i>	the checking result
<i>ip</i>	Optional: IP address to execute command on (for IP connected devices, string <i>ident</i> = "")
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with `device_getResponseCodeString`

21.10.2.269 `RETURN_CODE IDTechSDK.IDT_NEO2.lcd_removeItem (string screenName, string objectName, string ident = " ")`

Removed Item

Removes a component.

Parameters

<i>screenName</i>	Screen name where to remove the target from.
<i>objectName</i>	Identifier of the component
<i>ip</i>	Optional: IP address to execute command on (for IP connected devices, string <i>ident</i> = "")
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with `device_getResponseCodeString`

21.10.2.270 `static void IDTechSDK.IDT_NEO2.lcd_retrieveMessage (DisplayMessages.DISPLAY_MESSAGE_LANGUAGE lang, DisplayMessages.DISPLAY_MESSAGE_IDENTIFIER id, ref string line1, ref string line2) [static]`

Retrieve LCD Message

Returns the string value for a message ID returned for LCD messaging

Parameters

<i>lang</i>	Language.
<i>id</i>	Message ID
<i>line1</i>	Line 1 string value
<i>line2</i>	Line 2 string value
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with `device_getResponseCodeString`

21.10.2.271 `RETURN_CODE IDTechSDK.IDT_NEO2.lcd_setAudioVolume (int volume, string ident = " ")`

Set Audio Volume

Sets the audio playback volume

Parameters

<i>volume</i>	Value 0-20, where 0 is silent and 20 is full volume
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with `errorCode.getErrorString()`

21.10.2.272 `RETURN_CODE IDTechSDK.IDT_NEO2.lcd_setBacklight (bool isBacklightOn, byte backlightVal, string ident = " ")`

Set Backlight

Set backlight percentage. If the percent >100, it will be rejected. If percent < 10, backlight percent will be set to 10

Parameters

<i>isBacklightOn</i>	True = ON, False = Off
<i>backlightVal</i>	Backlight percent value to be set
<i>ip</i>	Optional: IP address to execute command on (for IP connected devices, string <i>ident</i> = "")
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with `device_getResponseCodeString`

21.10.2.273 `void IDTechSDK.IDT_NEO2.lcd_setButtonCallback (string screenName, string buttonName, buttonCallback callback, string ip, string ident = " ")`

Set Button Callback

Sets the callback for a button.

Parameters

<i>screenName</i>	Screen name where the button is located
<i>buttonName</i>	Object name of the button
<i>callback</i>	buttonCallback to receive button presses. Passing NULL will return button presses to main messageCallback.
<i>ip</i>	Optional: IP address to execute command on (for IP connected devices, string <i>ident</i> = "")

NOTE ON BUTTON PRESS EVENTS: A button press consists of uint16 specifying the screen, and uint16 specifying the button ID. If buttonCallback is used, it will have the following signature:

```
public delegate void buttonCallback(IDT_DEVICE_Types sender, UInt16 screenID, UInt16 itemID, string IP);
```

If buttonCallback is NULL, then the button presses will be passed to the messageCallback, with DeviceState, ButtonEvent, and data = data[0], data[1] = uint16 screenID, and data[2], data[3] = uint16 buttonID

21.10.2.274 `void IDTechSDK.IDT_NEO2.lcd_setPinCancelPromptCallback (CancelPromptCallback callback, string ident = " ")`

Set Pin Cancel Prompt Callback

Sets the callback for the PIN Cancel Prompt Callback.

Parameters

<i>callback</i>	CancelPromptCallback
<i>ip</i>	Optional: IP address to execute command on (for IP connected devices, string ident = "")

Callback will have the follow signature:

```
public delegate void CancelPromptCallback(string ipAddress, ident);
```

21.10.2.275 void IDTechSDK.IDT_NEO2.lcd_setPinFailureCallback (FailureCallback *callback*, string *ident* = " ")

Set Pin Failure Callback

Sets the callback for the PIN Failure Callback.

Parameters

<i>callback</i>	FailureCallback
<i>ip</i>	Optional: IP address to execute command on (for IP connected devices, string ident = "")

Callback will have the follow signature:

```
public delegate void FailureCallback(string ipAddress, RETURN_CODE errorCode, ident);
```

21.10.2.276 void IDTechSDK.IDT_NEO2.lcd_setPinInputCallback (SwipeCallback *callback*, string *ident* = " ")

Set Pin Input Callback

Sets the callback for the PIN Input Callback.

Parameters

<i>callback</i>	SwipeCallback
<i>ip</i>	Optional: IP address to execute command on (for IP connected devices, string ident = "")

Callback will have the follow signature:

```
public delegate void SwipeCallback(string ipAddress, string input, ident);
```

21.10.2.277 void IDTechSDK.IDT_NEO2.lcd_setPinSwipeCallback (SwipeCallback *callback*, string *ident* = " ")

Set Pin Swipe Callback

Sets the callback for the PIN Swipe Callback.

Parameters

<i>callback</i>	SwipeCallback
<i>ip</i>	Optional: IP address to execute command on (for IP connected devices, string ident = "")

Callback will have the follow signature:

```
public delegate void InputCallback(string ipAddress, IDTTTransactionData input, ident);
```

21.10.2.278 void IDTechSDK.IDT_NEO2.lcd_setPinTimeoutCallback (TimeoutCallback *callback*, string *ident* = " ")

Set Pin Timeout Callback

Sets the callback for the PIN Timeout Callback.

Parameters

<i>callback</i>	TimeoutCallback
<i>ip</i>	Optional: IP address to execute command on (for IP connected devices, string <i>ident</i> = "")

Callback will have the follow signature:

```
public delegate void TimeoutCallback(string ipAddress, ident);
```

21.10.2.279 RETURN_CODE IDTechSDK.IDT_NEO2.lcd_showScreen (string *screenName*, string *ident* = " ")

Show Screen

Displays and makes active a previously created screen.

Parameters

<i>screenName</i>	Screen to display. The screen name is defined with <code>lcd_createScreen</code> .
<i>ip</i>	Optional: IP address to execute command on (for IP connected devices, string <i>ident</i> = "")
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with `device_getResponseCodeString`

21.10.2.280 RETURN_CODE IDTechSDK.IDT_NEO2.lcd_startCameraCapture (ushort *timeout*, string *ident* = " ")

Start Camera Capture

When the camera is turned on, a "take a picture" button appears on the display. If the button is pressed, the device takes a picture, which is stored in SD card and displayed on-screen for five seconds. The device stores a maximum of 20 pictures. Upon reaching capacity, the 21st picture replaces the 1st.

Parameters

<i>ip</i>	timeout Timeout value. Minimum is 30 seconds (values under 30 seconds will default to 30 seconds, string <i>ident</i> = "")
<i>ip</i>	Optional IP Address
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with `errorCode.getErrorString()`

21.10.2.281 RETURN_CODE IDTechSDK.IDT_NEO2.lcd_startScanQR (ushort *timeout*, string *ident* = " ")

Start QR Scanning

This command will enable the camera and attempt to scan a QR code. The data will be returned to the Message↔ Callback, with DeviceState.CameraEventData

Parameters

<i>timeout</i>	Timeout value. Passing a value of 0 defaults to 30 seconds
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with `errorCode.getErrorString()`

21.10.2.282 RETURN_CODE IDTechSDK.IDT_NEO2.lcd_startScanQR_ext (ushort *timeout*, ushort *xcord*, ushort *ycord*, ushort *width*, ushort *height*, string *ident* = " ")

Start QR Scanning Extended

This command will enable the camera and attempt to scan a QR code. The data will be returned to the Message↔ Callback, with DeviceState.CameraEventData.

Parameters

<i>timeout</i>	Timeout value. Passing a value of 0 defaults to 30 seconds
<i>xcord</i>	Display Window X Coordinate 0-271
<i>ycord</i>	Display Window Y Coordinate 0-479
<i>width</i>	Display Window Width 0-271
<i>height</i>	Display Window Height 0-479
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with `errorCode.getErrorString()`

21.10.2.283 RETURN_CODE IDTechSDK.IDT_NEO2.lcd_startScreenSaver (string *name*, string *ident* = " ")

Start Screen Saver

The command starts the screen saver, which is disabled when the touchpad is touched, the customize screen is shown, or a transaction is started. The device reads the video from the inserted SD Card; the video format must be MJPEG with a maximum frame width of 480px and maximum frame height of 272px. Note that the video displays rotated +90 degrees.

Parameters

<i>name</i>	Name of file on SD Card
<i>ip</i>	Optional IP Address
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with `errorCode.getErrorString()`

21.10.2.284 RETURN_CODE IDTechSDK.IDT_NEO2.lcd_stopAudio (string *ident* = " ")

Stop Audio

This command stop playing audio started with lcd_playAudio.

Parameters

<i>ip</i>	Optional IP Address
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with errorCode.getErrorString()

21.10.2.285 RETURN_CODE IDTechSDK.IDT_NEO2.lcd_stopCameraCapture (string *ident* = " ")

Stop Camera Capture

This command will stop the camera that started with lcd_startCameraCapture

Parameters

<i>ip</i>	Optional IP Address
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with errorCode.getErrorString()

21.10.2.286 RETURN_CODE IDTechSDK.IDT_NEO2.lcd_stopScanQR (string *ident* = " ")

Stop QR Scanning

This command will stop QR scanning that started with lcd_startScanQR

Parameters

<i>ip</i>	Optional IP Address
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with errorCode.getErrorString()

21.10.2.287 RETURN_CODE IDTechSDK.IDT_NEO2.lcd_storeScreenInfo (string *ident* = " ")

Store Screen Info

Store all current screen information from RAM to flash

Parameters

<i>ip</i>	Optional: IP address to execute command on (for IP connected devices, string ident = "")
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString

21.10.2.288 RETURN_CODE IDTechSDK.IDT_NEO2.lcd_updateColor (string *screenName*, string *objectName*, byte[] *color*, string *ident* = " ")

Update Color

Updates the component color, or updates the LED colors if specifying LCD component

Parameters

<i>screenName</i>	Screen name that will be the target of update color
<i>objecName</i>	Identifier of the component
<i>color</i>	<p>Non LCD Widget: Four bytes for color, example, Blue = 0xFF000000, Black = 0x00000000</p> <ul style="list-style-type: none"> • Byte 0 = B • Byte 1 = G • Byte 2 = R • Byte 3 = Reserved LCD Widget: Four bytes for LED color, byte 0 = LED 0, byte 1 = LED 1, byte 2 = LED2, byte 3 = LED 3 • Value 0 = LED OFF • Value 1 = LED Green • Value 2 = LED Yellow • Value 3 = LED Red
<i>ip</i>	Optional: IP address to execute command on (for IP connected devices, string ident = "")
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString

21.10.2.289 RETURN_CODE IDTechSDK.IDT_NEO2.lcd_updateLabel (string *screenName*, string *objectName*, string *label*, string *ident* = " ")

Update Label

Updates the component label.

Parameters

<i>screenName</i>	Screen name that will be the target of update label
-------------------	---

Parameters

<i>objectName</i>	Identifier of the component
<i>label</i>	Label to show on the component
<i>ip</i>	Optional: IP address to execute command on (for IP connected devices, string ident = "")
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString

21.10.2.290 RETURN_CODE IDTechSDK.IDT_NEO2.lcd_updatePosition (string *screenName*, string *objectName*, byte *alignment*, UInt16 *new_xCord*, UInt16 *new_yCord*, string *ident* = " ")

Update Position

Updates the component position.

Parameters

<i>screenName</i>	Screen Name that will be the target of update position
<i>objectName</i>	Identifier of the component
<i>alignment</i>	Alignment for the target <ul style="list-style-type: none"> • 0 = Display object at the horizon center of specified y, while x ignored • 1 = Display object at specified x and y • 2 = Display object at center of screen, x, y are both ignored • 3 = Display object at left of the screen of specified y, while x ignored • 4 = Display object at right of the screen of specified y, while x ignored
<i>new_xCord</i>	x-coordinate for Text, range 0-271
<i>new_yCord</i>	y-coordinate for Text, range 0-479
<i>ip</i>	Optional: IP address to execute command on (for IP connected devices, string ident = "")
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString

21.10.2.291 static void IDTechSDK.IDT_NEO2.monitorNetworkForDevices (bool *monitorON*, int *port* = 0, bool *authClient* = false, int *sendTimeout* = 6000, int *receiveTimeout* = 6000) [static]

Monitor for Network TLS Devices

Instructs SDK to attempt to automatically monitor and establish connections with incoming TCP/IP Devices

Parameters

<i>monitorON</i>	TRUE = enable monitoring, FALSE = disable monitoring
<i>port</i>	Can override the default port if necessary. TLS default = 1443, non-TLS = 1025
<i>authClient</i>	TRUE = Authorize client certificates

Parameters

<i>sendTimeout</i>	Timeout value for sending packets, in milliseconds. Default 6000ms
<i>receive</i>	Timeout value for receiving packets, in milliseconds. Default 6000ms

NOTE: The devices will report to the main callback connect/disconnect events

21.10.2.292 **RETURN_CODE** IDTechSDK.IDT_NEO2.msr_cancelMSRSwipe (string *ident* = " ")

Disable MSR Swipe Cancels MSR swipe request.

Parameters

<i>ip</i>	Optional: IP address to execute command on (for IP connected devices, string <i>ident</i> = "")
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with `errorCode.getErrorString(string ident = "")`

21.10.2.293 **RETURN_CODE** IDTechSDK.IDT_NEO2.msr_getConfiguration (ref byte[] *config*, string *ident* = " ")

Set MSR Configuration

Gets msr configuration data.

Parameters

<i>config</i>	Configuration data to get
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with `errorCode.getErrorString(string ident = "")`

21.10.2.294 **RETURN_CODE** IDTechSDK.IDT_NEO2.msr_getMSRTrack (ref int *val*, string *ident* = " ")

Gets the MSR tracks setting.

Parameters

<i>val</i>	Track Value: <ul style="list-style-type: none"> • 0 = Any Track • 1 = Track 1 • 2 = Track 2 • 3 = Track 1, Track 2 • 4 = Track 3 • 5 = Track 1, Track 3 • 6 = Track 2, Track 3 • 7 = Track 1, Track 2, Track 3
<i>ip</i>	Optional IP parameter
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString

21.10.2.295 RETURN_CODE IDTechSDK.IDT_NEO2.msr_setConfiguration (byte[] *config*, string *ident* = " ")

Set MSR Configuration

Sets msr configuration data.

Parameters

<i>config</i>	Configuration data to send
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with errorCode.getErrorString(string ident = "")

21.10.2.296 RETURN_CODE IDTechSDK.IDT_NEO2.msr_setMSRTrack (int *val*, string *ip*, string *ident* = " ")

Sets the MSR tracks to recognize. When "Any Track" is selected (0 - default value), an error condition will be returned if no tracks are read. When specific tracks are selected (1 - 7), if any of the requested tracks are missing, it will return error.

Parameters

<i>val</i>	Track Value: <ul style="list-style-type: none"> • 0 = Any Track • 1 = Track 1 • 2 = Track 2 • 3 = Track 1, Track 2 • 4 = Track 3 • 5 = Track 1, Track 3 • 6 = Track 2, Track 3 • 7 = Track 1, Track 2, Track 3
<i>ip</i>	Optional IP parameter
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with `device_getResponseCodeString`

21.10.2.297 RETURN_CODE IDTechSDK.IDT_NEO2.msr_startMSRSwipe (int *timeout*, string *ident* = " ")

Enable MSR Swipe

Enables MSR, waiting for swipe to occur.

Parameters

<i>timeout</i>	Swipe Timeout Value
<i>ip</i>	Optional: IP address to execute command on (for IP connected devices, string <i>ident</i> = "")
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with `errorCode.getErrorString(string ident = "")`

21.10.2.298 RETURN_CODE IDTechSDK.IDT_NEO2.msr_startMSRSwipe_ext (int *timeout*, string *ident*, SwipeCallback *swipeCallback*, TimeoutCallback *timeoutCallback*, FailureCallback *failureCallback*)

Enable MSR Swipe

Enables MSR, waiting for swipe to occur.

Parameters

<i>timeout</i>	Swipe Timeout Value
<i>swipeCallback</i>	Redirects the input from main callback to SwipeCallback(string <i>ipAddress</i> , IDTTransactionData <i>transactionData</i> , string <i>ident</i> = "")
<i>failureCallback</i>	Redirects the input from main callback to FailureCallback(string <i>ipAddress</i> , RETURN_CODE <i>errorCode</i> , string <i>ident</i> = "")

Parameters

<i>timeoutCallback</i>	Redirects the input from main callback to TimeoutCallback(string ipAddress, string ident = "")
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with `errorCode.getErrorString(string ident = "")`

21.10.2.299 RETURN_CODE IDTechSDK.IDT_NEO2.pin_cancelPINEntry (string *ident* = " ")

Cancel PIN Entry

Cancel "Get Function Key" & "Get Encrypted PIN" & "Get Numeric" & "Get Amount"

Parameters

<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.
--------------	--

Returns

RETURN_CODE: Values can be parsed with `errorCode.getErrorString(string ident = "")`

21.10.2.300 RETURN_CODE IDTechSDK.IDT_NEO2.pin_capturePin (int *timeout*, int *type*, string *PAN*, int *minPIN*, int *maxPIN*, string *message*, string *ident* = " ")

Capture PIN

Parameters

<i>timeout</i>	Timeout, in seconds. Value of 0 will use system timeout, if any
<i>type</i>	PAN and Key Type <ul style="list-style-type: none"> • 00h = MKSK to encrypt PIN, Internal PAN (from MSR, string ident = "") • 01h = DUKPT to encrypt PIN, Internal PAN (from MSR, string ident = "") • 10h = MKSK to encrypt PIN, External Plaintext PAN • 11h = DUKPT to encrypt PIN, External Plaintext PAN • 20h = MKSK to encrypt PIN, External Ciphertext PAN • 21h = DUKPT to encrypt PIN, External Ciphertext PAN
<i>PAN</i>	Personal Account Number (if internal, value is 0, string ident = "")
<i>minPIN</i>	Minimum PIN Length
<i>maxPIN</i>	Maximum PIN Length
<i>message</i>	LCD Message
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Results returned to MessageCallback. If successful PIN capture, data is returned as DeviceState.Transaction↔Data, with IDTTransactionData cData.pin_pinblock and cData.pin_KSN. If timeout, returns DeviceState.PINTimeout. If error, returns DeviceState.PINFail with one of the following values in data[0]:

- 01h – Fail, Key Pad Cancel
- 02h – Fail, External Command Cancel
- 03h – Fail, Invalid input parameters
- 04h – Fail, PAN error
- 05h – Fail, PIN DUKPT Key is absent
- 06h – Fail, PIN DUKPT Key is exhausted
- 07h – Fail, Display message error
- 0Ch – Fail, Operation Failed

Returns

RETURN_CODE: Values can be parsed with `errorCode.getErrorString(string ident = "")`

21.10.2.301 `RETURN_CODE IDTechSDK.IDT_NEO2.pin_capturePin_ext (int timeout, int type, string PAN, int minPIN, int maxPIN, string message, string ident, SwipeCallback inputCallback, FailureCallback failureCallback, TimeoutCallback timeoutCallback, CancelPromptCallback cancelPromptCallback)`

Capture PIN

Parameters

<i>timeout</i>	Timeout, in seconds. Value of 0 will use system timeout, if any
<i>type</i>	PAN and Key Type <ul style="list-style-type: none"> • 00h = MKSK to encrypt PIN, Internal PAN (from MSR, string ident = "") • 01h = DUKPT to encrypt PIN, Internal PAN (from MSR, string ident = "") • 10h = MKSK to encrypt PIN, External Plaintext PAN • 11h = DUKPT to encrypt PIN, External Plaintext PAN • 20h = MKSK to encrypt PIN, External Ciphertext PAN • 21h = DUKPT to encrypt PIN, External Ciphertext PAN
<i>PAN</i>	Personal Account Number (if internal, value is 0, string ident = "")
<i>minPIN</i>	Minimum PIN Length
<i>maxPIN</i>	Maximum PIN Length
<i>message</i>	LCD Message

Parameters

<i>ident</i>	Device address to execute command on Results returned to MessageCallback. If successful PIN capture, data is returned as DeviceState.TransactionData, with IDTTransactionData cData.pin_pinblock and cData.pin_KSN. If timeout, returns DeviceState.PINTimeout If error, returns DeviceState.PINFail with one of the following values in data[0]: <ul style="list-style-type: none"> • 01h – Fail, Key Pad Cancel • 02h – Fail, External Command Cancel • 03h – Fail, Invalid input parameters • 04h – Fail, PAN error • 05h – Fail, PIN DUKPT Key is absent • 06h – Fail, PIN DUKPT Key is exhausted • 07h – Fail, Display message error • 0Ch – Fail, Operation Failed
<i>inputCallback</i>	Redirects the input from main callback to SwipeCallback(string ipAddress, IDTTransactionData input, ident);
<i>failureCallback</i>	Redirects the input from main callback to FailureCallback(string ipAddress, RETURN_CODE errorCode, string ident = "")
<i>timeoutCallback</i>	Redirects the input from main callback to TimeoutCallback(string ipAddress, string ident = "")
<i>cancelPromptCallback</i>	Redirects the input from main callback to CancelPromptCallback(string ipAddress, string ident = "")
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with errorCode.getErrorString(string ident = "")

21.10.2.302 RETURN_CODE IDTechSDK.IDT_NEO2.pin_capturePinExt (int *timeout*, int *type*, string *PAN*, int *minPIN*, int *maxPIN*, string *message1*, string *message2*, string *verify1*, string *verify2*, string *ident* = " ")

- Capture PIN

Parameters

<i>timeout</i>	180 seconds for first pin, or 360 seconds for first pin + verify pin
----------------	--

Parameters

<i>type</i>	PAN and Key Type <ul style="list-style-type: none"> • 00h: MKSK to encrypt PIN, Internal PAN (from MSR or Manual PAN Entry or Contactless EMV Transaction) • 01h: DUKPT to encrypt PIN, Internal PAN (from MSR or Manual PAN Entry or Contactless EMV Transaction) • 10h: MKSK to encrypt PIN, External Plaintext PAN • 11h: DUKPT to encrypt PIN, External Plaintext PAN • 20h: MKSK to encrypt PIN, External Ciphertext PAN (for PIN pad only) • 21h: DUKPT to encrypt PIN, External Ciphertext PAN (for PIN pad only) • 80h: MKSK to encrypt PIN, Internal PAN, Verify PIN (from MSR or Manual PAN Entry or Contactless EMV Transaction) • 81h: DUKPT to encrypt PIN, Internal PAN, Verify PIN (from MSR or Manual PAN Entry or Contactless EMV Transaction) • 90h: MKSK to encrypt PIN, External Plaintext PAN, Verify PIN • 91h: DUKPT to encrypt PIN, External Plaintext PAN, Verify PIN
<i>PAN</i>	Personal Account Number (if internal, value is 0)
<i>PANLen</i>	Length of PAN
<i>minPIN</i>	Minimum PIN Length
<i>maxPIN</i>	Maximum PIN Length
<i>message1</i>	First line LCD message, up to 16 characters. If null it will display default msg "PLEASE ENTER PIN"
<i>message2</i>	Second line LCD message, up to 16 characters. message1 must not be null.
<i>verify1</i>	First line LCD verification message, up to 16 characters. If null it will display default msg "ENTER PIN AGAIN"
<i>verify2</i>	Second line LCD verification message, up to 16 characters. verify1 must not be null.
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Results returned to MessageCallback. If successful PIN capture, data is returned as DeviceState.Transaction←Data, with IDTTransactionData cData.pin_pinblock and cData.pin_KSN. If timeout, returns DeviceState.PINTimeout. If error, returns DeviceState.PINFail with one of the following values in data[0]:

- 01h – Fail, Key Pad Cancel
- 02h – Fail, External Command Cancel
- 03h – Fail, Invalid input parameters
- 04h – Fail, PAN error
- 05h – Fail, PIN DUKPT Key is absent
- 06h – Fail, PIN DUKPT Key is exhausted
- 07h – Fail, Display message error
- 0Ch – Fail, Operation Failed

Returns

RETURN_CODE: Values can be parsed with errorCode.getErrorString(string ident = "")

21.10.2.303 RETURN_CODE IDTechSDK.IDT_NEO2.pin_getFunctionKey (int *timeout*, string *ident* = " ")**Get Function Key**

Results returned to MessageCallback. If successful function key capture, data is returned as DeviceState.Function↵Key, with data being the ASCII value of the key that was pressed.

@param timeout Timeout, in seconds. Value of 0 will use system timeout, if any
 @param ip Optional: IP address to execute command on (for IP connected devices, string ident = "")

Parameters

<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.
--------------	--

Returns

RETURN_CODE: Values can be parsed with errorCode.getErrorString()g

21.10.2.304 RETURN_CODE IDTechSDK.IDT_NEO2.pin_getFunctionKey_ext (int *timeout*, string *ip*, SwipeCallback *inputCallback*, FailureCallback *failureCallback*, TimeoutCallback *timeoutCallback*, CancelPromptCallback *cancelPromptCallback*)**Get Function Key**

Results returned to MessageCallback. If successful function key capture, data is returned as DeviceState.Function↵Key, with data being the ASCII value of the key that was pressed.

@param timeout Timeout, in seconds. Value of 0 will use system timeout, if any
 @param ip IP address to execute command on (for IP connected devices, string ident = "")

Parameters

<i>inputCallback</i>	Redirects the input from main callback to InputCallback(string ipAddress, IDTTranasctionData input, ident);
<i>failureCallback</i>	Redirects the input from main callback to FailureCallback(string ipAddress, RETURN_CODE errorCode, string ident = "")
<i>timeoutCallback</i>	Redirects the input from main callback to TimeoutCallback(string ipAddress, string ident = "")
<i>cancelPromptCallback</i>	Redirects the input from main callback to CancelPromptCallback(string ipAddress, string ident = "")

Returns

RETURN_CODE: Values can be parsed with errorCode.getErrorString()g

21.10.2.305 RETURN_CODE IDTechSDK.IDT_NEO2.pin_getPanEntry (bool *csc*, bool *expDate*, bool *ADR*, bool *ZIP*, bool *mod10*, UInt16 *timeout*, bool *encPANOnly* = false, string *ident* = " ")**Get Pan**

prompt the user to manually enter a card PAN and Expiry Date (and optionally CSC) from the keypad and return it to the POS.

Parameters

<i>timeout</i>	Number of seconds that the reader waits for the data entry session to complete, stored as a big-endian number. 0 = no timeout
<i>csc</i>	Request CSS
<i>expDate</i>	Request Expiration Date
<i>ADR</i>	Request Address
<i>ZIP</i>	Request Zip
<i>mod10</i>	Validate entered PAN passes MOD-10 checking before accepting
<i>encPANOnly</i>	Output only encrypted PAN
<i>ip</i>	Optional: IP address to execute command on (for IP connected devices, string ident = "")
<i>swipeCallback</i>	Optional: Redirects the input from main callback to SwipeCallback(string ipAddress, IDTTransactionData transactionData, string ident = "")
<i>timeoutCallback</i>	Optional: Redirects the input from main callback to TimeoutCallback(string ipAddress, string ident = "")
<i>cancelPromptCallback</i>	Optional: Redirects the input from main callback to CancelPromptCallback(string ipAddress, string ident = "")
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString

21.10.2.306 RETURN_CODE IDTechSDK.IDT_NEO2.pin_getPanEntry_ext (bool *csc*, bool *expDate*, bool *ADR*, bool *ZIP*, bool *mod10*, UInt16 *timeout*, bool *encPANOnly*, string *ip*, SwipeCallback *swipeCallback*, TimeoutCallback *timeoutCallback*, CancelPromptCallback *cancelPromptCallback*)

Get Pan

prompt the user to manually enter a card PAN and Expiry Date (and optionally CSC) from the keypad and return it to the POS.

Parameters

<i>timeout</i>	Number of seconds that the reader waits for the data entry session to complete, stored as a big-endian number. 0 = no timeout
<i>csc</i>	Request CSS
<i>expDate</i>	Request Expiration Date
<i>ADR</i>	Request Address
<i>ZIP</i>	Request Zip
<i>mod10</i>	Validate entered PAN passes MOD-10 checking before accepting
<i>encPANOnly</i>	Output only encrypted PAN
<i>ip</i>	IP address to execute command on (for IP connected devices, string ident = "")
<i>swipeCallback</i>	Redirects the input from main callback to SwipeCallback(string ipAddress, IDTTransactionData transactionData, string ident = "")
<i>timeoutCallback</i>	Redirects the input from main callback to TimeoutCallback(string ipAddress, string ident = "")
<i>cancelPromptCallback</i>	Redirects the input from main callback to CancelPromptCallback(string ipAddress, string ident = "")

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString

21.10.2.307 RETURN_CODE IDTechSDK.IDT_NEO2.pin_promptForAmount (int *timeout*, int *minLen*, int *maxLen*, string *message*, byte[] *signature*, string *ident* = " ")

Capture Amount

Parameters

<i>timeout</i>	Timeout, in seconds. Value of 0 will use system timeout, if any
<i>minLen</i>	Minimum input Length
<i>maxLen</i>	Maximum input Length
<i>message</i>	LCD Message
<i>signature</i>	Display message signed by Numeric Private Key using RSAPSS algorithm: <ol style="list-style-type: none"> 1. Calculate 32 bytes Hash for "<Display Flag><Key max="" length>="">< Key Min Length><Plaintext display="" message>="">" 2. Using RSAPSS algorithm calculate the Hash to be 256 bytes Raw Data 3. Using Numeric Private Key to sign the Raw Data to be 256 bytes signature
<i>ip</i>	Optional: IP address to execute command on (for IP connected devices, string ident = "")
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with errorCode.getErrorString(string ident = "")

21.10.2.308 RETURN_CODE IDTechSDK.IDT_NEO2.pin_promptForAmount_ext (int *timeout*, int *minLen*, int *maxLen*, string *message*, byte[] *signature*, string *ident*, SwipeCallback *inputCallback*, FailureCallback *failureCallback*, TimeoutCallback *timeoutCallback*, CancelPromptCallback *cancelPromptCallback*)

Capture Amount

Parameters

<i>timeout</i>	Timeout, in seconds. Value of 0 will use system timeout, if any
<i>minLen</i>	Minimum input Length
<i>maxLen</i>	Maximum input Length
<i>message</i>	LCD Message
<i>signature</i>	Display message signed by Numeric Private Key using RSAPSS algorithm: <ol style="list-style-type: none"> 1. Calculate 32 bytes Hash for "<Display Flag><Key max="" length>="">< Key Min Length><Plaintext display="" message>="">" 2. Using RSAPSS algorithm calculate the Hash to be 256 bytes Raw Data 3. Using Numeric Private Key to sign the Raw Data to be 256 bytes signature
<i>ident</i>	Device address to execute command on (for IP connected devices, string ident = "")
<i>inputCallback</i>	Redirects the input from main callback to SwipeCallback(string ipAddress, IDTTransactionData input, ident);

Parameters

<i>failureCallback</i>	Redirects the input from main callback to FailureCallback(string ipAddress, RETURN_CODE errorCode, string ident = "")
<i>timeoutCallback</i>	Redirects the input from main callback to TimeoutCallback(string ipAddress, string ident = "")
<i>cancelPromptCallback</i>	Redirects the input from main callback to CancelPromptCallback(string ipAddress, string ident = "")
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with errorCode.getErrorString(string ident = "")

21.10.2.309 RETURN_CODE IDTechSDK.IDT_NEO2.pin_promptForInput (int *messageID*, short *timeout*, string *ip* = null, string *ident* = " ")

Capture Numeric Input From Message ID

Parameters

<i>messageID</i>	Message ID <ul style="list-style-type: none"> 01 = "Please Swipe Or Key Employee ID", delayed masking, enable MSR, min len = 1, max len = 16
<i>timeout</i>	Timeout, in seconds. Value of 0 will use system timeout, if any
<i>ip</i>	Optional: IP address to execute command on (for IP connected devices, string ident = "")
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with errorCode.getErrorString(string ident = "")

21.10.2.310 RETURN_CODE IDTechSDK.IDT_NEO2.pin_promptForInput_ext (int *messageID*, short *timeout*, string *ident*, SwipeCallback *inputCallback*, SwipeCallback *swipeCallback*, FailureCallback *failureCallback*, TimeoutCallback *timeoutCallback*, CancelPromptCallback *cancelPromptCallback*)

Capture Numeric Input From Message ID

Parameters

<i>messageID</i>	Message ID <ul style="list-style-type: none"> 01 = "Please Swipe Or Key Employee ID", delayed masking, enable MSR, min len = 1, max len = 16
<i>timeout</i>	Timeout, in seconds. Value of 0 will use system timeout, if any
<i>ident</i>	Device address to execute command on (for IP connected devices, string ident = "")
<i>inputCallback</i>	Redirects the input from main callback to SwipeCallback(string ipAddress, IDTTransactionData input, ident);
<i>swipeCallback</i>	Redirects the input from main callback to SwipeCallback(string ipAddress, IDTTransactionData transactionData, string ident = "")

Parameters

<i>failureCallback</i>	Redirects the input from main callback to FailureCallback(string ipAddress, RETURN_CODE errorCode, string ident = "")
<i>timeoutCallback</i>	Redirects the input from main callback to TimeoutCallback(string ipAddress, string ident = "")
<i>cancelPromptCallback</i>	Redirects the input from main callback to CancelPromptCallback(string ipAddress, string ident = "")
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with errorCode.getErrorString(string ident = "")

21.10.2.311 RETURN_CODE IDTechSDK.IDT_NEO2.pin_promptForNumericKeyWithSwipe (short *timeout*, byte *function*, int *minLen*, int *maxLen*, string *line1*, string *line2*, byte[] *signature*, string *ident* = " ")

Capture Numeric Input

@param timeout Timeout, in seconds. Value of 0 will use system timeout, if any

Parameters

<i>function</i>	<ul style="list-style-type: none"> • 0x00 = Plaintext Input • 0x01 = Masked Input • 0x02 = Delayed Masking Input • 0x10 = Plaintext Input + MSR Active • 0x11 = Masked Input + MSR Active • 0x12 = Delayed Masking Input + MSR Active
<i>minLen</i>	Minimum input Length
<i>maxLen</i>	Maximum input Length
<i>line1</i>	Line 1 of LCD Message - 16 chars max
<i>line2</i>	Line 2 of LCD Message - 16 chars max
<i>signature</i>	Display message signed by Numeric Private Key using RSAPSS algorithm: <ol style="list-style-type: none"> 1. Calculate 32 bytes Hash for "<Display Flag><Key max="" length="">< Key Min Length><Plaintext display="" message="">" 2. Using RSAPSS algorithm calculate the Hash to be 256 bytes Raw Data 3. Using Numeric Private Key to sign the Raw Data to be 256 bytes signature
<i>ip</i>	Optional: IP address to execute command on (for IP connected devices, string ident = "")
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with `errorCode.getErrorString(string ident = "")`

21.10.2.312 `RETURN_CODE IDTechSDK.IDT_NEO2.pin_promptForNumericKeyWithSwipe_ext (short timeout, byte function, int minLen, int maxLen, string line1, string line2, byte[] signature, string ident, SwipeCallback inputCallback, SwipeCallback swipeCallback, FailureCallback failureCallback, TimeoutCallback timeoutCallback, CancelPromptCallback cancelPromptCallback)`

Capture Numeric Input

@param timeout Timeout, in seconds. Value of 0 will use system timeout, if any

Parameters

<i>function</i>	<ul style="list-style-type: none"> • 0x00 = Plaintext Input • 0x01 = Masked Input • 0x02 = Delayed Masking Input • 0x10 = Plaintext Input + MSR Active • 0x11 = Masked Input + MSR Active • 0x12 = Delayed Masking Input + MSR Active
<i>minLen</i>	Minimum input Length
<i>maxLen</i>	Maximum input Length
<i>line1</i>	Line 1 of LCD Message - 16 chars max
<i>line2</i>	Line 2 of LCD Message - 16 chars max
<i>signature</i>	Display message signed by Numeric Private Key using RSAPSS algorithm: <ol style="list-style-type: none"> 1. Calculate 32 bytes Hash for "<Display Flag><Key max="" length="">< Key Min Length><Plaintext display="" message="">" 2. Using RSAPSS algorithm calculate the Hash to be 256 bytes Raw Data 3. Using Numeric Private Key to sign the Raw Data to be 256 bytes signature
<i>ident</i>	: Device address to execute command on (for IP connected devices, string ident = "")
<i>inputCallback</i>	Redirects the input from main callback to <code>SwipeCallback(string ipAddress, IDTTransactionData input, ident);</code>
<i>swipeCallback</i>	Redirects the input from main callback to <code>SwipeCallback(string ipAddress, IDTTransactionData transactionData, string ident = "")</code>
<i>failureCallback</i>	Redirects the input from main callback to <code>FailureCallback(string ipAddress, RETURN_CODE errorCode, string ident = "")</code>
<i>timeoutCallback</i>	Redirects the input from main callback to <code>TimeoutCallback(string ipAddress)p</code>
<i>cancelPromptCallback</i>	Redirects the input from main callback to <code>CancelPromptCallback(string ipAddress, string ident = "")</code>
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with `errorCode.getErrorString(string ident = "")`

21.10.2.313 RETURN_CODE IDTechSDK.IDT_NEO2.pin_sendBeep (string *ident* = " ")**Send Beep**

Executes a beep request.

Parameters

<i>ip</i>	Optional: IP address to execute command on (for IP connected devices, string <i>ident</i> = "")
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with `device.getResponseCodeString`

21.10.2.314 static String IDTechSDK.IDT_NEO2.SDK_Version () [static]**SDK Version**

- All Devices

Returns the current version of SDK

Returns

Framework version

21.10.2.315 static void IDTechSDK.IDT_NEO2.setCallback (CallBack *my_Callback*) [static]**Set Callback**

Sets the class callback

21.10.2.316 static void IDTechSDK.IDT_NEO2.setCallback (IntPtr *my_Callback*, SynchronizationContext *context*) [static]**Set Callback**

Sets the class callback

Parameters

<i>my_Callback</i>	The callback function to receive the response message from device. defined as follows. public unsafe delegate void MFCCallBack(Parameters parameters, ident);
<i>context</i>	The context of the UI thread

21.10.2.317 `static void IDTechSDK.IDT_NEO2.setCallbackIP (CallbackIP my_Callback, string ident = " ") [static]`

Set Callback with IP

Sets the class callback that also reports back IP device information (for network connected devices, string ident = "")

Parameters

<i>my_Callback</i>	The callback function to receive the response message from device. defined as follows. public delegate void CallbackIP(IDT_DEVICE_Types sender, DeviceState state, byte[] data, IDTTransactionData card, EMV_Callback emvCallback, RETURN_CODE transactionResultCode, string IP, ident);
<i>ip</i>	Optional: The callback function will only be applicable to the provided IP.

21.10.2.318 `static void IDTechSDK.IDT_NEO2.setCommandTimeout (int milliseconds, string ident = " ") [static]`

Set Command Timeout

Sets the default timeout (in milliseconds) waiting for a blocking command response

Parameters

<i>milliseconds</i>	Time
---------------------	------

21.10.2.319 `static void IDTechSDK.IDT_NEO2.setLongPressCallback (longPressCallback callback, string ident = " ") [static]`

Set Longpress Callback

Sets the class callback that also reports back IP device information (for network connected devices, string ident = "")

Parameters

<i>callback</i>	The callback function to receive the response message from device. defined as follows. Location 0 = top left, Location 1 = top right. public delegate void longPressCallback(IDT_DEVICE_Types sender, UInt16 screenID, byte location, string IP, ident);
<i>ip</i>	Optional: The callback function will only be applicable to the provided IP.

21.10.2.320 `static bool IDTechSDK.IDT_NEO2.useSerialPort (int port) [static]`

Use Serial Port Interface

Instructs SDK to attempt to use the Serial Port for communication with [IDT_NEO2](#) using default baud rate

Parameters

<i>port</i>	Serial Port to use. Example COM1 = 1.
-------------	---------------------------------------

Returns

bool TRUE=successful, FALSE=failure

21.10.2.321 `static bool IDTechSDK.IDT_NEO2.useSerialPort (int port, int baud)` `[static]`

Use Serial Port Interface with baud rate

Instructs SDK to attempt to use the Serial Port for communication with [IDT_NEO2](#)

Parameters

<i>port</i>	Serial Port to use. Example COM1 = 1.
<i>baud</i>	Baud rate to override default. Example 115200;

Returns

bool TRUE=successful, FALSE=failure

21.10.2.322 `static bool IDTechSDK.IDT_NEO2.useSerialPortLinux (string path)` `[static]`

Use Serial Port Interface on Linux

Instructs SDK to attempt to use the Serial Port for communication with BTMag using default baud rate on Linux implementations

Parameters

<i>path</i>	Path to use. Example /dev/ttyUSB*
-------------	-----------------------------------

Returns

bool TRUE=successful, FALSE=failure

21.10.2.323 `static bool IDTechSDK.IDT_NEO2.useSerialPortLinux (string path, int baud)` `[static]`

Use Serial Port Interface on Linux with baud rate

Instructs SDK to attempt to use the Serial Port for communication with BTPay on Linux implementations

Parameters

<i>path</i>	Path to use. Example /dev/ttyUSB*
<i>baud</i>	Baud rate to override default. Example 115200;

Returns

bool TRUE=successful, FALSE=failure

21.10.3 Property Documentation

21.10.3.1 IDT_NEO2 IDTechSDK.IDT_NEO2.SharedController [static], [get]

Singleton Instance

Establishes an singleton instance of [IDT_NEO2](#) class.

Returns

Instance of [IDT_NEO2](#)

The documentation for this class was generated from the following file:

- /Users/randy/Repo/universal-sdk/IDTechSDK_STD/DeviceHeaders/IDT_NEO2.cs

21.11 IDTechSDK.IDT_PIP Class Reference

Public Member Functions

- RETURN_CODE [config_getSerialNumber](#) (ref string response, string ident="")
- RETURN_CODE [device_retrieveAIDList](#) (ref byte[][] response, string ident="")
- RETURN_CODE [device_sendDataCommand](#) (string cmd, bool calcLRC, ref byte[] response, string ident="")
- RETURN_CODE [device_sendDataCommand_ext](#) (string cmd, bool calcLRC, ref byte[] response, int timeout, bool noResponse, string ident="")
- RETURN_CODE [device_getFirmwareVersion](#) (ref string response, string ident="")
- RETURN_CODE [device_getMerchantRecord](#) (int index, ref byte[] record, string ident="")
- RETURN_CODE [device_setPIPMODE](#) (int mode, string ident="")
- RETURN_CODE [device_getPIPMODE](#) (ref int mode, string ident="")
- RETURN_CODE [device_getRKIStatus](#) (bool isProd, bool isMultiKey, ref string status, ref Dictionary< string, RKI_KEY_TYPE > keys, string ident="")
- RETURN_CODE [device_getAnyRKIStatus](#) (bool isMultiKey, ref string status, ref Dictionary< string, RKI_KEY_TYPE > keys, string ident="")
- RETURN_CODE [device_getTransactionResults](#) (ref IDTTransactionData results, string ident="")
- RETURN_CODE [device_pingDevice](#) (string ident="")
- RETURN_CODE [device_controlUserInterface](#) (byte[] values, string ident="")
- RETURN_CODE [device_sendVivoCommandP2](#) (byte command, byte subCommand, byte[] data, ref byte[] response, string ident="")
- RETURN_CODE [device_sendVivoCommandP2_ext](#) (byte command, byte subCommand, byte[] data, ref byte[] response, int timeout, bool noResponse, string ident="")
- RETURN_CODE [ctls_retrieveAIDList](#) (ref byte[][] response, string ident="")
- RETURN_CODE [ctls_getAllConfigurationGroups](#) (ref byte[][] response, string ident="")
- RETURN_CODE [ctls_retrieveApplicationData](#) (byte[] AID, ref byte[] tlv, string ident="")
- RETURN_CODE [ctls_removeApplicationData](#) (byte[] AID, string ident="")
- RETURN_CODE [ctls_setApplicationData](#) (byte[] tlv, string ident="")
- RETURN_CODE [ctls_setDefaultConfiguration](#) (string ident="")
- RETURN_CODE [ctls_setConfigurationGroup](#) (byte[] tlv, string ident="")
- RETURN_CODE [ctls_retrieveTerminalData](#) (ref byte[] tlv, string ident="")
- RETURN_CODE [ctls_setTerminalData](#) (byte[] tlv, string ident="")
- RETURN_CODE [ctls_trySetTerminalData](#) (byte[] tlv, ref byte[] rejectedTLV, ref byte[] convertedTLV, string ident="")
- RETURN_CODE [ctls_getConfigurationGroup](#) (int group, ref byte[] tlv, string ident="")
- RETURN_CODE [ctls_removeConfigurationGroup](#) (int group, string ident="")
- RETURN_CODE [device_startTransaction](#) (double amount, double amtOther, int exponent, int type, int timeout, byte[] tags, byte[] options=null, bool isFastEMV=false, int transMode=0, string ident="")
- RETURN_CODE [device_activateTransaction](#) (int timeout, byte[] tags, byte[] options=null, bool isFastEMV=false, string ident="")

- RETURN_CODE [ctls_startTransaction](#) (double amount, double amtOther, int exponent, int type, int timeout, byte[] tags, bool forceOnline=false, bool isFastEMV=false, string ident="")
- RETURN_CODE [ctls_updateBalance](#) (byte statusCode, byte[] authCode, byte[] date, byte[] time, string ident="")
- RETURN_CODE [ctls_activateTransaction](#) (int timeout, byte[] tags, bool forceOnline=false, bool isFastEMV=false, string ident="")
- RETURN_CODE [ctls_cancelTransaction](#) (string ident="")
- RETURN_CODE [device_setBurstMode](#) (byte mode, string ident="")
- RETURN_CODE [device_setMerchantRecord](#) (int index, bool enabled, string merchantID, string merchantURL, string ident="")
- RETURN_CODE [device_pollForToken](#) (byte seconds, ref byte card, ref byte[] serialNumber, string ident="")
- RETURN_CODE [device_setPollMode](#) (byte mode, string ident="")
- RETURN_CODE [device_updateDeviceFirmware](#) (byte[] firmwareData, string ident="")
- RETURN_CODE [device_enablePassThrough](#) (bool enablePassThrough, string ident="")
- RETURN_CODE [ctls_nfcCommand](#) (byte[] nfcCmdPkt, ref byte[] response, string ident="")
- RETURN_CODE [felica_requestService](#) (byte[] nodeCode, ref byte[] response, string ident="")
- RETURN_CODE [device_updateFirmwareFromZip](#) (byte[] zipfile, string ident="", bool isForeground=false)
- RETURN_CODE [device_sendConfigurationFromZip](#) (byte[] zip, string filename, VC_OPERATION_TYPE type, bool matchFW, string ident="", bool isForeground=false)
- RETURN_CODE [device_sendConfiguration](#) (string filename, VC_OPERATION_TYPE type, bool matchFW, string ident="", bool isForeground=false)
- RETURN_CODE [device_readConfigurationToMemory](#) (string memo, Configs.InstallRules rules, List< Configs.DeviceCommand > cmds, string ident="", bool isForeground=false)
- RETURN_CODE [device_getConfigurationFromMemory](#) (ref string json, string ident="")
- RETURN_CODE [device_RemoteKeyInjection](#) (RKI_KEY_TYPE type, string keyName, string ident="", bool performOnForeground=false)

Static Public Member Functions

- static int [getCommandTimeout](#) (string ident="")
- static void [setCommandTimeout](#) (int milliseconds, string ident="")
- static void [initSC](#) ()
- static void [setCallback](#) (Callback my_Callback)
- static void [setCallback](#) (IntPtr my_Callback, SynchronizationContext context)
- static String [SDK_Version](#) ()
- static void [lcd_retrieveMessage](#) (DisplayMessages.DISPLAY_MESSAGE_LANGUAGE lang, DisplayMessages.DISPLAY_MESSAGE_IDENTIFIER id, ref string line1, ref string line2)

Properties

- static [IDT_PIP SharedController](#) [get]

21.11.1 Detailed Description

Class for PIP ICC reader

21.11.2 Member Function Documentation

21.11.2.1 RETURN_CODE IDTechSDK.IDT_PIP.config_getSerialNumber (ref string response, string ident = " ")

Polls device for Serial Number

Parameters

<i>response</i>	Returns Serial Number
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with `device_getResponseCodeString`

21.11.2.2 RETURN_CODE IDTechSDK.IDT_PIP.ctls_activateTransaction (int timeout, byte[] tags, bool forceOnline = false, bool isFastEMV = false, string ident = " ")

Start CTLS Transaction Request

Authorizes the CTLS transaction

The tags will be returned in the callback routine.

Parameters

<i>timeout</i>	Timeout value in seconds.
<i>tags</i>	The tags to be included in the request. Passed as TLV Data. Example, tag 9F02 with amount 0x000000000100 would be 0x9F0206000000000100
<i>forceOnline</i>	Not used for PIP. Parameter included for API compability with other IDTech Devices
<i>isFastEMV</i>	Not used for PIP. Parameter included for API compability with other IDTech Devices
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with `errorCode.getErrorString()`

NOTE ON APPLEPAY VAS: To enable ApplePay VAS, first a merchant record must be defined in one of the six available index positions (1-6) using `device_setMerchantRecord`, then container tag FFEE06 must be sent as part of the additional tags parameter of `ctls_startTransaction`. Tag FFEE06 must contain tag 9F26 and 9F22, and can optionally contain tags 9F2B and DFO1. Example FFEE06189F220201009F2604000000009F2B050100000000DF010101 9F22 = two bytes = ApplePay Terminal Application Version Number. Hard defined as 0100 for now. (required, string ident = "") 9F26 = four bytes = ApplePay Terminal Capabilities Information (required, string ident = "")

- Byte 1 = RFU
- Byte 2 = Terminal Type
- - Bit 8 = VAS Support (1=on, 0 = off, string ident = "")
- - Bit 7 = Touch ID Required (1=on, 0 = off, string ident = "")
- - Bit 6 = RFU
- - Bit 5 = RFU
- - Bit 1,2,3,4
- - - 0 = Payment Terminal
- - - 1 = Transit Terminal
- - - 2 = Access Terminal

- - - 3 = Wireless Handoff Terminal
- - - 4 = App Handoff Terminal
- - - 15 = Other Terminal
- Byte 3 = RFU
- Byte 4 = Terminal Mode
- - 0 = ApplePay VAS OR ApplePay
- - 1 = ApplePay VAS AND ApplePay
- - 2 = ApplePay VAS ONLY
- - 3 = ApplePay ONLY 9F2B = 5 bytes = ApplePay VAS Filter. Each byte filters for that specific merchant index (optional, string ident = "") DF01 = 1 byte = ApplePay VAS Protocol. (optional, string ident = "")
- - Bit 1 : 1 = URL VAS, 0 = Full VAS
- - Bit 2 : 1 = VAS Beeps, 0 = No VAS Beeps
- - Bit 3 : 1 = Silent Comm Error, 2 = EMEA Comm Error
- - Bit 4-8 : RFU

21.11.2.3 RETURN_CODE IDTechSDK.IDT_PIP.ctls_cancelTransaction (string *ident* = " ")

Cancel EMV Transaction

Cancels the currently executing EMV transaction.

Parameters

<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.
--------------	--

Returns

RETURN_CODE: Values can be parsed with `errorCode.getErrorString()`

21.11.2.4 RETURN_CODE IDTechSDK.IDT_PIP.ctls_getAllConfigurationGroups (ref byte *response*[], string *ident* = " ")

Retrieve All Configuration Groups

Returns all the Configuration Groups installed on the terminal for CTLS

Parameters

<i>response</i>	array of CTLS groups as TLV bytes
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with `errorCode.getErrorString(string ident = "")`

21.11.2.5 RETURN_CODE IDTechSDK.IDT_PIP.ctls_getConfigurationGroup (int *group*, ref byte[] *tlv*, string *ident* = " ")

Get Configuration Group

Retrieves the Configuration for the specified Group.

Parameters

<i>group</i>	Configuration Group
<i>tlv</i>	return data
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with `errorCode.getErrorString()`

21.11.2.6 RETURN_CODE IDTechSDK.IDT_PIP.ctls_nfcCommand (byte[] *nfcCmdPkt*, ref byte[] *response*, string *ident* = " ")

NFC Command

This command uses `nfcCmdPkt[0]` in command data field to implement different functions. This command should be used in Pass-Through mode and command with "Poll for a NFC Tag" data should be used first. Command with other data can only be used once the "Poll for a NFC Tag" command has indicated that a NFC tag is present.

Parameters

<i>nfcCmdPkt</i>	<p>System Code</p> <ul style="list-style-type: none"> • Poll for NFC Tag: nfcCmdPkt[0] = 0xff, nfcCmdPkt[1] = timeout value (in seconds, string ident = "") • Tag1 Static Get All Data: nfcCmdPkt[0] = 0x11 • Tag1 Static Read a Byte: nfcCmdPkt[0] = 0x12, nfcCmdPkt[1] = Address of Data • Tag1 Static Write a Byte: nfcCmdPkt[0] = 0x13 nfcCmdPkt[1] = Address of Data, nfcCmdPkt[2] = Data to be written • Tag1 Static Write a Byte NE: nfcCmdPkt[0] = 0x14, nfcCmdPkt[1] = Address of Data, nfcCmdPkt[2] = Data to be written • Tag1 Dynamic Read a Segment: nfcCmdPkt[0] = 0x15, nfcCmdPkt[1] = Address of Segment • Tag1 Dynamic Read 8 Bytes: nfcCmdPkt[0] = 0x16, nfcCmdPkt[1] = Address of Data • Tag1 Dynamic Write 8 Bytes: nfcCmdPkt[0] = 0x17, nfcCmdPkt[1] = Address of Data, nfcCmdPkt[2]~nfcCmdPkt[9] = Data to be written • Tag1 Dynamic Write 8 Bytes NE: nfcCmdPkt[0] = 0x18, nfcCmdPkt[1] = Address of Data, nfcCmdPkt[2]~nfcCmdPkt[9] = Data to be written • Tag2 Read Data (16 bytes): nfcCmdPkt[0] = 0x21, nfcCmdPkt[1] = Address of Data • Tag2 Write Data (4 bytes): nfcCmdPkt[0] = 0x22, nfcCmdPkt[1] = Address of Data, nfcCmdPkt[2]~nfcCmdPkt[5] = Data to be written • Tag2 Select Sect: nfcCmdPkt[0] = 0x23, nfcCmdPkt[1] = Sect number • Tag3 Read Data: – nfcCmdPkt[0] = 0x41, – nfcCmdPkt[1] = Number of services, value n – nfcCmdPkt[2]~nfcCmdPkt[2n+1]: Service code list – nfcCmdPkt[2n+2]: Number of blocks, value m. – nfcCmdPkt[2n+3....]: Block list, length is 2m~3m • Tag3 Write Data: – nfcCmdPkt[0] = 0x41, – nfcCmdPkt[1] = Number of services, value n – nfcCmdPkt[2]~nfcCmdPkt[2n+1]: Service code list – nfcCmdPkt[2n+2]: Number of blocks, value m. – nfcCmdPkt[2n+3....]: Block list, length is 2m~3m – nfcCmdPkt[...]: Block data, length is 16m • Tag4 Command: nfcCmdPkt[0] = 0x81, nfcCmdPkt[1]~nfcCmdPkt[n]:data
<i>response</i>	Response as explained in FeliCA Lite-S User's Manual
<i>ip</i>	IP Address of target device (optional, string ident = "")
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with `errorCode.getErrorString(string ident = "")`

21.11.2.7 RETURN_CODE IDTechSDK.IDT_PIP.ctls_removeApplicationData (byte[] AID, string ident = " ")

Remove Application Data by AID

Removes the Application Data for CTLS as specified by the AID name passed as a parameter

Parameters

<i>AID</i>	Name of ApplicationID Must be between 5 and 16 bytes
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with `errorCode.getErrorString(string ident = "")`

21.11.2.8 RETURN_CODE IDTechSDK.IDT_PIP.ctls_removeConfigurationGroup (int *group*, string *ident* = " ")

Remove Configuration Group

Removes the Configuration as specified by the Group. Must not by group 0

Parameters

<i>group</i>	Configuration Group
--------------	---------------------

Return values

<i>RETURN_CODE</i>	Return codes listed as typedef enum in IDTCommon:RETURN_CODE. Values can be parsed with IDT_BTPay::device_getResponseCodeString:(string ident = "")
--------------------	---

21.11.2.9 RETURN_CODE IDTechSDK.IDT_PIP.ctls_retrieveAIDList (ref byte *response*[[], string *ident* = " ")

Retrieve AID list

Returns all the AID names and their assigned groups installed on the terminal for CTLS.

Parameters

<i>response</i>	array of 2-tag TLV data objects: FFE4 (group name) followed by 9F06 (AID, string <i>ident</i> = "")
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with `errorCode.getErrorString(string ident = "")`

21.11.2.10 RETURN_CODE IDTechSDK.IDT_PIP.ctls_retrieveApplicationData (byte[] *AID*, ref byte[] *tlv*, string *ident* = " ")

Retrieve Application Data by AID

Retrieves the CTLS Application Data as specified by the AID name passed as a parameter.

Parameters

<i>AID</i>	Name of ApplicationID. Must be between 5 and 16 bytes
<i>tlv</i>	The TLV elements of the requested AID
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with `errorCode.getErrorString()`

21.11.2.11 RETURN_CODE IDTechSDK.IDT_PIP.ctls_retrieveTerminalData (ref byte[] *tlv*, string *ident* = " ")

Retrieve Terminal Data

Retrieves the Terminal Data for CTLS. This is configuration group 0 (Tag FFEE - > FFEE0100). The terminal data can also be retrieved by `ctls_getConfigurationGroup(0)`.

Parameters

<i>tlv</i>	Response returned as a TLV
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with `errorCode.getErrorString()`

21.11.2.12 RETURN_CODE IDTechSDK.IDT_PIP.ctls_setApplicationData (byte[] *tlv*, string *ident* = " ")

Set Application Data by AID

Sets the Application Data for CTLS as specified by TLV data

Parameters

<i>tlv</i>	Application data in TLV format The first tag of the TLV data must be the group number (FFE4). The second tag of the TLV data must be the AID (9F06)
------------	---

Example valid TLV, for Group #2, AID a0000000035010: "ffe401029f0607a0000000051010ffe10101ffe50110ffe30114ffe20106"

Parameters

<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.
--------------	--

Returns

RETURN_CODE: Values can be parsed with `errorCode.getErrorString()`

21.11.2.13 RETURN_CODE IDTechSDK.IDT_PIP.ctls_setConfigurationGroup (byte[] *tlv*, string *ident* = " ")

Set Configuration Group

Sets the Configuration Group for CTLS as specified by the TLV data

Parameters

<i>tlv</i>	Configuration Group Data in TLV format The first tag of the TLV data must be the group number (FFE4). A second tag must exist
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with `errorCode.getErrorString()`

21.11.2.14 RETURN_CODE IDTechSDK.IDT_PIP.ctls_setDefaultConfiguration (string *ident* = " ")

Set Default Configuration Group

Resets the device to default CTLS configuration group settings

Parameters

<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.
--------------	--

Returns

RETURN_CODE: Values can be parsed with `errorCode.getErrorString()`

21.11.2.15 RETURN_CODE IDTechSDK.IDT_PIP.ctls_setTerminalData (byte[] *tlv*, string *ident* = " ")

Set Terminal Data

Sets the Terminal Data for CTLS as specified by the TLV. The first TLV must be Configuration Group Number (Tag FFE4). The terminal global data is group 0, so the first TLV would be FFE40100. Other groups can be defined using this method (1 or greater), and those can be retrieved with `emv_getConfigurationGroup(int group)`, and deleted with `emv_removeConfigurationGroup(int group)`. You cannot delete group 0.

Parameters

<i>tlv</i>	TerminalData configuration data
------------	---------------------------------

Return values

<i>RETURN_CODE</i>	Return codes listed as typedef enum in IDTCommon:RETURN_CODE. Values can be parsed with IDT_BTPay::device_getResponseCodeString:(string ident = "")
--------------------	---

21.11.2.16 RETURN_CODE IDTechSDK.IDT_PIP.ctls_startTransaction (double *amount*, double *amtOther*, int *exponent*, int *type*, int *timeout*, byte[] *tags*, bool *forceOnline* = false, bool *isFastEMV* = false, string *ident* = " ")

Start CTLS Transaction Request

The tags will be returned in the callback routine.

Parameters

<i>amount</i>	Transaction amount value (tag value 9F02, string ident = "")
<i>amtOther</i>	Other amount value, if any (tag value 9F03, string ident = "")
<i>type</i>	Transaction type (tag value 9C).
<i>timeout</i>	Timeout value in seconds.
<i>tags</i>	Any other tags to be included in the request. Passed as TLV data. Example, tag 9F02 with amount 0x000000000100 would be 0x9F0206000000000100 If tags 9F02 (amount), 9F03 (other amount), or 9C (transaction type) are included, they will take priority over these values supplied as individual parameters to this method.

Parameters

<i>forceOnline</i>	Not used for PIP. Parameter included for API compability with other IDTech Devices
<i>isFastEMV</i>	Not used for PIP. Parameter included for API compability with other IDTech Devices
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with `errorCode.getErrorString()`

NOTE ON APPLEPAY VAS: To enable ApplePay VAS, first a merchant record must be defined in one of the six available index positions (1-6) using `device_setMerchantRecord`, then container tag FFEE06 must be sent as part of the additional tags parameter of `ctls_startTransaction`. Tag FFEE06 must contain tag 9F26 and 9F22, and can optionally contain tags 9F2B and DFO1. Example FFEE06189F220201009F2604000000009F2B050100000000↵DF010101 9F22 = two bytes = ApplePay Terminal Application Version Number. Hard defined as 0100 for now. (required, string ident = "") 9F26 = four bytes = ApplePay Terminal Capabilities Information (required, string ident = "")

- Byte 1 = RFU
- Byte 2 = Terminal Type
 - - Bit 8 = VAS Support (1=on, 0 = off, string ident = "")
 - - Bit 7 = Touch ID Required (1=on, 0 = off, string ident = "")
 - - Bit 6 = RFU
 - - Bit 5 = RFU
 - - Bit 1,2,3,4
 - - - 0 = Payment Terminal
 - - - 1 = Transit Terminal
 - - - 2 = Access Terminal
 - - - 3 = Wireless Handoff Terminal
 - - - 4 = App Handoff Terminal
 - - - 15 = Other Terminal
- Byte 3 = RFU
- Byte 4 = Terminal Mode
 - - 0 = ApplePay VAS OR ApplePay
 - - 1 = ApplePay VAS AND ApplePay
 - - 2 = ApplePay VAS ONLY
 - - 3 = ApplePay ONLY 9F2B = 5 bytes = ApplePay VAS Filter. Each byte filters for that specific merchant index (optional, string ident = "") DFO1 = 1 byte = ApplePay VAS Protocol. (optional, string ident = "")
 - - Bit 1 : 1 = URL VAS, 0 = Full VAS
 - - Bit 2 : 1 = VAS Beeps, 0 = No VAS Beeps
 - - Bit 3 : 1 = Silent Comm Error, 2 = EMEA Comm Error
 - - Bit 4-8 : RFU

21.11.2.17 **RETURN_CODE** IDTechSDK.IDT_PIP.ctls_trySetTerminalData (byte[] *tlv*, ref byte[] *rejectedTLV*, ref byte[] *convertedTLV*, string *ident* = " ")

Try to Set CTLS Terminal Data

Attempts to set the CTLS Terminal Data. Use this functions to attempt to set the terminal data to the device. This function will allow the cross-device tag compatible writing of IDTech Legacy/New tag definitions

Parameters

<i>tlv</i>	TerminalData TLV Data
<i>rejectedTLV</i>	Contains the tags (if any) that were rejected by the firmware
<i>convertedTLV</i>	Contains the tags (if any) that were converted and accepted by the firmware
<i>ident</i>	Optional identifier

Return values

RETURN_CODE	Return codes listed as typedef enum in IDTCommon:RETURN_CODE. Values can be parsed with IDT_BTPay::device_getResponseCodeString:()
--------------------	--

21.11.2.18 **RETURN_CODE** IDTechSDK.IDT_PIP.ctls_updateBalance (byte *statusCode*, byte[] *authCode*, byte[] *date*, byte[] *time*, string *ident* = " ")

Update Balance

This command is the authorization response sent by the issuer to the terminal including the Authorization Status (OK or NOT OK).

This command is also being used in some implementations (i.e. EMEA) to communicate the results of Issuer Authentication to the reader in order to display the correct LCD messages. With this command, the POS passes the authorization result (OK/NOT OK), and possibly the Authorization Code (Auth_Code)/Date/Time to the terminal.

Parameters

<i>statusCode</i>	00: OK, 01: NOT OK, 02: (ARC response 89 for Interac, string ident = "")
<i>authCode</i>	Authorization code from host. Six bytes. Optional
<i>date</i>	Transaction date. If null, uses current terminal date. 3 bytes compressed numeric YYMMDD (tag value 9A).
<i>time</i>	Transaction time. If null, uses current terminal time. 3 bytes compressed numeric HHMMSS (tag value 9F21).
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString

21.11.2.19 **RETURN_CODE** IDTechSDK.IDT_PIP.device_activateTransaction (int *timeout*, byte[] *tags*, byte[] *options* = null, bool *isFastEMV* = false, string *ident* = " ")

Start CTLS Transaction Request

Authorizes the CTLS transaction

The tags will be returned in the callback routine.

Parameters

<i>timeout</i>	Timeout value in seconds.
<i>tags</i>	The tags to be included in the request. Passed as TLV Data. Example, tag 9F02 with amount 0x000000000100 would be 0x9F0206000000000100
<i>options</i>	Not used for PIP. Parameter included for API compability with other IDTech Devices
<i>isFastEMV</i>	Not used for PIP. Parameter included for API compability with other IDTech Devices
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with `errorCode.getErrorString()`

NOTE ON APPLEPAY VAS: To enable ApplePay VAS, first a merchant record must be defined in one of the six available index positions (1-6) using `device_setMerchantRecord`, then container tag FFEE06 must be sent as part of the additional tags parameter of `ctls_startTransaction`. Tag FFEE06 must contain tag 9F26 and 9F22, and can optionally contain tags 9F2B and DFO1. Example FFEE06189F220201009F2604000000009F2B050100000000DF010101 9F22 = two bytes = ApplePay Terminal Application Version Number. Hard defined as 0100 for now. (required, string ident = "") 9F26 = four bytes = ApplePay Terminal Capabilities Information (required, string ident = "")

- Byte 1 = RFU
- Byte 2 = Terminal Type
- - Bit 8 = VAS Support (1=on, 0 = off, string ident = "")
- - Bit 7 = Touch ID Required (1=on, 0 = off, string ident = "")
- - Bit 6 = RFU
- - Bit 5 = RFU
- - Bit 1,2,3,4
- - - 0 = Payment Terminal
- - - 1 = Transit Terminal
- - - 2 = Access Terminal
- - - 3 = Wireless Handoff Terminal
- - - 4 = App Handoff Terminal
- - - 15 = Other Terminal
- Byte 3 = RFU
- Byte 4 = Terminal Mode
- - 0 = ApplePay VAS OR ApplePay
- - 1 = ApplePay VAS AND ApplePay
- - 2 = ApplePay VAS ONLY
- - 3 = ApplePay ONLY 9F2B = 5 bytes = ApplePay VAS Filter. Each byte filters for that specific merchant index (optional, string ident = "") DFO1 = 1 byte = ApplePay VAS Protocol. (optional, string ident = "")
- - Bit 1 : 1 = URL VAS, 0 = Full VAS
- - Bit 2 : 1 = VAS Beeps, 0 = No VAS Beeps
- - Bit 3 : 1 = Silent Comm Error, 2 = EMEA Comm Error
- - Bit 4-8 : RFU

21.11.2.20 RETURN_CODE IDTechSDK.IDT_PIP.device_controlUserInterface (byte[] values, string ident = " ")

Control User Interface

Controls the User Interface: Display, Beep, LED

```
@param values Four bytes to control the user interface
Byte[0] = LCD Message
Messages 00-07 are normally controlled by the reader.
- 00h: Idle Message (Welcome, string ident = "")
- 01h: Present card (Please Present Card, string ident = "")
- 02h: Time Out or Transaction cancel (No Card, string ident = "")
- 03h: Transaction between reader and card is in the middle (Processing...)
- 04h: Transaction Pass (Thank You, string ident = "")
- 05h: Transaction Fail (Fail, string ident = "")
- 06h: Amount (Amount $ 0.00 Tap Card, string ident = "")
- 07h: Balance or Offline Available funds (Balance $ 0.00) Messages 08-0B are controlled by the terminal
- 08h: Insert or Swipe card (Use Chip & PIN, string ident = "")
- 09h: Try Again(Tap Again, string ident = "")
- 0Ah: Tells the customer to present only one card (Present 1 card only)
- 0Bh: Tells the customer to wait for authentication/authorization (Wait)
- FFh: indicates the command is setting the LED/Buzzer only.
Byte[1] = Beep Indicator
- 00h: No beep
- 01h: Single beep
- 02h: Double beep
- 03h: Three short beeps
- 04h: Four short beeps
- 05h: One long beep of 200 ms
- 06h: One long beep of 400 ms
- 07h: One long beep of 600 ms
- 08h: One long beep of 800 ms
Byte[2] = LED Number
- 00h: LED 0 (Power LED) 01h: LED 1
- 02h: LED 2
- 03h: LED 3
- FFh: All LEDs
Byte[3] = LED Status
- 00h: LED Off
- 01h: LED On
```

Parameters

<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.
--------------	--

Returns

RETURN_CODE: Values can be parsed with `errorCode.getErrorString(string ident = "")`

21.11.2.21 RETURN_CODE IDTechSDK.IDT_PIP.device_enablePassThrough (bool enablePassThrough, string ident = " ")

Enable Pass Through

Enables Pass Through Mode for direct communication with L1 interface (power on icc, send apdu, etc).

Parameters

<i>enablePassThrough</i>	true = pass through ON, false = pass through OFF
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with `errorCode.getErrorString(string ident = "")`

21.11.2.22 RETURN_CODE IDTechSDK.IDT_PIP.device_getAnyRKIStatus (bool *isMultiKey*, ref string *status*, ref Dictionary< string, RKI_KEY_TYPE > *keys*, string *ident* = " ")

Get RKI Status Polls the RKI servers to validate device RKI eligibility. Note: if device type is known in advance (production or demo device), it is more efficient to use `device_getRKIStatus` instead

Parameters

<i>isMultiKey</i>	True = MultiKey, False = SingleKey
<i>status</i>	RKI status returned as a string
<i>keys</i>	Returns all available keys
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with `errorCode.getErrorString()`. When no data is available, return code = RETURN_CODE_NO_DATA_AVAILABLE

21.11.2.23 RETURN_CODE IDTechSDK.IDT_PIP.device_getConfigurationFromMemory (ref string *json*, string *ident* = " ")

Read Device Configuration From Memory Retrieves the ViVOconfig configuration data from memory after a successful `device_readConfigurationToMemory`

Parameters

<i>json</i>	Returns the .json file data that can be saved to disk
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with `errorCode.getErrorString()`. When no data is available, return code = RETURN_CODE_NO_DATA_AVAILABLE

21.11.2.24 RETURN_CODE IDTechSDK.IDT_PIP.device_getFirmwareVersion (ref string *response*, string *ident* = " ")

Polls device for Firmware Version

Parameters

<i>response</i>	Response returned of Firmware Version
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with `errorCode.getErrorString(string ident = "")`

21.11.2.25 **RETURN_CODE** IDTechSDK.IDT_PIP.device_getMerchantRecord (int *index*, ref byte[] *record*, string *ident* = " ")

Get Merchant Record Gets the merchant record for ApplePay VAS

Parameters

<i>index</i>	Merchant Record index, valid values 1-6
<i>record</i>	Data returned containing 99 bytes: Byte 0 = Merchand Index Byte 1 = Merchant Enabled (1 = enabled, string ident = "") Byte 2 - 33 = Merchant Protocol Hash-256 value Byte 34 = Length of Merchant URL Bytes 35 - 99 = URL
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with `errorCode.getErrorString(string ident = "")`

21.11.2.26 **RETURN_CODE** IDTechSDK.IDT_PIP.device_getPIPMMode (ref int *mode*, string *ident* = " ")

Get Device Mode Gets the device operating mode for PIP

Parameters

<i>mode</i>	0 = HID, 1 = KB, 3 = HID+KB
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with `errorCode.getErrorString()`. When no data is available, return code = **RETURN_CODE_NO_DATA_AVAILABLE**

21.11.2.27 **RETURN_CODE** IDTechSDK.IDT_PIP.device_getRKIStatus (bool *isProd*, bool *isMultiKey*, ref string *status*, ref Dictionary< string, RKI_KEY_TYPE > *keys*, string *ident* = " ")

Get RKI Status From Specified Server Polls the RKI server to validate device RKI eligibility

Parameters

<i>isProd</i>	TRUE = poll production RKI server, FALSE = poll demo RKI server
<i>isMultiKey</i>	True = MultiKey, False = SingleKey
<i>status</i>	RKI status returned as a string
<i>keys</i>	Returns dictionary of available keys
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with `errorCode.getErrorString()`. When no data is available, return code = **RETURN_CODE_NO_DATA_AVAILABLE**

21.11.2.28 **RETURN_CODE** IDTechSDK.IDT_PIP.device_getTransactionResults (ref IDTTransactionData *results*, string *ident* = " ")

Get Transaction Results Gets the transaction results when the reader is functioning in "Auto Poll" mode

Parameters

<i>results</i>	The transaction results
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with `errorCode.getErrorString()`. When no data is available, return code = **RETURN_CODE_NO_DATA_AVAILABLE**

21.11.2.29 **RETURN_CODE** IDTechSDK.IDT_PIP.device_pingDevice (string *ident* = " ")

Ping Device

Pings the reader. If connected, returns success. Otherwise, returns timeout.

Parameters

<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.
--------------	--

Returns

RETURN_CODE: Values can be parsed with `errorCode.getErrorString(string ident = "")`

21.11.2.30 **RETURN_CODE** IDTechSDK.IDT_PIP.device_pollForToken (byte *seconds*, ref byte *card*, ref byte[] *serialNumber*, string *ident* = " ")

Poll for Token

Once Pass-Through Mode is started, ViVOpay will not poll for any cards until the "Poll for Token" command is received. This command tells ViVOpay to start polling for a Type A or Type B PICC until a PICC is detected or a timeout occurs.

This command automatically turns the RF Antenna on.

If a PICC is detected within the specified time limit, ViVOpay activates it and responds back to the terminal with card related data such as the Serial Number. If no PICC is detected within the specified time limit, ViVOpay stops polling and responds back indicating that no card was found. No card related data is returned in this case

Parameters

<i>timeout</i>	Timeout, in seconds to wait for card to be detected
----------------	---

Parameters

<i>card</i>	Card Type: <ul style="list-style-type: none"> • 00h None (Card Not Detected or Could not Activate, string ident = "") • 01h ISO 14443 Type A (Supports ISO 14443-4 Protocol, string ident = "") • 02h ISO 14443 Type B (Supports ISO 14443-4 Protocol, string ident = "") • 03h Mifare Type A (Standard, string ident = "") • 04h Mifare Type A (Ultralight, string ident = "") • 05h ISO 14443 Type A (Does not support ISO 14443-4 Protocol) • 06h ISO 14443 Type B (Does not support ISO 14443-4 Protocol) • 07h ISO 14443 Type A and Mifare (NFC phone, string ident = "")
<i>serialNumber</i>	Serial Number or the UID of the PICC
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with `device_getResponseCodeString`

21.11.2.31 `RETURN_CODE IDTechSDK.IDT_PIP.device_readConfigurationToMemory (string memo, Configs.InstallRules rules, List< Configs.DeviceCommand > cmds, string ident = " ", bool isForeground = false)`

Read Device Configuration To Memory Executes a ViVOconfig read of device settings and stores the results in memory

Once a ViVOconfig read starts, all messages will be returned to the callback as `DeviceState.ViVOconfig`. The ViVOconfig operation is complete when the return code from a `DeviceState.ViVOconfig` message is one of the following values:

- `RETURN_CODE_SDK_VIVOCONFIG_READ_CONFIG_SUCCESS` = Read process completed successfully. Execute `device_getConfigurationFromMemory` to retrieve the configuration data
- `RETURN_CODE_SDK_VIVOCONFIG_READ_CONFIG_FAILED` = Read process FAILED.

Parameters

<i>memo</i>	Optional: populates the memo field of the captured .json file
<i>rules</i>	Optional: populates the .json file with available install rules
<i>cmds</i>	Optional: populates the .json file with commands to execute
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.
<i>isForeground</i>	If TRUE, will block program until update complete. Otherwise, FALSE performs update on background.

Returns

RETURN_CODE: Values can be parsed with `errorCode.getErrorString()`. When no data is available, return code = `RETURN_CODE_NO_DATA_AVAILABLE`

21.11.2.32 **RETURN_CODE** IDTechSDK.IDT_PIP.device_RemoteKeyInjection (*RKI_KEY_TYPE type*, *string keyName*, *string ident = ""*, *bool performOnForeground = false*)

Remote Key Injection Performs a remote key injection for the device

Parameters

<i>type</i>	Remote Key Injection Type
<i>keyName</i>	Name of key (optional)
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.
<i>performOnForeground</i>	TRUE = block during process (default is false)

Returns

RETURN_CODE: Values can be parsed with `errorCode.getErrorString()`. When no data is available, return code = **RETURN_CODE_NO_DATA_AVAILABLE**

21.11.2.33 **RETURN_CODE** IDTechSDK.IDT_PIP.device_retrieveAIDList (*ref byte response[]*, *string ident = ""*)

Retrieve AID list

Returns all the AID names and their assigned groups installed on the terminal for CTLS/CONTACT.

Parameters

<i>response</i>	array of TLV data objects: FFE4 (group name) followed by 9F06 (AID), and DFEE4F (Interface 01 = CTLS, 02 = CONTACT, string ident = "")
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with `errorCode.getErrorString(string ident = "")`

21.11.2.34 **RETURN_CODE** IDTechSDK.IDT_PIP.device_sendConfiguration (*string filename*, *VC_OPERATION_TYPE type*, *bool matchFW*, *string ident = ""*, *bool isForeground = false*)

Send Configuration Executes a ViVOconfig update or verify to a device

Once a ViVOconfig write or verify starts, all messages will be returned to the callback as `DeviceState.ViVOconfig`. The ViVOconfig operation is complete when the return code from a `DeviceState.ViVOconfig` message is one of the following values:

- **RETURN_CODE_SDK_VIVOCONFIG_VERIFY_SUCCESS** = Verification process completed successfully. No differences found.
- **RETURN_CODE_SDK_VIVOCONFIG_VERIFY_SUCCESS_WITH_WARNING** = Verification process completed with warnings.
- **RETURN_CODE_SDK_VIVOCONFIG_VERIFY_FAILED** = Verification process FAILED
- **RETURN_CODE_SDK_VIVOCONFIG_WRITE_SUCCESS** = Write process completed successfully.
- **RETURN_CODE_SDK_VIVOCONFIG_WRITE_SUCCESS_WITH_WARNING** = Write process completed with warnings
- **RETURN_CODE_SDK_VIVOCONFIG_WRITE_FAILED** = Write process FAILED

Parameters

<i>filename</i>	The .json configuration file to update the device with NOTE: You can also use this parameter to pass the path of where the .json file is located at on local storage instead
<i>filename</i>	The name of the .json configuration file
<i>type</i>	VC_OPERATION_TYPE

- VC_OPERATION_TYPE_WRITE = Write File To Device, Hash must validate
- VC_OPERATION_TYPE_VERIFY = Verify Device With File
- VC_OPERATION_TYPE_WRITE_IGNORE_HASH = Write File To Device, Ignore Hash
- VC_OPERATION_TYPE_WRITE_FIX_HASH = Write File To Device, Fix Hash If Necessary

Parameters

<i>matchFW</i>	TRUE = Device FW must match file FW, FALSE = Don't validate device FW against file
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.
<i>isForeground</i>	If TRUE, will block program until update complete. Otherwise, FALSE performs update on background.

Returns

RETURN_CODE: Values can be parsed with `errorCode.getErrorString()`. When no data is available, return code = RETURN_CODE_NO_DATA_AVAILABLE

21.11.2.35 **RETURN_CODE** IDTechSDK.IDT_PIP.device_sendConfigurationFromZip (*byte[] zip*, *string filename*, *VC_OPERATION_TYPE type*, *bool matchFW*, *string ident* = "", *bool isForeground* = false)

Send Configuration From Zip Executes a ViVOconfig update or verify to a device using files from a .zip archive

Once a ViVOconfig write or update starts, all messages will be returned to the callback as DeviceState.ViVOconfig. The ViVOconfig operation is complete when the return code from a DeviceState.ViVOconfig message is one of the following values:

- RETURN_CODE_SDK_VIVOCONFIG_VERIFY_SUCCESS = Verification process completed successfully. No differences found.
- RETURN_CODE_SDK_VIVOCONFIG_VERIFY_SUCCESS_WITH_WARNING = Verification process completed with warnings.
- RETURN_CODE_SDK_VIVOCONFIG_VERIFY_FAILED = Verification process FAILED
- RETURN_CODE_SDK_VIVOCONFIG_WRITE_SUCCESS = Write process completed successfully.
- RETURN_CODE_SDK_VIVOCONFIG_WRITE_SUCCESS_WITH_WARNING = Write process completed with warnings
- RETURN_CODE_SDK_VIVOCONFIG_WRITE_FAILED = Write process FAILED

Parameters

<i>zip</i>	.Zip archive containing the .json configuration file and also any associated image assets NOTE: You can also use this parameter to pass the path of where the .zip file is located at on local storage instead
<i>filename</i>	The name of the .json configuration file
<i>type</i>	VC_OPERATION_TYPE

- VC_OPERATION_TYPE_WRITE = Write File To Device, Hash must validate
- VC_OPERATION_TYPE_VERIFY = Verify Device With File
- VC_OPERATION_TYPE_WRITE_IGNORE_HASH = Write File To Device, Ignore Hash
- VC_OPERATION_TYPE_WRITE_FIX_HASH = Write File To Device, Fix Hash If Necessary

Parameters

<i>matchFW</i>	TRUE = Device FW must match file FW, FALSE = Don't validate device FW against file
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.
<i>isForeground</i>	If TRUE, will block program until update complete. Otherwise, FALSE performs update on background.

Returns

RETURN_CODE: Values can be parsed with `errorCode.getErrorString()`. When no data is available, return code = RETURN_CODE_NO_DATA_AVAILABLE

21.11.2.36 RETURN_CODE IDTechSDK.IDT_PIP.device_sendDataCommand (string *cmd*, bool *calcLRC*, ref byte[] *response*, string *ident* = " ")

Send a data command to the device

Sends a command to the device.

Parameters

<i>cmd</i>	String representation of command to execute
<i>calcLRC</i>	If TRUE, this will wrap command with start/length/lrc/sum/end: '{STX}{Len_Low}{Len_High} data {CheckLRC} {CheckSUM} {ETX}'
<i>response</i>	Response data
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with `device_getResponseCodeString`

21.11.2.37 RETURN_CODE IDTechSDK.IDT_PIP.device_sendDataCommand_ext (string *cmd*, bool *calcLRC*, ref byte[] *response*, int *timeout*, bool *noResponse*, string *ident* = " ")

Send a data command to the device - extended

Sends a command to the device.

Parameters

<i>cmd</i>	String representation of command to execute
<i>calcLRC</i>	If TRUE, this will wrap command as NGA with start/length/lrc/sum/end: '{STX}{Len_Low}{Len_High} data {CheckLRC} {CheckSUM} {ETX}'
<i>response</i>	Response data
<i>timeout</i>	Timeout value waiting for response, in milliseconds (1000 = 1 second, string <i>ident</i> = "")
<i>noResponse</i>	if TRUE, this will not wait for a response and immediately return SUCCESS
<i>calcITP</i>	If TRUE, this will wrap command as ITP with start/end/lrc: '{STX} data {ETX}{CheckLRC}'
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString

21.11.2.38 RETURN_CODE IDTechSDK.IDT_PIP.device_sendVivoCommandP2 (byte *command*, byte *subCommand*, byte[] *data*, ref byte[] *response*, string *ident* = " ")

Send Vivo Command Protocol 2

Sends a protocol 2 command to Vivo readers (IDG/NEO, string ident = "")

Parameters

<i>command</i>	Command
<i>subCommand</i>	Sub-Command
<i>data</i>	Data. May be null
<i>response</i>	Response
<i>ip</i>	Optional IP
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with errorCode.getErrorString(string ident = "")

21.11.2.39 RETURN_CODE IDTechSDK.IDT_PIP.device_sendVivoCommandP2_ext (byte *command*, byte *subCommand*, byte[] *data*, ref byte[] *response*, int *timeout*, bool *noResponse*, string *ident* = " ")

Send Vivo Command Protocol 2 Extended

Sends a protocol 2 command to Vivo readers (IDG/NEO, string ident = "")

Parameters

<i>command</i>	Command
<i>subCommand</i>	Sub-Command
<i>data</i>	Data. May be null
<i>response</i>	Response
<i>timeout</i>	Timeout, in milliseconds (3000 = 3 seconds, string ident = "")
<i>noResponse</i>	TRUE = don't wait for response, FALSE = wait for response defined by timeout
<i>ip</i>	Optional IP
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with errorCode.getErrorString(string ident = "")

21.11.2.40 RETURN_CODE IDTechSDK.IDT_PIP.device_setBurstMode (byte *mode*, string *ident* = " ")

Send Burst Mode

Sets the burst mode for the device.

Parameters

<i>mode</i>	0 = OFF, 1 = Always On, 2 = Auto Exit
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with `errorCode.getErrorString(string ident = "")`

21.11.2.41 RETURN_CODE IDTechSDK.IDT_PIP.device_setMerchantRecord (int *index*, bool *enabled*, string *merchantID*, string *merchantURL*, string *ident* = " ")

Set Merchant Record Sets the merchant record for ApplePay VAS

Parameters

<i>index</i>	Merchant Record index, valid values 1-6
<i>enabled</i>	Merchant Enabled/Valid flag
<i>merchantID</i>	Merchant unique identifier registered with Apple. Example com.idtechproducts.applePay
<i>merchantURL</i>	Merchant URL, when applicable
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with `errorCode.getErrorString(string ident = "")`

21.11.2.42 RETURN_CODE IDTechSDK.IDT_PIP.device_setPIPMODE (int *mode*, string *ident* = " ")

Set Device Mode Sets the device operating mode for PIP

Parameters

<i>mode</i>	0 = HID, 1 = KB, 3 = HID+KB
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with `errorCode.getErrorString()`. When no data is available, return code = RETURN_CODE_NO_DATA_AVAILABLE

21.11.2.43 RETURN_CODE IDTechSDK.IDT_PIP.device_setPollMode (byte *mode*, string *ident* = " ")

Send Poll Mode

Sets the poll mode for the device. Auto Poll keeps reader active, Poll On Demand only polls when requested by terminal

Parameters

<i>mode</i>	0 = Auto Poll, 1 = Poll On Demand
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with `errorCode.getErrorString(string ident = "")`

21.11.2.44 RETURN_CODE IDTechSDK.IDT_PIP.device_startTransaction (double *amount*, double *amtOther*, int *exponent*, int *type*, int *timeout*, byte[] *tags*, byte[] *options* = null, bool *isFastEMV* = false, int *transMode* = 0, string *ident* = "")

Start CTLS Transaction Request

The tags will be returned in the callback routine.

Parameters

<i>amount</i>	Transaction amount value (tag value 9F02, string ident = "")
<i>amtOther</i>	Other amount value, if any (tag value 9F03, string ident = "")
<i>type</i>	Transaction type (tag value 9C).
<i>timeout</i>	Timeout value in seconds.
<i>tags</i>	Any other tags to be included in the request. Passed as TLV data. Example, tag 9F02 with amount 0x000000000100 would be 0x9F0206000000000100 If tags 9F02 (amount), 9F03 (other amount), or 9C (transaction type) are included, they will take priority over these values supplied as individual parameters to this method.
<i>options</i>	Not used for PIP. Parameter included for API compability with other IDTech Devices
<i>isFastEMV</i>	Not used for PIP. Parameter included for API compability with other IDTech Devices
<i>transMode</i>	Not used for PIP. Parameter included for API compability with other IDTech Devices
<i>ip</i>	Not used for PIP. Parameter included for API compability with other IDTech Devices
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with `errorCode.getErrorString()`

NOTE ON APPLEPAY VAS: To enable ApplePay VAS, first a merchant record must be defined in one of the six available index positions (1-6) using `device_setMerchantRecord`, then container tag FFEE06 must be sent as part of the additional tags parameter of `ctls_startTransaction`. Tag FFEE06 must contain tag 9F26 and 9F22, and can optionally contain tags 9F2B and DFO1. Example FFEE06189F220201009F2604000000009F2B050100000000DF010101 9F22 = two bytes = ApplePay Terminal Application Version Number. Hard defined as 0100 for now. (required, string ident = "") 9F26 = four bytes = ApplePay Terminal Capabilities Information (required, string ident = "")

- Byte 1 = RFU
- Byte 2 = Terminal Type
- - Bit 8 = VAS Support (1=on, 0 = off, string ident = "")
- - Bit 7 = Touch ID Required (1=on, 0 = off, string ident = "")
- - Bit 6 = RFU
- - Bit 5 = RFU
- - Bit 1,2,3,4
- - - 0 = Payment Terminal
- - - 1 = Transit Terminal
- - - 2 = Access Terminal

- - - 3 = Wireless Handoff Terminal
- - - 4 = App Handoff Terminal
- - - 15 = Other Terminal
- Byte 3 = RFU
- Byte 4 = Terminal Mode
- - 0 = ApplePay VAS OR ApplePay
- - 1 = ApplePay VAS AND ApplePay
- - 2 = ApplePay VAS ONLY
- - 3 = ApplePay ONLY 9F2B = 5 bytes = ApplePay VAS Filter. Each byte filters for that specific merchant index (optional, string ident = "") DF01 = 1 byte = ApplePay VAS Protocol. (optional, string ident = "")
- - Bit 1 : 1 = URL VAS, 0 = Full VAS
- - Bit 2 : 1 = VAS Beeps, 0 = No VAS Beeps
- - Bit 3 : 1 = Silent Comm Error, 2 = EMEA Comm Error
- - Bit 4-8 : RFU

21.11.2.45 **RETURN_CODE** IDTechSDK.IDT_PIP.device_updateDeviceFirmware (byte[] *firmwareData*, string *ident* = " ")

Update Firmware

Updates the firmware .

Parameters

<i>firmwareData</i>	Signed binary data of a firmware file provided by IDTech
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with `errorCode.getErrorString(string ident = "")`

After you pass the `firmwareData` file, a new thread will start to execute the firmware download. You will receive status of the progress through callbacks to the `IDTechSDK.Callback()` delegate. The following parameters will be passed back:

- `sender` = `IDT_DEVICE_Types.IDT_DEVICE_AUGUSTA`
- `state` = `DeviceState.FirmwareUpdate`
- `transactionResultCode` = status of the firmware update (starting, entering bootloader, applying update, block success, firmware success, string ident = "")
- `data` = File Progress. Four bytes, with bytes [0][1] = current block, and bytes [2][3] = total blocks. 0x00030010 = block 3 of 16

Example code starting a firmware update

```
OpenFileDialog diag = new OpenFileDialog( ident);
diag.Filter = "NGA FW Files|*.fm";

if (diag.ShowDialog() == DialogResult.OK, string ident = "")
```

```

{
    byte[] file = File.ReadAllBytes(diag.FileName, ident);
    RETURN_CODE rt = IDT_Device.SharedController.device_updateDeviceFirmware(file, ident);
    if (rt == RETURN_CODE.RETURN_CODE_DO_SUCCESS, string ident = "")
    {
        //Was a success
    }
    else
    {
        //Error starting firmware download
    }
}

```

Example monitoring firmware update status / success

```

private void MessageCallBack(IDTechSDK.IDT_DEVICE_Types type, DeviceState state, byte[] data,
    IDTTransactionData cardData, EMV_Callback emvCallback, RETURN_CODE transactionResultCode, string ident = "")
{
    switch (state, string ident = "")
    {
        case DeviceState.FirmwareUpdate:
            switch (transactionResultCode, string ident = "")
            {
                case RETURN_CODE.RETURN_CODE_FW_STARTING_UPDATE:
                    SetOutputText("Starting Firmware Update\n", ident);
                    break;
                case RETURN_CODE.RETURN_CODE_DO_SUCCESS:
                    SetOutputText("Firmware Update Successful\n", ident);
                    break;
                case RETURN_CODE.RETURN_CODE_APPLYING_FIRMWARE_UPDATE:
                    SetOutputText("Applying Firmware Update...\n", ident);
                    break;
                case RETURN_CODE.RETURN_CODE_ENTERING_BOOTLOADER_MODE:
                    SetOutputText("Entering Bootloader Mode...\n", ident);
                    break;
                case RETURN_CODE.RETURN_CODE_BLOCK_TRANSFER_SUCCESS:

                    int start = data[0] * 0x100 + data[1];
                    int end = data[2] * 0x100 + data[3];

                    SetOutputText("Sent block " + start.ToString() + " of " + end.ToString() + "\n", ident);

                    break;
                default:
                    SetOutputText("Firmware Update Error Code: " + "0x" + String.Format("{0:X}", (ushort)
                        transactionResultCode) + ": " + IDTechSDK.errorCode.getErrorString(transactionResultCode) + "\r\n",
                        ident);
                    break;
            }
            break;
    }
}

```

21.11.2.46 RETURN_CODE IDTechSDK.IDT_PIP.device_updateFirmwareFromZip (byte[] zipfile, string ident = "", bool isForeground = false)

Update Firmware From Zip

Performs one or more device firmware updates from firmware files passed as a compressed archive with a valid configuration file (.json format)

Parameters

<i>zipfile</i>	Zip file containing on or more firmware update files and a .json configuration file
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.
<i>isForeground</i>	If TRUE, will block program until firmware update complete. Otherwise, FALSE performs FW update on background.

Returns

RETURN_CODE: Values can be parsed with `errorCode.getErrorString(string ident = "")`

21.11.2.47 `RETURN_CODE IDTechSDK.IDT_PIP.felica_requestService (byte[] nodeCode, ref byte[] response, string ident = " ")`

FeliCa Request Service

Perform functions a Felica Request Service

NOTE: The reader must be in Pass Through Mode for FeliCa commands to work.

Parameters

<i>nodeCode</i>	Node Code
<i>response</i>	Response as explained in FeliCA Lite-S User's Manual
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with `errorCode.getErrorString(string ident = "")`

21.11.2.48 `static int IDTechSDK.IDT_PIP.getCommandTimeout (string ident = " ") [static]`

Get Command Timeout

Gets the default timeout (in milliseconds) waiting for a blocking command response

Return values

<i>time</i>	Time
-------------	------

21.11.2.49 `static void IDTechSDK.IDT_PIP.lcd_retrieveMessage (DisplayMessages.DISPLAY_MESSAGE_LANGUAGE lang, DisplayMessages.DISPLAY_MESSAGE_IDENTIFIER id, ref string line1, ref string line2) [static]`

Retrieve LCD Message

Returns the string value for a message ID returned for LCD messaging

Parameters

<i>lang</i>	Language.
<i>id</i>	Message ID
<i>line1</i>	Line 1 string value
<i>line2</i>	Line 2 string value

21.11.2.50 `static String IDTechSDK.IDT_PIP.SDK_Version () [static]`

SDK Version

- All Devices

Returns the current version of SDK

Returns

Framework version

21.11.2.51 `static void IDTechSDK.IDT_PIP.setCallback (Callback my_Callback) [static]`

Set Callback

Sets the class callback

21.11.2.52 `static void IDTechSDK.IDT_PIP.setCallback (IntPtr my_Callback, SynchronizationContext context) [static]`

Set Callback

Sets the class callback

Parameters

<i>my_Callback</i>	The callback function to receive the response message from device. defined as follows. public unsafe delegate void MFCCallBack(Parameters parameters, ident);
<i>context</i>	The context of the UI thread

21.11.2.53 `static void IDTechSDK.IDT_PIP.setCommandTimeout (int milliseconds, string ident = " ") [static]`

Set Command Timeout

Sets the default timeout (in milliseconds) waiting for a blocking command response

Parameters

<i>milliseconds</i>	Time
---------------------	------

21.11.3 Property Documentation

21.11.3.1 `IDT_PIP IDTechSDK.IDT_PIP.SharedController [static],[get]`

Singleton Instance

Establishes an singleton instance of [IDT_PIP](#) class.

Returns

Instance of [IDT_PIP](#)

The documentation for this class was generated from the following file:

- /Users/randy/Repo/universal-sdk/IDTechSDK_STD/DeviceHeaders/IDT_PIP.cs

21.12 IDTechSDK.IDT_SecureKey Class Reference

Public Member Functions

- RETURN_CODE [msr_switchUSBInterfaceMode](#) (bool blsUSBKeyboardMode)
- RETURN_CODE [msr_switchUSBInterfaceMode](#) (bool blsUSBKeyboardMode, ref string ident="")
- RETURN_CODE [device_getOutputType](#) (bool isSwipe, ref int response, string ident="")
- RETURN_CODE [device_setOutputType](#) (bool isSwipe, int value, string ident="")
- RETURN_CODE [config_getModelNumber](#) (ref string response, string ident="")
- RETURN_CODE [device_sendVivoCommandP3](#) (byte command, byte subCommand, byte[] data, ref byte[] response, string ident="")
- RETURN_CODE [device_sendVivoCommandP3_ext](#) (byte command, byte subCommand, byte[] data, ref byte[] response, int timeout, bool noResponse, string ident="")
- RETURN_CODE [device_sendVivoCommandP2](#) (byte command, byte subCommand, byte[] data, ref byte[] response, string ident="")
- RETURN_CODE [device_StartRKI](#) (int type, string ident="")
- RETURN_CODE [device_sendVivoCommandP2_ext](#) (byte command, byte subCommand, byte[] data, ref byte[] response, int timeout, bool noResponse, string ident="")
- RETURN_CODE [config_getSerialNumber](#) (ref string response, string ident="")
- RETURN_CODE [device_getFirmwareVersion](#) (ref string response, string ident="")
- string [device_getResponseCodeString](#) (RETURN_CODE eCode)
- RETURN_CODE [device_sendDataCommand](#) (string cmd, bool calcLRC, ref byte[] response, string ident="")
- RETURN_CODE [device_sendDataCommand_ext](#) (string cmd, bool calcLRC, ref byte[] response, int timeout, bool noResponse, string ident="")
- RETURN_CODE [device_capturePINFromLast12](#) (int min, int max, int timeout, string last12, string ident="")
- RETURN_CODE [device_sendPAE](#) (string command, ref string response, int timeout, string ident="")
- RETURN_CODE [device_getRKIStatus](#) (bool isProd, bool isMultiKey, ref string status, ref Dictionary< string, RKI_KEY_TYPE > keys, string ident="")
- RETURN_CODE [device_getAnyRKIStatus](#) (bool isMultiKey, ref string status, ref Dictionary< string, RKI_KEY_TYPE > keys, string ident="")
- RETURN_CODE [device_updateFirmwareFromZip](#) (byte[] zipfile, string ident="", bool isForeground=false)
- RETURN_CODE [device_sendConfigurationFromZip](#) (byte[] zip, string filename, VC_OPERATION_TYPE type, bool matchFW, string ident="", bool isForeground=false)
- RETURN_CODE [device_sendConfiguration](#) (string filename, VC_OPERATION_TYPE type, bool matchFW, string ident="", bool isForeground=false)
- RETURN_CODE [device_readConfigurationToMemory](#) (string memo, Configs.InstallRules rules, List< Configs.DeviceCommand > cmds, string ident="", bool isForeground=false)
- RETURN_CODE [device_getConfigurationFromMemory](#) (ref string json, string ident="")
- RETURN_CODE [device_RemoteKeyInjection](#) (RKI_KEY_TYPE type, string keyName, string ident="", bool performOnForeground=false)

Static Public Member Functions

- static void [initSC](#) ()
- static int [getCommandTimeout](#) (string ident="")
- static void [setCommandTimeout](#) (int milliseconds, string ident="")
- static void [setCallback](#) (Callback my_Callback)
- static void [setCallback](#) (IntPtr my_Callback, SynchronizationContext context)
- static String [SDK_Version](#) ()
- static void [lcd_retrieveMessage](#) (DisplayMessages.DISPLAY_MESSAGE_LANGUAGE lang, DisplayMessages.DISPLAY_MESSAGE_IDENTIFIER id, ref string line1, ref string line2)

Properties

- static [IDT_SecureKey SharedController](#) [get]

21.12.1 Detailed Description

Class for SecureKey MSR reder

21.12.2 Member Function Documentation

21.12.2.1 RETURN_CODE IDTechSDK.IDT_SecureKey.config_getModelNumber (ref string *response*, string *ident* = " ")

Polls device for Model Number

Parameters

<i>response</i>	Returns Model Number
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString

21.12.2.2 RETURN_CODE IDTechSDK.IDT_SecureKey.config_getSerialNumber (ref string *response*, string *ident* = " ")

Polls device for Serial Number

Parameters

<i>response</i>	Returns Serial Number
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString

21.12.2.3 RETURN_CODE IDTechSDK.IDT_SecureKey.device_capturePINFromLast12 (int *min*, int *max*, int *timeout*, string *last12*, string *ident* = " ")

Capture PIN From Last 12

Captures a PIN from provided last 12 PAN.

Parameters

<i>min</i>	Minimum PIN entry
<i>max</i>	Maximum PIN Entry
<i>timeout</i>	Timeout
<i>last12</i>	Last 12 digits of PAN
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString

21.12.2.4 **RETURN_CODE** IDTechSDK.IDT_SecureKey.device_getAnyRKIStatus (bool *isMultiKey*, ref string *status*, ref Dictionary< string, RKI_KEY_TYPE > *keys*, string *ident* = " ")

Get RKI Status Polls the RKI servers to validate device RKI eligibility. Note: if device type is known in advance (production or demo device), it is more efficient to use device_getRKIStatus instead

Parameters

<i>isMultiKey</i>	True = MultiKey, False = SingleKey
<i>status</i>	RKI status returned as a string
<i>keys</i>	Returns all available keys
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with errorCode.getErrorString(). When no data is available, return code = RETURN_CODE_NO_DATA_AVAILABLE

21.12.2.5 **RETURN_CODE** IDTechSDK.IDT_SecureKey.device_getConfigurationFromMemory (ref string *json*, string *ident* = " ")

Read Device Configuration From Memory Retrieves the ViVOconfig configuration data from memory after a successful device_readConfigurationToMemory

Parameters

<i>json</i>	Returns the .json file data that can be saved to disk
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with errorCode.getErrorString(). When no data is available, return code = RETURN_CODE_NO_DATA_AVAILABLE

21.12.2.6 **RETURN_CODE** IDTechSDK.IDT_SecureKey.device_getFirmwareVersion (ref string *response*, string *ident* = " ")

Polls device for Firmware Version

Parameters

<i>response</i>	Response returned of Firmware Version
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString

21.12.2.7 **RETURN_CODE** IDTechSDK.IDT_SecureKey.device_getOutputType (bool *isSwipe*, ref int *response*, string *ident* = " ")

Get Captured Data Output Type

Parameters

<i>isSwipe</i>	TRUE = settings for Swipe, FALSE = settings for MSR (only applicable for Original/Enhanced formats, string ident = "") NOTE: IF XML is chosen, this parameter is ignored
<i>response</i>	<ul style="list-style-type: none"> • 0 : Original Format • 1 : Enhanced Format • 2 : XML Format
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString

21.12.2.8 string IDTechSDK.IDT_SecureKey.device_getResponseCodeString (RETURN_CODE eCode)

Get the description of response result.

Parameters

<i>eCode</i>	the response result.
--------------	----------------------

Return values

<i>the</i>	string for description of response result
------------	---

- case 0: "no error, beginning task";
- case 1: "no response from reader";
- case 2: "invalid response data";
- case 3: "time out for task or CMD";
- case 4: "wrong parameter";
- case 5: "SDK is doing MSR or ICC task";
- case 6: "SDK is doing PINPad task";
- case 7: "SDK is doing CTLS task";
- case 8: "SDK is doing EMV task";
- case 9: "SDK is doing Other task";
- case 10: "err response or data";
- case 11: "no reader attached";
- case 12: "mono audio is enabled";
- case 13: "did connection";
- case 14: "audio volume is too low";

- case 15: "task or CMD be canceled";
- case 16: "UF wrong string format";
- case 17: "UF file not found";
- case 18: "UF wrong file format";
- case 19: "Attempt to contact online host failed";
- case 20: "Attempt to perform RKI failed";
- case 0x300: "Key Type(TDES) of Session Key is not same as the related Master Key.";
- case 0x400: "Related Key was not loaded.";
- case 0x500: "Key Same.";
- case 0x501: "Key is all zero";
- case 0x502: "TR-31 format error";
- case 0x702: "PAN is Error Key.";
- case 0x705: "No Internal MSR PAN (or Internal MSR PAN is erased timeout)";
- case 0X0C01: "Incorrect Frame Tag";
- case 0X0C02: "Incorrect Frame Type";
- case 0X0C03: "Unknown Frame Type";
- case 0X0C04: "Unknown Command";
- case 0X0C05: "Unknown Sub-Command";
- case 0X0C06: "CRC Error";
- case 0X0C07: "Failed";
- case 0X0C08: "Timeout";
- case 0X0C0A: "Incorrect Parameter";
- case 0X0C0B: "Command Not Supported";
- case 0X0C0C: "Sub-Command Not Supported";
- case 0X0C0D: "Parameter Not Supported / Status Abort Command";
- case 0X0C0F: "Sub-Command Not Allowed";
- case 0X0D01: "Incorrect Header Tag";
- case 0X0D02: "Unknown Command";
- case 0X0D03: "Unknown Sub-Command";
- case 0X0D04: "CRC Error in Frame";
- case 0X0D05: "Incorrect Parameter";
- case 0X0D06: "Parameter Not Supported";
- case 0X0D07: "Mal-formatted Data";
- case 0X0D08: "Timeout";
- case 0X0D0A: "Failed / NACK";
- case 0X0D0B: "Command not Allowed";

- case 0X0D0C: "Sub-Command not Allowed";
- case 0X0D0D: "Buffer Overflow (Data Length too large for reader buffer)";
- case 0X0D0E: "User Interface Event";
- case 0X0D11: "Communication type not supported, VT-1, burst, etc.";
- case 0X0D12: "Secure interface is not functional or is in an intermediate state.";
- case 0X0D13: "Data field is not mod 8";
- case 0X0D14: "Pad 0x80 not found where expected";
- case 0X0D15: "Specified key type is invalid";
- case 0X0D1: "Could not retrieve key from the SAM(InitSecureComm)";
- case 0X0D17: "Hash code problem";
- case 0X0D18: "Could not store the key into the SAM(InstallKey)";
- case 0X0D19: "Frame is too large";
- case 0X0D1A: "Unit powered up in authentication state but POS must resend the InitSecureComm command";
- case 0X0D1B: "The EEPROM may not be initialized because SecCommInterface does not make sense";
- case 0X0D1C: "Problem encoding APDU";
- case 0X0D20: "Unsupported Index(ILM) SAM Transceiver error – problem communicating with the SAM(Key Mgr)";
- case 0X0D2: "Unexpected Sequence Counter in multiple frames for single bitmap(ILM) Length error in data returned from the SAM(Key Mgr)";
- case 0X0D22: "Improper bit map(ILM)";
- case 0X0D23: "Request Online Authorization";
- case 0X0D24: "ViVOCard3 raw data read successful";
- case 0X0D25: "Message index not available(ILM) ViVOcomm activate transaction card type(ViVOcomm)";
- case 0X0D26: "Version Information Mismatch(ILM)";
- case 0X0D27: "Not sending commands in correct index message index(ILM)";
- case 0X0D28: "Time out or next expected message not received(ILM)";
- case 0X0D29: "ILM languages not available for viewing(ILM)";
- case 0X0D2A: "Other language not supported(ILM)";
- case 0X0D41: "Unknown Error from SAM";
- case 0X0D42: "Invalid data detected by SAM";
- case 0X0D43: "Incomplete data detected by SAM";
- case 0X0D44: "Reserved";
- case 0X0D45: "Invalid key hash algorithm";
- case 0X0D46: "Invalid key encryption algorithm";
- case 0X0D47: "Invalid modulus length";
- case 0X0D48: "Invalid exponent";

- case 0X0D49: "Key already exists";
- case 0X0D4A: "No space for new RID";
- case 0X0D4B: "Key not found";
- case 0X0D4C: "Crypto not responding";
- case 0X0D4D: "Crypto communication error";
- case 0X0D4E: "Module-specific error for Key Manager";
- case 0X0D4F: "All key slots are full (maximum number of keys has been installed)";
- case 0X0D50: "Auto-Switch OK";
- case 0X0D51: "Auto-Switch failed";
- case 0X0D90: "Account DUKPT Key not exist";
- case 0X0D91: "Account DUKPT Key KSN exhausted";
- case 0x0D00: "This Key had been loaded.";
- case 0x0E00: "Base Time was loaded.";
- case 0x0F00: "Encryption Or Decryption Failed.";
- case 0x1000: "Battery Low Warning (It is High Priority Response while Battery is Low.)";
- case 0x1800: "Send "Cancel Command" after send "Get Encrypted PIN" &"Get Numeric "& "Get Amount""";
- case 0x1900: "Press "Cancel" key after send "Get Encrypted PIN" &"Get Numeric "& "Get Amount""";
- case 0x30FF: "Security Chip is not connect";
- case 0x3000: "Security Chip is deactivation & Device is In Removal Legally State.";
- case 0x3101: "Security Chip is activation & Device is In Removal Legally State.";
- case 0x5500: "No Admin DUKPT Key.";
- case 0x5501: "Admin DUKPT Key STOP.";
- case 0x5502: "Admin DUKPT Key KSN is Error.";
- case 0x5503: "Get Authentication Code1 Failed.";
- case 0x5504: "Validate Authentication Code Error.";
- case 0x5505: "Encrypt or Decrypt data failed.";
- case 0x5506: "Not Support the New Key Type.";
- case 0x5507: "New Key Index is Error.";
- case 0x5508: "Step Error.";
- case 0x5509: "KSN Error";
- case 0x550A: "MAC Error.";
- case 0x550B: "Key Usage Error.";
- case 0x550C: "Mode Of Use Error.";
- case 0x550F: "Other Error.";
- case 0x6000: "Save or Config Failed / Or Read Config Error.";
- case 0x6200: "No Serial Number.";

- case 0x6900: "Invalid Command - Protocol is right, but task ID is invalid.";
- case 0x6A01: "Unsupported Command – Protocol and task ID are right, but command is invalid – In this State";
- case 0x6A00: "Unsupported Command - Protocol and task ID are right, but command is invalid.";
- case 0x6B00: "Unknown parameter in command - Protocol task ID and command are right, but parameter is invalid.";
- case 0x6C00: "Unknown parameter in command – Protocol task ID and command are right, but length is out of the requirement.";
- case 0x7200: "Device is suspend (MKSK suspend or press password suspend).";
- case 0x7300: "PIN DUKPT is STOP (21 bit 1).";
- case 0x7400: "Device is Busy.";
- case 0xE100: "Can not enter sleep mode";
- case 0xE200: "File has existed";
- case 0xE300: "File has not existed";
- case 0xE313: "IO line low -- Card error after session start";
- case 0xE400: "Open File Error";
- case 0xE500: "SmartCard Error";
- case 0xE600: "Get MSR Card data is error";
- case 0xE700: "Command time out";
- case 0xE800: "File read or write is error";
- case 0xE900: "Active 1850 error!";
- case 0xEA00: "Load bootloader error";
- case 0xEF00: "Protocol Error- STX or ETX or check error.";
- case 0xEB00: "Picture is not exist";
- case 0x2C02: "No Microprocessor ICC seated";
- case 0x2C06: "no card seated to request ATR";
- case 0x2D01: "Card Not Supported,";
- case 0x2D03: "Card Not Supported, wants CRC";
- case 0x690D: "Command not supported on reader without ICC support";
- case 0x8100: "ICC error time out on power-up";
- case 0x8200: "invalid TS character received - Wrong operation step";
- case 0x8300: "Decode MSR Error";
- case 0x8400: "TriMagII no Response";
- case 0x8500: "No Swipe MSR Card";
- case 0x8510: "No Financial Card";
- case 0x8600: "Unsupported F, D, or combination of F and D";
- case 0x8700: "protocol not supported EMV TD1 out of range";

- case 0x8800: "power not at proper level";
- case 0x8900: "ATR length too long";
- case 0x8B01: "EMV invalid TA1 byte value";
- case 0x8B02: "EMV TB1 required";
- case 0x8B03: "EMV Unsupported TB1 only 00 allowed";
- case 0x8B04: "EMV Card Error, invalid BWI or CWI";
- case 0x8B06: "EMV TB2 not allowed in ATR";
- case 0x8B07: "EMV TC2 out of range";
- case 0x8B08: "EMV TC2 out of range";
- case 0x8B09: "per EMV96 TA3 must be > 0xF";
- case 0x8B10: "ICC error on power-up";
- case 0x8B11: "EMV T=1 then TB3 required";
- case 0x8B12: "Card Error, invalid BWI or CWI";
- case 0x8B13: "Card Error, invalid BWI or CWI";
- case 0x8B17: "EMV TC1/TB3 conflict-";
- case 0x8B20: "EMV TD2 out of range must be T=1";
- case 0x8C00: "TCK error";
- case 0xA304: "connector has no voltage setting";
- case 0xA305: "ICC error on power-up invalid (SBLK(IFSD) exchange";
- case 0xE301: "ICC error after session start";
- case 0xFF00: "Request to go online";
- case 0xFF01: "EMV: Accept the offline transaction";
- case 0xFF02: "EMV: Decline the offline transaction";
- case 0xFF03: "EMV: Accept the online transaction";
- case 0xFF04: "EMV: Decline the online transaction";
- case 0xFF05: "EMV: Application may fallback to magstripe technology";
- case 0xFF06: "EMV: ICC detected tah the conditions of use are not satisfied";
- case 0xFF07: "EMV: ICC didn't accept transaction";
- case 0xFF08: "EMV: Transaction was cancelled";
- case 0xFF09: "EMV: Application was not selected by kernel or ICC format error or ICC missing data error";
- case 0xFF0A: "EMV: Transaction is terminated";
- case 0xFF0B: "EMV: Other EMV Error";
- case 0xFFFF: "NO RESPONSE";
- case 0xF002: "ICC communication timeout";
- case 0xF003: "ICC communication Error";
- case 0xF00F: "ICC Card Seated and Highest Priority, disable MSR work request";

- case 0xF200: "AID List / Application Data is not exist";
- case 0xF201: "Terminal Data is not exist";
- case 0xF202: "TLV format is error";
- case 0xF203: "AID List is full";
- case 0xF204: "Any CA Key is not exist";
- case 0xF205: "CA Key RID is not exist";
- case 0xF206: "CA Key Index it not exist";
- case 0xF207: "CA Key is full";
- case 0xF208: "CA Key Hash Value is Error";
- case 0xF209: "Transaction format error";
- case 0xF20A: "The command will not be processing";
- case 0xF20B: "CRL is not exist";
- case 0xF20C: "CRL number exceed max number";
- case 0xF20D: "Amount,Other Amount,Trasaction Type are missing";
- case 0xF20E: "The Identification of algorithm is mistake";
- case 0xF20F: "No Financial Card";
- case 0xF210: "In Encrypt Result state, TLV total Length is greater than Max Length";
- case 0x1001: "INVALID ARG";
- case 0x1002: "FILE_OPEN_FAILED";
- case 0x1003: "FILE OPERATION_FAILED";
- case 0x2001: "MEMORY_NOT_ENOUGH";
- case 0x3002: "SMARTCARD_FAIL";
- case 0x3003: "SMARTCARD_INIT_FAILED";
- case 0x3004: "FALLBACK_SITUATION";
- case 0x3005: "SMARTCARD_ABSENT";
- case 0x3006: "SMARTCARD_TIMEOUT";
- case 0x5001: "EMV_PARSING_TAGS_FAILED";
- case 0x5002: "EMV_DUPLICATE_CARD_DATA_ELEMENT";
- case 0x5003: "EMV_DATA_FORMAT_INCORRECT";
- case 0x5004: "EMV_NO_TERM_APP";
- case 0x5005: "EMV_NO_MATCHING_APP";
- case 0x5006: "EMV_MISSING_MANDATORY_OBJECT";
- case 0x5007: "EMV_APP_SELECTION_RETRY";
- case 0x5008: "EMV_GET_AMOUNT_ERROR";
- case 0x5009: "EMV_CARD_REJECTED";
- case 0x5010: "EMV_AIP_NOT_RECEIVED";

- case 0x5011: "EMV_AFL_NOT_RECEIVED";
- case 0x5012: "EMV_AFL_LEN_OUT_OF_RANGE";
- case 0x5013: "EMV_SFI_OUT_OF_RANGE";
- case 0x5014: "EMV_AFL_INCORRECT";
- case 0x5015: "EMV_EXP_DATE_INCORRECT";
- case 0x5016: "EMV_EFF_DATE_INCORRECT";
- case 0x5017: "EMV_ISS_COD_TBL_OUT_OF_RANGE";
- case 0x5018: "EMV_CRYPTOGAM_TYPE_INCORRECT";
- case 0x5019: "EMV_PSE_NOT_SUPPORTED_BY_CARD";
- case 0x5020: "EMV_USER_SELECTED_LANGUAGE";
- case 0x5021: "EMV_SERVICE_NOT_ALLOWED";
- case 0x5022: "EMV_NO_TAG_FOUND";
- case 0x5023: "EMV_CARD_BLOCKED";
- case 0x5024: "EMV_LEN_INCORRECT";
- case 0x5025: "CARD_COM_ERROR";
- case 0x5026: "EMV_TSC_NOT_INCREASED";
- case 0x5027: "EMV_HASH_INCORRECT";
- case 0x5028: "EMV_NO_ARC";
- case 0x5029: "EMV_INVALID_ARC";
- case 0x5030: "EMV_NO_ONLINE_COMM";
- case 0x5031: "TRAN_TYPE_INCORRECT";
- case 0x5032: "EMV_APP_NO_SUPPORT";
- case 0x5033: "EMV_APP_NOT_SELECT";
- case 0x5034: "EMV_LANG_NOT_SELECT";
- case 0x5035: "EMV_NO_TERM_DATA";
- case 0x6001: "CVM_TYPE_UNKNOWN";
- case 0x6002: "CVM_AIP_NOT_SUPPORTED";
- case 0x6003: "CVM_TAG_8E_MISSING";
- case 0x6004: "CVM_TAG_8E_FORMAT_ERROR";
- case 0x6005: "CVM_CODE_IS_NOT_SUPPORTED";
- case 0x6006: "CVM_COND_CODE_IS_NOT_SUPPORTED";
- case 0x6007: "NO_MORE_CVM";
- case 0x6008: "PIN_BYPASSED_BEFORE";
- case 0x7001: "PK_BUFFER_SIZE_TOO_BIG";
- case 0x7002: "PK_FILE_WRITE_ERROR";
- case 0x7003: "PK_HASH_ERROR";

- case 0x8001: "NO_CARD_HOLDER_CONFIRMATION";
- case 0x8002: "GET_ONLINE_PIN";
- case 0xD000: "Data not exist";
- case 0xD001: "Data access error";
- case 0xD100: "RID not exist";
- case 0xD101: "RID existed";
- case 0xD102: "Index not exist";
- case 0xD200: "Maximum exceeded";
- case 0xD201: "Hash error";
- case 0xD205: "System Busy";
- case 0x0E01: "Unable to go online";
- case 0x0E02: "Technical Issue";
- case 0x0E03: "Declined";
- case 0x0E04: "Issuer Referral transaction";
- case 0x0F01: "Decline the online transaction";
- case 0x0F02: "Request to go online";
- case 0x0F03: "Transaction is terminated";
- case 0x0F05: "Application was not selected by kernel or ICC format error or ICC missing data error";
- case 0x0F07: "ICC didn't accept transaction";
- case 0x0F0A: "Application may fallback to magstripe technology";
- case 0x0F0C: "Transaction was cancelled";
- case 0x0F0D: "Timeout";
- case 0x0F0F: "Other EMV Error";
- case 0x0F10: "Accept the offline transaction";
- case 0x0F11: "Decline the offline transaction";
- case 0x0F21: "ICC detected tah the conditions of use are not satisfied";
- case 0x0F22: "No app were found on card matching terminal configuration";
- case 0x0F23: "Terminal file does not exist";
- case 0x0F24: "CAPK file does not exist";
- case 0x0F25: "CRL Entry does not exist";
- case 0x0FFE: "code when blocking is disabled";
- case 0x0FFF: "code when command is not applicable on the selected device";
- case 0xF005: "ICC Encrypted C-APDU Data Structure Length Error Or Format Error.";
- case 0xBBE0: "CM100 Success";
- case 0xBBE1: "CM100 Parameter Error";
- case 0xBBE2: "CM100 Low Output Buffer";

- case 0xBBE3: "CM100 Card Not Found";
- case 0xBBE4: "CM100 Collision Card Exists";
- case 0xBBE5: "CM100 Too Many Cards Exist";
- case 0xBBE6: "CM100 Saved Data Does Not Exist";
- case 0xBBE8: "CM100 No Data Available";
- case 0xBBE9: "CM100 Invalid CID Returned";
- case 0xBBEA: "CM100 Invalid Card Exists";
- case 0xBBEC: "CM100 Command Unsupported";
- case 0xBBED: "CM100 Error In Command Process";
- case 0xBBEE: "CM100 Invalid Command";
- case 0X9031: "Unknown command";
- case 0X9032: "Wrong parameter (such as the length of the command is incorrect)";
- case 0X9038: "Wait (the command couldn't be finished in BWT)";
- case 0X9039: "Busy (a previously command has not been finished)";
- case 0X903A: "Number of retries over limit";
- case 0X9040: "Invalid Manufacturing system data";
- case 0X9041: "Not authenticated";
- case 0X9042: "Invalid Master DUKPT Key";
- case 0X9043: "Invalid MAC Key";
- case 0X9044: "Reserved for future use";
- case 0X9045: "Reserved for future use";
- case 0X9046: "Invalid DATA DUKPT Key";
- case 0X9047: "Invalid PIN Pairing DUKPT Key";
- case 0X9048: "Invalid DATA Pairing DUKPT Key";
- case 0X9049: "No nonce generated";
- case 0X9949: "No GUID available. Perform getVersion first.";
- case 0X9950: "MAC Calculation unsuccessful. Check BDK value.";
- case 0X904A: "Not ready";
- case 0X904B: "Not MAC data";
- case 0X9050: "Invalid Certificate";
- case 0X9051: "Duplicate key detected";
- case 0X9052: "AT checks failed";
- case 0X9053: "TR34 checks failed";
- case 0X9054: "TR31 checks failed";
- case 0X9055: "MAC checks failed";
- case 0X9056: "Firmware download failed";

- case 0X9060: "Log is full";
- case 0X9061: "Removal sensor unengaged";
- case 0X9062: "Any hardware problems";
- case 0X9070: "ICC communication timeout";
- case 0X9071: "ICC data error (such check sum error)";
- case 0X9072: "Smart Card not powered up";

21.12.2.9 `RETURN_CODE IDTechSDK.IDT_SecureKey.device_getRKIStatus (bool isProd, bool isMultiKey, ref string status, ref Dictionary< string, RKI_KEY_TYPE > keys, string ident = " ")`

Get RKI Status From Specified Server Polls the RKI server to validate device RKI eligibility

Parameters

<i>isProd</i>	TRUE = poll production RKI server, FALSE = poll demo RKI server
<i>isMultiKey</i>	True = MultiKey, False = SingleKey
<i>status</i>	RKI status returned as a string
<i>keys</i>	Returns dictionary of available keys
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with `errorCode.getErrorString()`. When no data is available, return code = `RETURN_CODE_NO_DATA_AVAILABLE`

21.12.2.10 `RETURN_CODE IDTechSDK.IDT_SecureKey.device_readConfigurationToMemory (string memo, Configs.InstallRules rules, List< Configs.DeviceCommand > cmds, string ident = " ", bool isForeground = false)`

Read Device Configuration To Memory Executes a ViVOconfig read of device settings and stores the results in memory

Once a ViVOconfig read starts, all messages will be returned to the callback as `DeviceState.ViVOconfig`. The ViV↔Oconfig operation is complete when the return code from a `DeviceState.ViVOconfig` message is one of the following values:

- `RETURN_CODE_SDK_VIVOCONFIG_READ_CONFIG_SUCCESS` = Read process completed successfully. Execute `device_getConfigurationFromMemory` to retrieve the configuration data
- `RETURN_CODE_SDK_VIVOCONFIG_READ_CONFIG_FAILED` = Read process FAILED.

Parameters

<i>memo</i>	Optional: populates the memo field of the captured .json file
<i>rules</i>	Optional: populates the .json file with available install rules
<i>cmds</i>	Optional: populates the .json file with commands to execute
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.
<i>isForeground</i>	If TRUE, will block program until update complete. Otherwise, FALSE performs update on background.

Returns

RETURN_CODE: Values can be parsed with `errorCode.getErrorString()`. When no data is available, return code = RETURN_CODE_NO_DATA_AVAILABLE

21.12.2.11 RETURN_CODE IDTechSDK.IDT_SecureKey.device_RemoteKeyInjection (RKI_KEY_TYPE type, string keyName, string ident = " ", bool performOnForeground = false)

Remote Key Injection Performs a remote key injection for the device

Parameters

<i>type</i>	Remote Key Injection Type
<i>keyName</i>	Name of key (optional)
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.
<i>performOnForeground</i>	TRUE = block during process (default is false)

Returns

RETURN_CODE: Values can be parsed with `errorCode.getErrorString()`. When no data is available, return code = RETURN_CODE_NO_DATA_AVAILABLE

21.12.2.12 RETURN_CODE IDTechSDK.IDT_SecureKey.device_sendConfiguration (string filename, VC_OPERATION_TYPE type, bool matchFW, string ident = " ", bool isForeground = false)

Send Configuration Executes a ViVOconfig update or verify to a device

Once a ViVOconfig write or verify starts, all messages will be returned to the callback as DeviceState.ViVOconfig. The ViVOconfig operation is complete when the return code from a DeviceState.ViVOconfig message is one of the following values:

- RETURN_CODE_SDK_VIVOCONFIG_VERIFY_SUCCESS = Verification process completed successfully. No differences found.
- RETURN_CODE_SDK_VIVOCONFIG_VERIFY_SUCCESS_WITH_WARNING = Verification process completed with warnings.
- RETURN_CODE_SDK_VIVOCONFIG_VERIFY_FAILED = Verification process FAILED
- RETURN_CODE_SDK_VIVOCONFIG_WRITE_SUCCESS = Write process completed successfully.
- RETURN_CODE_SDK_VIVOCONFIG_WRITE_SUCCESS_WITH_WARNING = Write process completed with warnings
- RETURN_CODE_SDK_VIVOCONFIG_WRITE_FAILED = Write process FAILED

Parameters

<i>filename</i>	The .json configuration file to update the device with NOTE: You can also use this parameter to pass the path of where the .json file is located at on local storage instead
<i>filename</i>	The name of the .json configuration file
<i>type</i>	VC_OPERATION_TYPE

- VC_OPERATION_TYPE_WRITE = Write File To Device, Hash must validate
- VC_OPERATION_TYPE_VERIFY = Verify Device With File

- VC_OPERATION_TYPE_WRITE_IGNORE_HASH = Write File To Device, Ignore Hash
- VC_OPERATION_TYPE_WRITE_FIX_HASH = Write File To Device, Fix Hash If Necessary

Parameters

<i>matchFW</i>	TRUE = Device FW must match file FW, FALSE = Don't validate device FW against file
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.
<i>isForeground</i>	If TRUE, will block program until update complete. Otherwise, FALSE performs update on background.

Returns

RETURN_CODE: Values can be parsed with `errorCode.getErrorString()`. When no data is available, return code = RETURN_CODE_NO_DATA_AVAILABLE

21.12.2.13 RETURN_CODE IDTechSDK.IDT_SecureKey.device_sendConfigurationFromZip (byte[] zip, string filename, VC_OPERATION_TYPE type, bool matchFW, string ident = " ", bool isForeground = false)

Send Configuration From Zip Executes a ViVOconfig update or verify to a device using files from a .zip archive

Once a ViVOconfig write or update starts, all messages will be returned to the callback as DeviceState.ViVOconfig. The ViVOconfig operation is complete when the return code from a DeviceState.ViVOconfig message is one of the following values:

- RETURN_CODE_SDK_VIVOCONFIG_VERIFY_SUCCESS = Verification process completed successfully. No differences found.
- RETURN_CODE_SDK_VIVOCONFIG_VERIFY_SUCCESS_WITH_WARNING = Verification process completed with warnings.
- RETURN_CODE_SDK_VIVOCONFIG_VERIFY_FAILED = Verification process FAILED
- RETURN_CODE_SDK_VIVOCONFIG_WRITE_SUCCESS = Write process completed successfully.
- RETURN_CODE_SDK_VIVOCONFIG_WRITE_SUCCESS_WITH_WARNING = Write process completed with warnings
- RETURN_CODE_SDK_VIVOCONFIG_WRITE_FAILED = Write process FAILED

Parameters

<i>zip</i>	.Zip archive containing the .json configuration file and also any associated image assets NOTE: You can also use this parameter to pass the path of where the .zip file is located at on local storage instead
<i>filename</i>	The name of the .json configuration file
<i>type</i>	VC_OPERATION_TYPE

- VC_OPERATION_TYPE_WRITE = Write File To Device, Hash must validate
- VC_OPERATION_TYPE_VERIFY = Verify Device With File
- VC_OPERATION_TYPE_WRITE_IGNORE_HASH = Write File To Device, Ignore Hash
- VC_OPERATION_TYPE_WRITE_FIX_HASH = Write File To Device, Fix Hash If Necessary

Parameters

<i>matchFW</i>	TRUE = Device FW must match file FW, FALSE = Don't validate device FW against file
----------------	--

Parameters

<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.
<i>isForeground</i>	If TRUE, will block program until update complete. Otherwise, FALSE performs update on background.

Returns

RETURN_CODE: Values can be parsed with `errorCode.getErrorString()`. When no data is available, return code = RETURN_CODE_NO_DATA_AVAILABLE

21.12.2.14 RETURN_CODE IDTechSDK.IDT_SecureKey.device_sendDataCommand (string *cmd*, bool *calcLRC*, ref byte[] *response*, string *ident* = " ")

Send a data command to the device

Sends a command to the device.

Parameters

<i>cmd</i>	String representation of command to execute
<i>calcLRC</i>	If TRUE, this will wrap command with start/length/lrc/sum/end: '{STX}{Len_Low}{Len_High} data {CheckLRC} {CheckSUM} {ETX}'
<i>response</i>	Response data
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with `device_getResponseCodeString`

21.12.2.15 RETURN_CODE IDTechSDK.IDT_SecureKey.device_sendDataCommand_ext (string *cmd*, bool *calcLRC*, ref byte[] *response*, int *timeout*, bool *noResponse*, string *ident* = " ")

Send a data command to the device - extended

Sends a command to the device.

Parameters

<i>cmd</i>	String representation of command to execute
<i>calcLRC</i>	If TRUE, this will wrap command as NGA with start/length/lrc/sum/end: '{STX}{Len_Low}{Len_High} data {CheckLRC} {CheckSUM} {ETX}'
<i>response</i>	Response data
<i>timeout</i>	Timeout value waiting for response, in milliseconds (1000 = 1 second, string <i>ident</i> = "")
<i>noResponse</i>	if TRUE, this will not wait for a response and immediately return SUCCESS
<i>calcITP</i>	If TRUE, this will wrap command as ITP with start/end/lrc: '{STX} data {ETX}{CheckLRC}'
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with `device_getResponseCodeString`

21.12.2.16 **RETURN_CODE** IDTechSDK.IDT_SecureKey.device_sendPAE (*string command*, *ref string response*, *int timeout*, *string ident = " "*)

Send Payment Application Engine Command

Executes a PAE command

Parameters

<i>command</i>	ASCII command string, should start with "*PAE"
<i>response</i>	command response
<i>timeout</i>	timeout waiting for PAE response
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with `errorCode.getErrorString(string ident = "")`

21.12.2.17 **RETURN_CODE** IDTechSDK.IDT_SecureKey.device_sendVivoCommandP2 (*byte command*, *byte subCommand*, *byte[] data*, *ref byte[] response*, *string ident = " "*)

Send Vivo Command Protocol 2

Sends a protocol 2 command to Vivo readers (IDG/NEO, string ident = "")

Parameters

<i>command</i>	Command
<i>subCommand</i>	Sub-Command
<i>data</i>	Data. May be null
<i>response</i>	Response
<i>ip</i>	Optional IP
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with `errorCode.getErrorString(string ident = "")`

21.12.2.18 **RETURN_CODE** IDTechSDK.IDT_SecureKey.device_sendVivoCommandP2_ext (*byte command*, *byte subCommand*, *byte[] data*, *ref byte[] response*, *int timeout*, *bool noResponse*, *string ident = " "*)

Send Vivo Command Protocol 2 Extended

Sends a protocol 2 command to Vivo readers (IDG/NEO, string ident = "")

Parameters

<i>command</i>	Command
<i>subCommand</i>	Sub-Command
<i>data</i>	Data. May be null
<i>response</i>	Response
<i>timeout</i>	Timeout, in milliseconds (3000 = 3 seconds, string ident = "")
<i>noResponse</i>	TRUE = don't wait for response, FALSE = wait for response defined by timeout

Parameters

<i>ip</i>	Optional IP
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with `errorCode.getErrorString(string ident = "")`

21.12.2.19 RETURN_CODE IDTechSDK.IDT_SecureKey.device_sendVivoCommandP3 (byte *command*, byte *subCommand*, byte[] *data*, ref byte[] *response*, string *ident* = " ")

Send Vivo Command Protocol 3

Sends a protocol 3 command to Vivo readers (IDG/NEO, string *ident* = "")

Parameters

<i>command</i>	Command
<i>subCommand</i>	Sub-Command
<i>data</i>	Data. May be null
<i>response</i>	Response
<i>ip</i>	Optional IP
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with `errorCode.getErrorString(string ident = "")`

21.12.2.20 RETURN_CODE IDTechSDK.IDT_SecureKey.device_sendVivoCommandP3_ext (byte *command*, byte *subCommand*, byte[] *data*, ref byte[] *response*, int *timeout*, bool *noResponse*, string *ident* = " ")

Send Vivo Command Protocol 3 Extended

Sends a protocol 3 command to Vivo readers (IDG/NEO, string *ident* = "")

Parameters

<i>command</i>	Command
<i>subCommand</i>	Sub-Command
<i>data</i>	Data. May be null
<i>response</i>	Response
<i>timeout</i>	Timeout, in milliseconds (3000 = 3 seconds, string <i>ident</i> = "")
<i>noResponse</i>	TRUE = don't wait for response, FALSE = wait for response defined by timeout
<i>ip</i>	Optional IP
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with `errorCode.getErrorString(string ident = "")`

21.12.2.21 RETURN_CODE IDTechSDK.IDT_SecureKey.device_setOutputType (bool *isSwipe*, int *value*, string *ident* = " ")

Set Captured Data Output Type

Parameters

<i>isSwipe</i>	TRUE = settings for Swipe, FALSE = settings for MSR (only applicable for Original/Enhanced formats, string <i>ident</i> = "") NOTE: IF XML is chosen, this parameter is ignored
<i>value</i>	<ul style="list-style-type: none"> • 0 : Original Format • 1 : Enhanced Format • 2 : XML Format
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString

21.12.2.22 RETURN_CODE IDTechSDK.IDT_SecureKey.device_StartRKI (int *type*, string *ident* = " ")

Start Remote Key Injection

Starts a remote key injection request with IDTech RKI servers. Set/Get RKI url with IDT_Device.RKI_URL.

Parameters

<i>type</i>	0 = Symmetric RKI Demo Unit 1 = Symmetric RKI Production Unit 2 = PKI-RKI Demo Unit 3 = PKI-RKI Production Unit
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString

21.12.2.23 RETURN_CODE IDTechSDK.IDT_SecureKey.device_updateFirmwareFromZip (byte[] *zipfile*, string *ident* = " ", bool *isForeground* = false)

Update Firmware From Zip

Performs one or more device firmware updates from firmware files passed as a compressed archive with a valid configuration file (.json format)

Parameters

<i>zipfile</i>	Zip file containing on or more firmware update files and a .json configuration file
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.
<i>isForeground</i>	If TRUE, will block program until firmware update complete. Otherwise, FALSE performs FW update on background.

Returns

RETURN_CODE: Values can be parsed with `errorCode.getErrorString(string ident = "")`

21.12.2.24 `static int IDTechSDK.IDT_SecureKey.getCommandTimeout (string ident = " ") [static]`

Get Command Timeout

Gets the default timeout (in milliseconds) waiting for a blocking command response

Return values

<i>time</i>	Time
-------------	------

21.12.2.25 `static void IDTechSDK.IDT_SecureKey.lcd_retrieveMessage (DisplayMessages.DISPLAY_MESSAGE_LANGUAGE lang, DisplayMessages.DISPLAY_MESSAGE_IDENTIFIER id, ref string line1, ref string line2) [static]`

Retrieve LCD Message

Returns the string value for a message ID returned for LCD messaging

Parameters

<i>lang</i>	Language.
<i>id</i>	Message ID
<i>line1</i>	Line 1 string value
<i>line2</i>	Line 2 string value
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with `device_getResponseCodeString`

21.12.2.26 `RETURN_CODE IDTechSDK.IDT_SecureKey.msr_switchUSBInterfaceMode (bool blsUSBKeyboardMode)`

Switch the USB interface mode between USB HID and USB KB mode.

For Non-SRED Augusta Only

Parameters

<i>blsUSBKeyboardMode</i>	USB interface mode <ul style="list-style-type: none"> • true: Enter into USB Keyboard mode • false: Enter into USB HID mode
---------------------------	---

Returns

RETURN_CODE: Values can be parsed with `device_getResponseCodeString`

21.12.2.27 `RETURN_CODE IDTechSDK.IDT_SecureKey.msr_switchUSBInterfaceMode (bool bIsUSBKeyboardMode, ref string ident)`

Switch the USB interface mode between USB HID and USB KB mode.

For Non-SRED Augusta Only

Parameters

<i>bIsUSBKeyboardMode</i>	USB interface mode <ul style="list-style-type: none"> • true: Enter into USB Keyboard mode • false: Enter into USB HID mode
<i>ident</i>	Device ID to send command to.

Returns

RETURN_CODE: Values can be parsed with `device_getResponseCodeString`

21.12.2.28 `static String IDTechSDK.IDT_SecureKey.SDK_Version () [static]`

SDK Version

- All Devices

Returns the current version of SDK

Returns

Framework version

21.12.2.29 `static void IDTechSDK.IDT_SecureKey.setCallback (Callback my_Callback) [static]`

Set Callback

Sets the class callback

Parameters

<i>my_Callback</i>	The callback function to receive the response message from device. defined as follows. <code>delegate void Callback(IDT_DEVICE_Types sender, DeviceState state, byte[] data, IDTTransactionData card, EMV_Callback emvCallback, RETURN_CODE transactionResultCode, ident);</code>
--------------------	--

21.12.2.30 `static void IDTechSDK.IDT_SecureKey.setCallback (IntPtr my_Callback, SynchronizationContext context) [static]`

Set Callback

Sets the class callback

Parameters

<i>my_Callback</i>	The callback function to receive the response message from device. defined as follows. public unsafe delegate void MFCCallBack(Parameters parameters, ident);
<i>context</i>	The context of the UI thread

21.12.2.31 static void IDTechSDK.IDT_SecureKey.setCommandTimeout (int *milliseconds*, string *ident* = " ") [static]

Set Command Timeout

Sets the default timeout (in milliseconds) waiting for a blocking command response

Parameters

<i>milliseconds</i>	Time
---------------------	------

21.12.3 Property Documentation

21.12.3.1 IDT_SecureKey IDTechSDK.IDT_SecureKey.SharedController [static],[get]

Singleton Instance

Establishes an singleton instance of [IDT_SecureKey](#) class.

Returns

Instance of [IDT_SecureKey](#)

The documentation for this class was generated from the following file:

- /Users/randy/Repo/universal-sdk/IDTechSDK_STD/DeviceHeaders/IDT_SecureKey.cs

21.13 IDTechSDK.IDT_SecureMag Class Reference

Public Member Functions

- RETURN_CODE [config_getModelNumber](#) (ref string response, string ident="")
- RETURN_CODE [config_getSerialNumber](#) (ref string response, string ident="")
- RETURN_CODE [device_getFirmwareVersion](#) (ref string response, string ident="")
- string [device_getResponseCodeString](#) (RETURN_CODE eCode)
- RETURN_CODE [device_sendDataCommand](#) (string cmd, bool calcLRC, ref byte[] response, string ident="")
- RETURN_CODE [device_sendDataCommand_ext](#) (string cmd, bool calcLRC, ref byte[] response, int timeout, bool noResponse, string ident="")
- RETURN_CODE [msr_captureMode](#) (bool isBufferMode, bool withNotification=false, string ident="")
- RETURN_CODE [msr_disable](#) (string ident="")
- RETURN_CODE [msr_getFunctionStatus](#) (ref bool enabled, ref bool isBufferMode, ref bool withNotification, string ident="")
- RETURN_CODE [msr_getClearPANID](#) (ref byte value, string ident="")
- RETURN_CODE [msr_getExpirationMask](#) (ref byte value, string ident="")
- RETURN_CODE [device_StartRKI](#) (string ident="")
- RETURN_CODE [msr_getSwipeEncryption](#) (ref byte encryption, string ident="")
- RETURN_CODE [msr_getSwipeForcedEncryptionOption](#) (ref byte option, string ident="")

- RETURN_CODE [msr_getSwipeMaskOption](#) (ref byte option, string ident="")
- RETURN_CODE [msr_RetrieveWhiteList](#) (ref byte[] value, string ident="")
- RETURN_CODE [msr_getSetting](#) (byte setting, ref byte value, string ident="")
- RETURN_CODE [msr_getSettings](#) (byte setting, ref byte[] value, string ident="")
- RETURN_CODE [device_sendPAE](#) (string command, ref string response, int timeout, string ident="")
- RETURN_CODE [msr_setSetting](#) (byte setting, byte value, string ident="")
- RETURN_CODE [msr_setSettings](#) (byte setting, byte[] value, string ident="")
- RETURN_CODE [msr_setClearPANID](#) (byte val, string ident="")
- RETURN_CODE [msr_setExpirationMask](#) (bool mask, string ident="")
- RETURN_CODE [msr_setSwipeEncryption](#) (byte encryption, string ident="")
- RETURN_CODE [msr_setSwipeForcedEncryptionOption](#) (bool track1, bool track2, bool track3, bool track3card0, string ident="")
- RETURN_CODE [msr_setSwipeMaskOption](#) (bool track1, bool track2, bool track3, string ident="")
- RETURN_CODE [msr_setWhiteList](#) (byte[] value, byte[] MAC, string ident="")
- RETURN_CODE [msr_startMSRSwipe](#) (int timeout, string ident="")
- RETURN_CODE [msr_cancelMSRSwipe](#) (string ident="")
- RETURN_CODE [device_StartRKI](#) (int type, string ident="")
- RETURN_CODE [device_getRKIStatus](#) (bool isProd, bool isMultiKey, ref string status, ref Dictionary< string, RKI_KEY_TYPE > keys, string ident="")
- RETURN_CODE [device_getAnyRKIStatus](#) (bool isMultiKey, ref string status, ref Dictionary< string, RKI_KEY_TYPE > keys, string ident="")
- RETURN_CODE [device_updateFirmwareFromZip](#) (byte[] zipfile, string ident="", bool isForeground=false)
- RETURN_CODE [device_sendConfigurationFromZip](#) (byte[] zip, string filename, VC_OPERATION_TYPE type, bool matchFW, string ident="", bool isForeground=false)
- RETURN_CODE [device_sendConfiguration](#) (string filename, VC_OPERATION_TYPE type, bool matchFW, string ident="", bool isForeground=false)
- RETURN_CODE [device_readConfigurationToMemory](#) (string memo, Configs.InstallRules rules, List< Configs.DeviceCommand > cmds, string ident="", bool isForeground=false)
- RETURN_CODE [device_getConfigurationFromMemory](#) (ref string json, string ident="")
- RETURN_CODE [device_RemoteKeyInjection](#) (RKI_KEY_TYPE type, string keyName, string ident="", bool performOnForeground=false)

Static Public Member Functions

- static void [initSC](#) ()
- static int [getCommandTimeout](#) (string ident="")
- static void [setCommandTimeout](#) (int milliseconds, string ident="")
- static void [setCallback](#) (CallBack my_Callback)
- static void [setCallback](#) (IntPtr my_Callback, SynchronizationContext context)
- static String [SDK_Version](#) ()
- static void [lcd_retrieveMessage](#) (DisplayMessages.DISPLAY_MESSAGE_LANGUAGE lang, DisplayMessages.DISPLAY_MESSAGE_IDENTIFIER id, ref string line1, ref string line2)

Properties

- static [IDT_SecureMag SharedController](#) [get]

21.13.1 Detailed Description

Class for SecureMag MSR reder

21.13.2 Member Function Documentation

21.13.2.1 RETURN_CODE IDTechSDK.IDT_SecureMag.config_getModelNumber (ref string *response*, string *ident* = " ")

Polls device for Model Number

Parameters

<i>response</i>	Returns Model Number
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString

21.13.2.2 RETURN_CODE IDTechSDK.IDT_SecureMag.config_getSerialNumber (ref string *response*, string *ident* = " ")

Polls device for Serial Number

Parameters

<i>response</i>	Returns Serial Number
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString

21.13.2.3 RETURN_CODE IDTechSDK.IDT_SecureMag.device_getAnyRKIStatus (bool *isMultiKey*, ref string *status*, ref Dictionary< string, RKI_KEY_TYPE > *keys*, string *ident* = " ")

Get RKI Status Polls the RKI servers to validate device RKI eligibility. Note: if device type is known in advance (production or demo device), it is more efficient to use device_getRKIStatus instead

Parameters

<i>isMultiKey</i>	True = MultiKey, False = SingleKey
<i>status</i>	RKI status returned as a string
<i>keys</i>	Returns all available keys
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with errorCode.getErrorString(). When no data is available, return code = RETURN_CODE_NO_DATA_AVAILABLE

21.13.2.4 RETURN_CODE IDTechSDK.IDT_SecureMag.device_getConfigurationFromMemory (ref string *json*, string *ident* = " ")

Read Device Configuration From Memory Retrieves the ViVOconfig configuration data from memory after a successful device_readConfigurationToMemory

Parameters

<i>json</i>	Returns the .json file data that can be saved to disk
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with `errorCode.getErrorString()`. When no data is available, return code = RETURN_CODE_NO_DATA_AVAILABLE

21.13.2.5 RETURN_CODE IDTechSDK.IDT_SecureMag.device_getFirmwareVersion (ref string *response*, string *ident* = " ")

Polls device for Firmware Version

Parameters

<i>response</i>	Response returned of Firmware Version
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with `device_getResponseCodeString`

21.13.2.6 string IDTechSDK.IDT_SecureMag.device_getResponseCodeString (RETURN_CODE *eCode*)

Get the description of response result.

Parameters

<i>eCode</i>	the response result.
--------------	----------------------

Return values

<i>the</i>	string for description of response result
------------	---

- case 0: "no error, beginning task";
- case 1: "no response from reader";
- case 2: "invalid response data";
- case 3: "time out for task or CMD";
- case 4: "wrong parameter";
- case 5: "SDK is doing MSR or ICC task";
- case 6: "SDK is doing PINPad task";
- case 7: "SDK is doing CTLS task";
- case 8: "SDK is doing EMV task";
- case 9: "SDK is doing Other task";

- case 10: "err response or data";
- case 11: "no reader attached";
- case 12: "mono audio is enabled";
- case 13: "did connection";
- case 14: "audio volume is too low";
- case 15: "task or CMD be canceled";
- case 16: "UF wrong string format";
- case 17: "UF file not found";
- case 18: "UF wrong file format";
- case 19: "Attempt to contact online host failed";
- case 20: "Attempt to perform RKI failed";
- case 0x300: "Key Type(TDES) of Session Key is not same as the related Master Key.";
- case 0x400: "Related Key was not loaded.";
- case 0x500: "Key Same.";
- case 0x501: "Key is all zero";
- case 0x502: "TR-31 format error";
- case 0x702: "PAN is Error Key.";
- case 0x705: "No Internal MSR PAN (or Internal MSR PAN is erased timeout)";
- case 0X0C01: "Incorrect Frame Tag";
- case 0X0C02: "Incorrect Frame Type";
- case 0X0C03: "Unknown Frame Type";
- case 0X0C04: "Unknown Command";
- case 0X0C05: "Unknown Sub-Command";
- case 0X0C06: "CRC Error";
- case 0X0C07: "Failed";
- case 0X0C08: "Timeout";
- case 0X0C0A: "Incorrect Parameter";
- case 0X0C0B: "Command Not Supported";
- case 0X0C0C: "Sub-Command Not Supported";
- case 0X0C0D: "Parameter Not Supported / Status Abort Command";
- case 0X0C0F: "Sub-Command Not Allowed";
- case 0X0D01: "Incorrect Header Tag";
- case 0X0D02: "Unknown Command";
- case 0X0D03: "Unknown Sub-Command";
- case 0X0D04: "CRC Error in Frame";
- case 0X0D05: "Incorrect Parameter";

- case 0X0D06: "Parameter Not Supported";
- case 0X0D07: "Mal-formatted Data";
- case 0X0D08: "Timeout";
- case 0X0D0A: "Failed / NACK";
- case 0X0D0B: "Command not Allowed";
- case 0X0D0C: "Sub-Command not Allowed";
- case 0X0D0D: "Buffer Overflow (Data Length too large for reader buffer)";
- case 0X0D0E: "User Interface Event";
- case 0X0D11: "Communication type not supported, VT-1, burst, etc.";
- case 0X0D12: "Secure interface is not functional or is in an intermediate state.";
- case 0X0D13: "Data field is not mod 8";
- case 0X0D14: "Pad 0x80 not found where expected";
- case 0X0D15: "Specified key type is invalid";
- case 0X0D1: "Could not retrieve key from the SAM(InitSecureComm)";
- case 0X0D17: "Hash code problem";
- case 0X0D18: "Could not store the key into the SAM(InstallKey)";
- case 0X0D19: "Frame is too large";
- case 0X0D1A: "Unit powered up in authentication state but POS must resend the InitSecureComm command";
- case 0X0D1B: "The EEPROM may not be initialized because SecCommInterface does not make sense";
- case 0X0D1C: "Problem encoding APDU";
- case 0X0D20: "Unsupported Index(ILM) SAM Transceiver error – problem communicating with the SAM(Key Mgr)";
- case 0X0D2: "Unexpected Sequence Counter in multiple frames for single bitmap(ILM) Length error in data returned from the SAM(Key Mgr)";
- case 0X0D22: "Improper bit map(ILM)";
- case 0X0D23: "Request Online Authorization";
- case 0X0D24: "ViVOCard3 raw data read successful";
- case 0X0D25: "Message index not available(ILM) ViVOcomm activate transaction card type(ViVOcomm)";
- case 0X0D26: "Version Information Mismatch(ILM)";
- case 0X0D27: "Not sending commands in correct index message index(ILM)";
- case 0X0D28: "Time out or next expected message not received(ILM)";
- case 0X0D29: "ILM languages not available for viewing(ILM)";
- case 0X0D2A: "Other language not supported(ILM)";
- case 0X0D41: "Unknown Error from SAM";
- case 0X0D42: "Invalid data detected by SAM";
- case 0X0D43: "Incomplete data detected by SAM";

- case 0X0D44: "Reserved";
- case 0X0D45: "Invalid key hash algorithm";
- case 0X0D46: "Invalid key encryption algorithm";
- case 0X0D47: "Invalid modulus length";
- case 0X0D48: "Invalid exponent";
- case 0X0D49: "Key already exists";
- case 0X0D4A: "No space for new RID";
- case 0X0D4B: "Key not found";
- case 0X0D4C: "Crypto not responding";
- case 0X0D4D: "Crypto communication error";
- case 0X0D4E: "Module-specific error for Key Manager";
- case 0X0D4F: "All key slots are full (maximum number of keys has been installed)";
- case 0X0D50: "Auto-Switch OK";
- case 0X0D51: "Auto-Switch failed";
- case 0X0D90: "Account DUKPT Key not exist";
- case 0X0D91: "Account DUKPT Key KSN exhausted";
- case 0x0D00: "This Key had been loaded.";
- case 0x0E00: "Base Time was loaded.";
- case 0x0F00: "Encryption Or Decryption Failed.";
- case 0x1000: "Battery Low Warning (It is High Priority Response while Battery is Low.)";
- case 0x1800: "Send "Cancel Command" after send "Get Encrypted PIN" & "Get Numeric" & "Get Amount";
- case 0x1900: "Press "Cancel" key after send "Get Encrypted PIN" & "Get Numeric" & "Get Amount";
- case 0x30FF: "Security Chip is not connect";
- case 0x3000: "Security Chip is deactivation & Device is In Removal Legally State.";
- case 0x3101: "Security Chip is activation & Device is In Removal Legally State.";
- case 0x5500: "No Admin DUKPT Key.";
- case 0x5501: "Admin DUKPT Key STOP.";
- case 0x5502: "Admin DUKPT Key KSN is Error.";
- case 0x5503: "Get Authentication Code1 Failed.";
- case 0x5504: "Validate Authentication Code Error.";
- case 0x5505: "Encrypt or Decrypt data failed.";
- case 0x5506: "Not Support the New Key Type.";
- case 0x5507: "New Key Index is Error.";
- case 0x5508: "Step Error.";
- case 0x5509: "KSN Error";
- case 0x550A: "MAC Error.";

- case 0x550B: "Key Usage Error.";
- case 0x550C: "Mode Of Use Error.";
- case 0x550F: "Other Error.";
- case 0x6000: "Save or Config Failed / Or Read Config Error.";
- case 0x6200: "No Serial Number.";
- case 0x6900: "Invalid Command - Protocol is right, but task ID is invalid.";
- case 0x6A01: "Unsupported Command – Protocol and task ID are right, but command is invalid – In this State";
- case 0x6A00: "Unsupported Command - Protocol and task ID are right, but command is invalid.";
- case 0x6B00: "Unknown parameter in command - Protocol task ID and command are right, but parameter is invalid.";
- case 0x6C00: "Unknown parameter in command – Protocol task ID and command are right, but length is out of the requirement.";
- case 0x7200: "Device is suspend (MKSK suspend or press password suspend).";
- case 0x7300: "PIN DUKPT is STOP (21 bit 1).";
- case 0x7400: "Device is Busy.";
- case 0xE100: "Can not enter sleep mode";
- case 0xE200: "File has existed";
- case 0xE300: "File has not existed";
- case 0xE313: "IO line low -- Card error after session start";
- case 0xE400: "Open File Error";
- case 0xE500: "SmartCard Error";
- case 0xE600: "Get MSR Card data is error";
- case 0xE700: "Command time out";
- case 0xE800: "File read or write is error";
- case 0xE900: "Active 1850 error!";
- case 0xEA00: "Load bootloader error";
- case 0xEF00: "Protocol Error- STX or ETX or check error.";
- case 0xEB00: "Picture is not exist";
- case 0x2C02: "No Microprocessor ICC seated";
- case 0x2C06: "no card seated to request ATR";
- case 0x2D01: "Card Not Supported,";
- case 0x2D03: "Card Not Supported, wants CRC";
- case 0x690D: "Command not supported on reader without ICC support";
- case 0x8100: "ICC error time out on power-up";
- case 0x8200: "invalid TS character received - Wrong operation step";
- case 0x8300: "Decode MSR Error";

- case 0x8400: "TriMagII no Response";
- case 0x8500: "No Swipe MSR Card";
- case 0x8510: "No Financial Card";
- case 0x8600: "Unsupported F, D, or combination of F and D";
- case 0x8700: "protocol not supported EMV TD1 out of range";
- case 0x8800: "power not at proper level";
- case 0x8900: "ATR length too long";
- case 0x8B01: "EMV invalid TA1 byte value";
- case 0x8B02: "EMV TB1 required";
- case 0x8B03: "EMV Unsupported TB1 only 00 allowed";
- case 0x8B04: "EMV Card Error, invalid BWI or CWI";
- case 0x8B06: "EMV TB2 not allowed in ATR";
- case 0x8B07: "EMV TC2 out of range";
- case 0x8B08: "EMV TC2 out of range";
- case 0x8B09: "per EMV96 TA3 must be > 0xF";
- case 0x8B10: "ICC error on power-up";
- case 0x8B11: "EMV T=1 then TB3 required";
- case 0x8B12: "Card Error, invalid BWI or CWI";
- case 0x8B13: "Card Error, invalid BWI or CWI";
- case 0x8B17: "EMV TC1/TB3 conflict-";
- case 0x8B20: "EMV TD2 out of range must be T=1";
- case 0x8C00: "TCK error";
- case 0xA304: "connector has no voltage setting";
- case 0xA305: "ICC error on power-up invalid (SBLK(IFSD) exchange";
- case 0xE301: "ICC error after session start";
- case 0xFF00: "Request to go online";
- case 0xFF01: "EMV: Accept the offline transaction";
- case 0xFF02: "EMV: Decline the offline transaction";
- case 0xFF03: "EMV: Accept the online transaction";
- case 0xFF04: "EMV: Decline the online transaction";
- case 0xFF05: "EMV: Application may fallback to magstripe technology";
- case 0xFF06: "EMV: ICC detected tah the conditions of use are not satisfied";
- case 0xFF07: "EMV: ICC didn't accept transaction";
- case 0xFF08: "EMV: Transaction was cancelled";
- case 0xFF09: "EMV: Application was not selected by kernel or ICC format error or ICC missing data error";
- case 0xFF0A: "EMV: Transaction is terminated";

- case 0xFF0B: "EMV: Other EMV Error";
- case 0xFFFF: "NO RESPONSE";
- case 0xF002: "ICC communication timeout";
- case 0xF003: "ICC communication Error";
- case 0xF00F: "ICC Card Seated and Highest Priority, disable MSR work request";
- case 0xF200: "AID List / Application Data is not exist";
- case 0xF201: "Terminal Data is not exist";
- case 0xF202: "TLV format is error";
- case 0xF203: "AID List is full";
- case 0xF204: "Any CA Key is not exist";
- case 0xF205: "CA Key RID is not exist";
- case 0xF206: "CA Key Index it not exist";
- case 0xF207: "CA Key is full";
- case 0xF208: "CA Key Hash Value is Error";
- case 0xF209: "Transaction format error";
- case 0xF20A: "The command will not be processing";
- case 0xF20B: "CRL is not exist";
- case 0xF20C: "CRL number exceed max number";
- case 0xF20D: "Amount,Other Amount,Trasaction Type are missing";
- case 0xF20E: "The Identification of algorithm is mistake";
- case 0xF20F: "No Financial Card";
- case 0xF210: "In Encrypt Result state, TLV total Length is greater than Max Length";
- case 0x1001: "INVALID ARG";
- case 0x1002: "FILE_OPEN_FAILED";
- case 0x1003: "FILE OPERATION_FAILED";
- case 0x2001: "MEMORY_NOT_ENOUGH";
- case 0x3002: "SMARTCARD_FAIL";
- case 0x3003: "SMARTCARD_INIT_FAILED";
- case 0x3004: "FALLBACK_SITUATION";
- case 0x3005: "SMARTCARD_ABSENT";
- case 0x3006: "SMARTCARD_TIMEOUT";
- case 0x5001: "EMV_PARSING_TAGS_FAILED";
- case 0x5002: "EMV_DUPLICATE_CARD_DATA_ELEMENT";
- case 0x5003: "EMV_DATA_FORMAT_INCORRECT";
- case 0x5004: "EMV_NO_TERM_APP";
- case 0x5005: "EMV_NO_MATCHING_APP";

- case 0x5006: "EMV_MISSING_MANDATORY_OBJECT";
- case 0x5007: "EMV_APP_SELECTION_RETRY";
- case 0x5008: "EMV_GET_AMOUNT_ERROR";
- case 0x5009: "EMV_CARD_REJECTED";
- case 0x5010: "EMV_AIP_NOT_RECEIVED";
- case 0x5011: "EMV_AFL_NOT_RECEIVED";
- case 0x5012: "EMV_AFL_LEN_OUT_OF_RANGE";
- case 0x5013: "EMV_SFI_OUT_OF_RANGE";
- case 0x5014: "EMV_AFL_INCORRECT";
- case 0x5015: "EMV_EXP_DATE_INCORRECT";
- case 0x5016: "EMV_EFF_DATE_INCORRECT";
- case 0x5017: "EMV_ISS_COD_TBL_OUT_OF_RANGE";
- case 0x5018: "EMV_CRYPTOGAM_TYPE_INCORRECT";
- case 0x5019: "EMV_PSE_NOT_SUPPORTED_BY_CARD";
- case 0x5020: "EMV_USER_SELECTED_LANGUAGE";
- case 0x5021: "EMV_SERVICE_NOT_ALLOWED";
- case 0x5022: "EMV_NO_TAG_FOUND";
- case 0x5023: "EMV_CARD_BLOCKED";
- case 0x5024: "EMV_LEN_INCORRECT";
- case 0x5025: "CARD_COM_ERROR";
- case 0x5026: "EMV_TSC_NOT_INCREASED";
- case 0x5027: "EMV_HASH_INCORRECT";
- case 0x5028: "EMV_NO_ARC";
- case 0x5029: "EMV_INVALID_ARC";
- case 0x5030: "EMV_NO_ONLINE_COMM";
- case 0x5031: "TRAN_TYPE_INCORRECT";
- case 0x5032: "EMV_APP_NO_SUPPORT";
- case 0x5033: "EMV_APP_NOT_SELECT";
- case 0x5034: "EMV_LANG_NOT_SELECT";
- case 0x5035: "EMV_NO_TERM_DATA";
- case 0x6001: "CVM_TYPE_UNKNOWN";
- case 0x6002: "CVM_AIP_NOT_SUPPORTED";
- case 0x6003: "CVM_TAG_8E_MISSING";
- case 0x6004: "CVM_TAG_8E_FORMAT_ERROR";
- case 0x6005: "CVM_CODE_IS_NOT_SUPPORTED";
- case 0x6006: "CVM_COND_CODE_IS_NOT_SUPPORTED";

- case 0x6007: "NO_MORE_CVM";
- case 0x6008: "PIN_BYPASSED_BEFORE";
- case 0x7001: "PK_BUFFER_SIZE_TOO_BIG";
- case 0x7002: "PK_FILE_WRITE_ERROR";
- case 0x7003: "PK_HASH_ERROR";
- case 0x8001: "NO_CARD HOLDER_CONFIRMATION";
- case 0x8002: "GET_ONLINE_PIN";
- case 0xD000: "Data not exist";
- case 0xD001: "Data access error";
- case 0xD100: "RID not exist";
- case 0xD101: "RID existed";
- case 0xD102: "Index not exist";
- case 0xD200: "Maximum exceeded";
- case 0xD201: "Hash error";
- case 0xD205: "System Busy";
- case 0x0E01: "Unable to go online";
- case 0x0E02: "Technical Issue";
- case 0x0E03: "Declined";
- case 0x0E04: "Issuer Referral transaction";
- case 0x0F01: "Decline the online transaction";
- case 0x0F02: "Request to go online";
- case 0x0F03: "Transaction is terminated";
- case 0x0F05: "Application was not selected by kernel or ICC format error or ICC missing data error";
- case 0x0F07: "ICC didn't accept transaction";
- case 0x0F0A: "Application may fallback to magstripe technology";
- case 0x0F0C: "Transaction was cancelled";
- case 0x0F0D: "Timeout";
- case 0x0F0F: "Other EMV Error";
- case 0x0F10: "Accept the offline transaction";
- case 0x0F11: "Decline the offline transaction";
- case 0x0F21: "ICC detected that the conditions of use are not satisfied";
- case 0x0F22: "No app were found on card matching terminal configuration";
- case 0x0F23: "Terminal file does not exist";
- case 0x0F24: "CAPK file does not exist";
- case 0x0F25: "CRL Entry does not exist";
- case 0x0FFE: "code when blocking is disabled";

- case 0x0FFF: "code when command is not applicable on the selected device";
- case 0xF005: "ICC Encrypted C-APDU Data Structure Length Error Or Format Error.";
- case 0xBBE0: "CM100 Success";
- case 0xBBE1: "CM100 Parameter Error";
- case 0xBBE2: "CM100 Low Output Buffer";
- case 0xBBE3: "CM100 Card Not Found";
- case 0xBBE4: "CM100 Collision Card Exists";
- case 0xBBE5: "CM100 Too Many Cards Exist";
- case 0xBBE6: "CM100 Saved Data Does Not Exist";
- case 0xBBE8: "CM100 No Data Available";
- case 0xBBE9: "CM100 Invalid CID Returned";
- case 0xBBEA: "CM100 Invalid Card Exists";
- case 0xBBEC: "CM100 Command Unsupported";
- case 0xBBED: "CM100 Error In Command Process";
- case 0xBBEE: "CM100 Invalid Command";
- case 0X9031: "Unknown command";
- case 0X9032: "Wrong parameter (such as the length of the command is incorrect)";
- case 0X9038: "Wait (the command couldn't be finished in BWT)";
- case 0X9039: "Busy (a previously command has not been finished)";
- case 0X903A: "Number of retries over limit";
- case 0X9040: "Invalid Manufacturing system data";
- case 0X9041: "Not authenticated";
- case 0X9042: "Invalid Master DUKPT Key";
- case 0X9043: "Invalid MAC Key";
- case 0X9044: "Reserved for future use";
- case 0X9045: "Reserved for future use";
- case 0X9046: "Invalid DATA DUKPT Key";
- case 0X9047: "Invalid PIN Pairing DUKPT Key";
- case 0X9048: "Invalid DATA Pairing DUKPT Key";
- case 0X9049: "No nonce generated";
- case 0X9949: "No GUID available. Perform getVersion first.";
- case 0X9950: "MAC Calculation unsuccessful. Check BDK value.";
- case 0X904A: "Not ready";
- case 0X904B: "Not MAC data";
- case 0X9050: "Invalid Certificate";
- case 0X9051: "Duplicate key detected";

- case 0X9052: "AT checks failed";
- case 0X9053: "TR34 checks failed";
- case 0X9054: "TR31 checks failed";
- case 0X9055: "MAC checks failed";
- case 0X9056: "Firmware download failed";
- case 0X9060: "Log is full";
- case 0X9061: "Removal sensor unengaged";
- case 0X9062: "Any hardware problems";
- case 0X9070: "ICC communication timeout";
- case 0X9071: "ICC data error (such check sum error)";
- case 0X9072: "Smart Card not powered up";

21.13.2.7 `RETURN_CODE IDTechSDK.IDT_SecureMag.device_getRKIStatus (bool isProd, bool isMultiKey, ref string status, ref Dictionary< string, RKI_KEY_TYPE > keys, string ident = " ")`

Get RKI Status From Specified Server Polls the RKI server to validate device RKI eligibility

Parameters

<i>isProd</i>	TRUE = poll production RKI server, FALSE = poll demo RKI server
<i>isMultiKey</i>	True = MultiKey, False = SingleKey
<i>status</i>	RKI status returned as a string
<i>keys</i>	Returns dictionary of available keys
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with `errorCode.getErrorString()`. When no data is available, return code = **RETURN_CODE_NO_DATA_AVAILABLE**

21.13.2.8 `RETURN_CODE IDTechSDK.IDT_SecureMag.device_readConfigurationToMemory (string memo, Configs.InstallRules rules, List< Configs.DeviceCommand > cmds, string ident = " ", bool isForeground = false)`

Read Device Configuration To Memory Executes a ViVOconfig read of device settings and stores the results in memory

Once a ViVOconfig read starts, all messages will be returned to the callback as `DeviceState.ViVOconfig`. The ViVOconfig operation is complete when the return code from a `DeviceState.ViVOconfig` message is one of the following values:

- **RETURN_CODE_SDK_VIVOCONFIG_READ_CONFIG_SUCCESS** = Read process completed successfully. Execute `device_getConfigurationFromMemory` to retrieve the configuration data
- **RETURN_CODE_SDK_VIVOCONFIG_READ_CONFIG_FAILED** = Read process FAILED.

Parameters

<i>memo</i>	Optional: populates the memo field of the captured .json file
-------------	---

Parameters

<i>rules</i>	Optional: populates the .json file with available install rules
<i>cmds</i>	Optional: populates the .json file with commands to execute
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.
<i>isForeground</i>	If TRUE, will block program until update complete. Otherwise, FALSE performs update on background.

Returns

RETURN_CODE: Values can be parsed with `errorCode.getErrorString()`. When no data is available, return code = RETURN_CODE_NO_DATA_AVAILABLE

21.13.2.9 RETURN_CODE IDTechSDK.IDT_SecureMag.device_RemoteKeyInjection (RKL_KEY_TYPE type, string keyName, string ident = " ", bool performOnForeground = false)

Remote Key Injection Performs a remote key injection for the device

Parameters

<i>type</i>	Remote Key Injection Type
<i>keyName</i>	Name of key (optional)
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.
<i>performOnForeground</i>	TRUE = block during process (default is false)

Returns

RETURN_CODE: Values can be parsed with `errorCode.getErrorString()`. When no data is available, return code = RETURN_CODE_NO_DATA_AVAILABLE

21.13.2.10 RETURN_CODE IDTechSDK.IDT_SecureMag.device_sendConfiguration (string filename, VC_OPERATION_TYPE type, bool matchFW, string ident = " ", bool isForeground = false)

Send Configuration Executes a ViVOconfig update or verify to a device

Once a ViVOconfig write or verify starts, all messages will be returned to the callback as DeviceState.ViVOconfig. The ViVOconfig operation is complete when the return code from a DeviceState.ViVOconfig message is one of the following values:

- RETURN_CODE_SDK_VIVOCONFIG_VERIFY_SUCCESS = Verification process completed successfully. No differences found.
- RETURN_CODE_SDK_VIVOCONFIG_VERIFY_SUCCESS_WITH_WARNING = Verification process completed with warnings.
- RETURN_CODE_SDK_VIVOCONFIG_VERIFY_FAILED = Verification process FAILED
- RETURN_CODE_SDK_VIVOCONFIG_WRITE_SUCCESS = Write process completed successfully.
- RETURN_CODE_SDK_VIVOCONFIG_WRITE_SUCCESS_WITH_WARNING = Write process completed with warnings
- RETURN_CODE_SDK_VIVOCONFIG_WRITE_FAILED = Write process FAILED

Parameters

<i>filename</i>	The .json configuration file to update the device with NOTE: You can also use this parameter to pass the path of where the .json file is located at on local storage instead
<i>filename</i>	The name of the .json configuration file
<i>type</i>	VC_OPERATION_TYPE

- VC_OPERATION_TYPE_WRITE = Write File To Device, Hash must validate
- VC_OPERATION_TYPE_VERIFY = Verify Device With File
- VC_OPERATION_TYPE_WRITE_IGNORE_HASH = Write File To Device, Ignore Hash
- VC_OPERATION_TYPE_WRITE_FIX_HASH = Write File To Device, Fix Hash If Necessary

Parameters

<i>matchFW</i>	TRUE = Device FW must match file FW, FALSE = Don't validate device FW against file
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.
<i>isForeground</i>	If TRUE, will block program until update complete. Otherwise, FALSE performs update on background.

Returns

RETURN_CODE: Values can be parsed with `errorCode.getErrorString()`. When no data is available, return code = RETURN_CODE_NO_DATA_AVAILABLE

21.13.2.11 RETURN_CODE IDTechSDK.IDT_SecureMag.device_sendConfigurationFromZip (byte[] zip, string filename, VC_OPERATION_TYPE type, bool matchFW, string ident = "", bool isForeground = false)

Send Configuration From Zip Executes a ViVOconfig update or verify to a device using files from a .zip archive

Once a ViVOconfig write or update starts, all messages will be returned to the callback as DeviceState.ViVOconfig. The ViVOconfig operation is complete when the return code from a DeviceState.ViVOconfig message is one of the following values:

- RETURN_CODE_SDK_VIVOCONFIG_VERIFY_SUCCESS = Verification process completed successfully. No differences found.
- RETURN_CODE_SDK_VIVOCONFIG_VERIFY_SUCCESS_WITH_WARNING = Verification process completed with warnings.
- RETURN_CODE_SDK_VIVOCONFIG_VERIFY_FAILED = Verification process FAILED
- RETURN_CODE_SDK_VIVOCONFIG_WRITE_SUCCESS = Write process completed successfully.
- RETURN_CODE_SDK_VIVOCONFIG_WRITE_SUCCESS_WITH_WARNING = Write process completed with warnings
- RETURN_CODE_SDK_VIVOCONFIG_WRITE_FAILED = Write process FAILED

Parameters

<i>zip</i>	.Zip archive containing the .json configuration file and also any associated image assets NOTE: You can also use this parameter to pass the path of where the .zip file is located at on local storage instead
<i>filename</i>	The name of the .json configuration file
<i>type</i>	VC_OPERATION_TYPE

- VC_OPERATION_TYPE_WRITE = Write File To Device, Hash must validate
- VC_OPERATION_TYPE_VERIFY = Verify Device With File
- VC_OPERATION_TYPE_WRITE_IGNORE_HASH = Write File To Device, Ignore Hash
- VC_OPERATION_TYPE_WRITE_FIX_HASH = Write File To Device, Fix Hash If Necessary

Parameters

<i>matchFW</i>	TRUE = Device FW must match file FW, FALSE = Don't validate device FW against file
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.
<i>isForeground</i>	If TRUE, will block program until update complete. Otherwise, FALSE performs update on background.

Returns

RETURN_CODE: Values can be parsed with `errorCode.getErrorString()`. When no data is available, return code = RETURN_CODE_NO_DATA_AVAILABLE

21.13.2.12 RETURN_CODE IDTechSDK.IDT_SecureMag.device_sendDataCommand (string *cmd*, bool *calcLRC*, ref byte[] *response*, string *ident* = " ")

Send a data command to the device

Sends a command to the device.

Parameters

<i>cmd</i>	String representation of command to execute
<i>calcLRC</i>	If TRUE, this will wrap command with start/length/lrc/sum/end: '{STX}{Len_Low}{Len_High} data {CheckLRC} {CheckSUM} {ETX}'
<i>response</i>	Response data
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with `device_getResponseCodeString`

21.13.2.13 RETURN_CODE IDTechSDK.IDT_SecureMag.device_sendDataCommand_ext (string *cmd*, bool *calcLRC*, ref byte[] *response*, int *timeout*, bool *noResponse*, string *ident* = " ")

Send a data command to the device - extended

Sends a command to the device.

Parameters

<i>cmd</i>	String representation of command to execute
<i>calcLRC</i>	If TRUE, this will wrap command as NGA with start/length/lrc/sum/end: '{STX}{Len_Low}{Len_High} data {CheckLRC} {CheckSUM} {ETX}'
<i>response</i>	Response data
<i>timeout</i>	Timeout value waiting for response, in milliseconds (1000 = 1 second, string <i>ident</i> = "")
<i>noResponse</i>	if TRUE, this will not wait for a response and immediately return SUCCESS
<i>calcITP</i>	If TRUE, this will wrap command as ITP with start/end/lrc: '{STX} data {ETX}{CheckLRC}'
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString

21.13.2.14 RETURN_CODE IDTechSDK.IDT_SecureMag.device_sendPAE (string *command*, ref string *response*, int *timeout*, string *ident* = " ")

Set Single MSR Setting value

Parameters

<i>setting</i>	MSR Setting to set
<i>value</i>	MSR Setting value
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString Send Payment Application Engine Command

Executes a PAE command

Parameters

<i>command</i>	ASCII command string, should start with "*PAE"
<i>response</i>	command response
<i>timeout</i>	timeout waiting for PAE response
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with errorCode.getErrorString(string ident = "")

21.13.2.15 RETURN_CODE IDTechSDK.IDT_SecureMag.device_StartRKI (string *ident* = " ")

Get Swipe Data Encryption

For Non-SRED Augusta Only

Returns the encryption used for swipe data

Parameters

<i>encryption</i>	1 = TDES, 2 = AES, 0 = NONE
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString Start Remote Key Injection

Starts a remote key injection request with IDTech RKI servers.

Parameters

<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.
--------------	--

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString

21.13.2.16 RETURN_CODE IDTechSDK.IDT_SecureMag.device_StartRKI (int *type*, string *ident* = " ")

Start Remote Key Injection

Starts a remote key injection request with IDTech RKI servers. Set/Get RKI url with IDT_Device.RKI_URL.

Parameters

<i>type</i>	0 = Symmetric RKI Demo Unit 1 = Symmetric RKI Production Unit 2 = PKI-RKI Demo Unit 3 = PKI-RKI Production Unit
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString

21.13.2.17 RETURN_CODE IDTechSDK.IDT_SecureMag.device_updateFirmwareFromZip (byte[] *zipfile*, string *ident* = " ", bool *isForeground* = false)

Update Firmware From Zip

Performs one or more device firmware updates from firmware files passed as a compressed archive with a valid configuration file (.json format)

Parameters

<i>zipfile</i>	Zip file containing on or more firmware update files and a .json configuration file
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.
<i>isForeground</i>	If TRUE, will block program until firmware update complete. Otherwise, FALSE performs FW update on background.

Returns

RETURN_CODE: Values can be parsed with errorCode.getErrorString(string ident = "")

21.13.2.18 static int IDTechSDK.IDT_SecureMag.getCommandTimeout (string *ident* = " ") [static]

Get Command Timeout

Gets the default timeout (in milliseconds) waiting for a blocking command response

Return values

<i>time</i>	Time
-------------	------

21.13.2.19 static void IDTechSDK.IDT_SecureMag.lcd_retrieveMessage (DisplayMessages.DISPLAY_MESSAGE_LANGUAGE *lang*, DisplayMessages.DISPLAY_MESSAGE_IDENTIFIER *id*, ref string *line1*, ref string *line2*) [static]

Retrieve LCD Message

Returns the string value for a message ID returned for LCD messaging

Parameters

<i>lang</i>	Language.
<i>id</i>	Message ID
<i>line1</i>	Line 1 string value
<i>line2</i>	Line 2 string value
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString

21.13.2.20 RETURN_CODE IDTechSDK.IDT_SecureMag.msr_cancelMSRSwipe (string *ident* = " ")

Disable MSR Swipe Cancels MSR swipe request.

Parameters

<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.
--------------	--

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString

21.13.2.21 RETURN_CODE IDTechSDK.IDT_SecureMag.msr_captureMode (bool *isBufferMode*, bool *withNotification* = false, string *ident* = " ")

Set MSR Capture Mode.

For Non-SRED Augusta Only

Switch between Auto mode and Buffer mode. Auto mode only available on KB interface

Parameters

<i>isBufferMode</i>	<ul style="list-style-type: none"> • true: Enter into Buffer mode. • false: Enter into Auto mode. KB mode only. Swipes automatically captured and sent to keyboard buffer
<i>withNotification</i>	<ul style="list-style-type: none"> • true: with notification when swiped MSR data. • false: without notification when swiped MSR data.
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString

21.13.2.22 RETURN_CODE IDTechSDK.IDT_SecureMag.msr_disable (string *ident* = " ")

Disable MSR function.

Parameters

<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.
--------------	--

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString

21.13.2.23 RETURN_CODE IDTechSDK.IDT_SecureMag.msr_getClearPANID (ref byte *value*, string *ident* = " ")

Get Clear PAN Digits

Returns the number of digits that begin the PAN that will be in the clear

Parameters

<i>value</i>	Number of digits in clear. Values are char '0' - '6':
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString

21.13.2.24 RETURN_CODE IDTechSDK.IDT_SecureMag.msr_getExpirationMask (ref byte *value*, string *ident* = " ")

Get Expiration Masking

Get the flag that determines if to mask the expiration date

Parameters

<i>value</i>	'0' = masked, '1' = not-masked
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString

21.13.2.25 RETURN_CODE IDTechSDK.IDT_SecureMag.msr_getFunctionStatus (ref bool *enabled*, ref bool *isBufferMode*, ref bool *withNotification*, string *ident* = " ")

Get MSR Function status Get MSR Function status about enable/disable and with or without seated notification

Parameters

<i>enabled</i>	<ul style="list-style-type: none"> • true: MSR Function enabled. • false: MSR Function disabled.
<i>isBufferMode</i>	<ul style="list-style-type: none"> • true: in the buffer mode. • false: in the auto mode.
<i>withNotification</i>	<ul style="list-style-type: none"> • true: with notification when swiped MSR Card. • false: without notification when swiped MSR Card.
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString

21.13.2.26 RETURN_CODE IDTechSDK.IDT_SecureMag.msr_getSetting (byte *setting*, ref byte *value*, string *ident* = " ")

Get Single MSR Setting value

Returns the encryption used for sweep data

Parameters

<i>setting</i>	MSR Setting to retrieve
<i>value</i>	MSR Setting value
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString

21.13.2.27 RETURN_CODE IDTechSDK.IDT_SecureMag.msr_getSettings (byte *setting*, ref byte[] *value*, string *ident* = " ")

Get Multi MSR Setting value

Returns the encryption used for sweep data

Parameters

<i>setting</i>	MSR Setting to retrieve
<i>value</i>	MSR Setting value
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString

21.13.2.28 RETURN_CODE IDTechSDK.IDT_SecureMag.msr_getSwipeForcedEncryptionOption (ref byte *option*, string *ident* = " ")

Get Swipe Data Encryption

Gets the swipe force encryption options

Parameters

<i>option</i>	Byte using lower four bits as flags. 0 = Force Encryption Off, 1 = Force Encryption On bit0 = Track 1 bit1 = Track 2 bit2 = Track 3 bit4 = Track 3 Card Option 0
---------------	--

Example: Response 0x03 = Track1/Track2 Forced Encryption, Track3/Track3-0 no Forced Encryption

Parameters

<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.
--------------	--

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString

21.13.2.29 RETURN_CODE IDTechSDK.IDT_SecureMag.msr_getSwipeMaskOption (ref byte *option*, string *ident* = " ")

Get Swipe Mask Option

Gets the swipe mask/clear data sending option

Parameters

<i>option</i>	Byte using lower three bits as flags. 0 = Mask Option Off, 1 = Mask Option On bit0 = Track 1 bit1 = Track 2 bit2 = Track 3
---------------	--

Example: Response 0x03 = Track1/Track2 Masked Option ON, Track3 Masked Option Off

Parameters

<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.
--------------	--

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString

21.13.2.30 RETURN_CODE IDTechSDK.IDT_SecureMag.msr_RetrieveWhiteList (ref byte[] *value*, string *ident* = " ")

Retrieve MSR White List

For Non-SRED Augusta Only

Parameters

<i>value</i>	is the white list data which is ASN.1 Block format
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString

21.13.2.31 RETURN_CODE IDTechSDK.IDT_SecureMag.msr_setClearPANID (byte *val*, string *ident* = " ")

Set Clear PAN Digits

Sets the amount of digits shown in the clear (not masked) at the beginning of the returned PAN value

Parameters

<i>val</i>	Number of digits to show in clear. Range 0-6.
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString

21.13.2.32 RETURN_CODE IDTechSDK.IDT_SecureMag.msr_setExpirationMask (bool *mask*, string *ident* = " ")

Set Expiration Masking

Sets the flag to mask the expiration date

Parameters

<i>mask</i>	TRUE = mask expiration
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString

21.13.2.33 RETURN_CODE IDTechSDK.IDT_SecureMag.msr_setSettings (byte *setting*, byte[] *value*, string *ident* = " ")

Set Multi MSR Setting value

Parameters

<i>setting</i>	MSR Setting to set
<i>value</i>	MSR Setting value
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString

21.13.2.34 RETURN_CODE IDTechSDK.IDT_SecureMag.msr_setSwipeEncryption (byte *encryption*, string *ident* = " ")

Set Swipe Data Encryption

For Non-SRED Augusta Only

Sets the swipe encryption method

Parameters

<i>encryption</i>	1 = TDES, 2 = AES
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString

21.13.2.35 RETURN_CODE IDTechSDK.IDT_SecureMag.msr_setSwipeForcedEncryptionOption (bool *track1*, bool *track2*, bool *track3*, bool *track3card0*, string *ident* = " ")

Set Swipe Force Encryption

Sets the swipe force encryption options

Parameters

<i>track1</i>	Force encrypt track 1
<i>track2</i>	Force encrypt track 2
<i>track3</i>	Force encrypt track 3
<i>track3card0</i>	Force encrypt track 3 when card type is 0
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString

21.13.2.36 RETURN_CODE IDTechSDK.IDT_SecureMag.msr_setSwipeMaskOption (bool *track1*, bool *track2*, bool *track3*, string *ident* = " ")

Set Swipe Mask Option

Sets the swipe mask/clear data sending option

Parameters

<i>track1</i>	Mask track 1 allowed
<i>track2</i>	Mask track 2 allowed
<i>track3</i>	Mask track 3 allowed
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString

21.13.2.37 RETURN_CODE IDTechSDK.IDT_SecureMag.msr_setWhiteList (byte[] *value*, byte[] *MAC*, string *ident* = " ")

Set MSR White List

For Non-SRED Augusta Only

Parameters

<i>value</i>	the white list data which is ASN.1 Block format
<i>MAC</i>	Message Authenticate Code. For non-PCI, use null
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString

21.13.2.38 RETURN_CODE IDTechSDK.IDT_SecureMag.msr_startMSRSwipe (int *timeout*, string *ident* = " ")

Enable MSR Swipe

Enables MSR, waiting for swipe to occur. Allows track selection. Returns IDTMSRData instance to deviceDelegate↵
::swipeMSRData:(string ident = "")

Parameters

<i>timeout</i>	Swipe Timeout Value
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString

21.13.2.39 static String IDTechSDK.IDT_SecureMag.SDK_Version () [static]

SDK Version

- All Devices

Returns the current version of SDK

Returns

Framework version

21.13.2.40 static void IDTechSDK.IDT_SecureMag.setCallback (Callback *my_Callback*) [static]

Set Callback

Sets the class callback

Parameters

<i>my_Callback</i>	The callback function to receive the response message from device. defined as follows. delegate void CallBack(IDT_DEVICE_Types sender, DeviceState state, byte[] data, IDTTransactionData card, EMV_Callback emvCallback, RETURN_CODE transactionResultCode, ident);
--------------------	---

21.13.2.41 static void IDTechSDK.IDT_SecureMag.setCallback (IntPtr *my_Callback*, SynchronizationContext *context*)
[static]

Set Callback

Sets the class callback

Parameters

<i>my_Callback</i>	The callback function to receive the response message from device. defined as follows. public unsafe delegate void MFCCallBack(Parameters parameters, ident);
<i>context</i>	The context of the UI thread

21.13.2.42 static void IDTechSDK.IDT_SecureMag.setCommandTimeout (int *milliseconds*, string *ident* = " ") [static]

Set Command Timeout

Sets the default timeout (in milliseconds) waiting for a blocking command response

Parameters

<i>milliseconds</i>	Time
---------------------	------

21.13.3 Property Documentation

21.13.3.1 IDT_SecureMag IDTechSDK.IDT_SecureMag.SharedController [static],[get]

Singleton Instance

Establishes an singleton instance of [IDT_SecureMag](#) class.

Returns

Instance of [IDT_SecureMag](#)

The documentation for this class was generated from the following file:

- /Users/randy/Repo/universal-sdk/IDTechSDK_STD/DeviceHeaders/IDT_SecureMag.cs

21.14 IDTechSDK.IDT_SpectrumPro Class Reference

Public Member Functions

- RETURN_CODE [pin_getPIN](#) (int mode, int PANSource, string iccPAN, int startTimeout, int entryTimeout, string language, string ident="")

- RETURN_CODE [pin_cancelPINEntry](#) (string ident="")
- RETURN_CODE [pin_passThroughMode](#) (bool enable, string ident="")
- RETURN_CODE [device_sendDataCommand](#) (string cmd, bool calcLRC, ref byte[] response, string ident="")
- RETURN_CODE [device_sendDataCommand_ext](#) (string cmd, bool calcLRC, ref byte[] response, int timeout, bool noResponse, string ident="")
- RETURN_CODE [device_sendMacDataCommand](#) (byte taskID, byte[] functionID, byte[] data, bool macData, ref byte[] response, string ident="")
- RETURN_CODE [device_getFirmwareVersion](#) (ref string response, string ident="")
- RETURN_CODE [device_getVersions](#) (ref SpectrumInfoExt info, string ident="")
- RETURN_CODE [device_getNonce](#) (byte[] hostnonce, ref byte[] devicenonce, string ident="")
- RETURN_CODE [device_setUID](#) (string UID, int keyType, ref SpectrumInfo info, int sessionTimeout=0x0064, string ident="")
- RETURN_CODE [device_startRKI](#) (bool isTest, string ident="")
- RETURN_CODE [device_updateFirmware](#) (byte[] firmwareData, string firmwareName, int encryptionType, byte[] keyBlob, string ident="")
- RETURN_CODE [device_setSpectrumProBDK](#) (string BDK, string ident="")
- RETURN_CODE [device_pollCardReader](#) (ref byte[] status, string ident="")
- RETURN_CODE [device_controlUserInterface](#) (byte[] values, string ident="")
- RETURN_CODE [device_cardNotification](#) (bool enable, byte option, bool captureMSR, string ident="")
- RETURN_CODE [msr_clearMSRData](#) (string ident="")
- RETURN_CODE [device_sendPAE](#) (string command, ref string response, int timeout, string ident="")
- RETURN_CODE [msr_getMSRData](#) (ref IDTTransactionData card, string ident="")
- RETURN_CODE [config_getSerialNumber](#) (ref string response, string ident="")
- RETURN_CODE [config_getModelNumber](#) (ref string response, string ident="")
- RETURN_CODE [msr_startMSRSwipe](#) (int timeout, string ident="")
- RETURN_CODE [msr_cancelMSRSwipe](#) (string ident="")
- RETURN_CODE [emv_getTerminalID](#) (ref string response, string ident="")
- RETURN_CODE [emv_setTerminalID](#) (string terminalID, string ident="")
- RETURN_CODE [emv_setTerminalMajorConfiguration](#) (int configuration, string ident="")
- RETURN_CODE [emv_getTerminalMajorConfiguration](#) (ref int configuration, string ident="")
- RETURN_CODE [emv_cancelTransaction](#) (string ident="")
- RETURN_CODE [emv_clearTransactionLog](#) (string ident="")
- RETURN_CODE [emv_startTransaction](#) (double amount, double amtOther, int exponent, int type, int timeout, byte[] tags, bool forceOnline, bool isFastEMV=false, string ident="")
- RETURN_CODE [emv_activateTransaction](#) (int timeout, byte[] tags, bool forceOnline, bool isFastEMV=false, string ident="")
- RETURN_CODE [device_startTransaction](#) (double amount, double amtOther, int exponent, int type, int timeout, byte[] tags, bool forceOnline, bool isFastEMV=false, string ident="")
- RETURN_CODE [device_activateTransaction](#) (int timeout, byte[] tags, bool forceOnline, bool isFastEMV=false, string ident="")
- RETURN_CODE [emv_authenticateTransaction](#) (byte[] updatedTLV, string ident="")
- RETURN_CODE [emv_completeTransaction](#) (bool commError, byte[] authCode, byte[] iad, byte[] tlvScripts, byte[] tlv, string ident="")
- RETURN_CODE [emv_callbackResponseLCD](#) (EMV_LCD_DISPLAY_MODE type, byte selection, string ident="")
- RETURN_CODE [emv_callbackResponsePIN](#) (EMV_PIN_MODE type, byte[] KSN, byte[] PIN, string ident="")
- RETURN_CODE [emv_callbackResponseMSR](#) (byte[] MSR, string ident="")
- RETURN_CODE [emv_getEMVKernelVersion](#) (ref string response, string ident="")
- RETURN_CODE [emv_getEMVKernelCheckValue](#) (ref string response, string ident="")
- RETURN_CODE [emv_getEMVConfigurationCheckValue](#) (ref string response, string ident="")
- RETURN_CODE [emv_retrieveTransactionResult](#) (byte[] tags, ref IDTTransactionData tlv, string ident="")
- RETURN_CODE [device_rebootDevice](#) (string ident="")
- RETURN_CODE [emv_removeApplicationData](#) (byte[] AID, string ident="")
- RETURN_CODE [emv_removeAllApplicationData](#) (string ident="")
- RETURN_CODE [emv_removeCAPK](#) (byte[] capk, string ident="")

- RETURN_CODE [emv_removeAllCAPK](#) (string ident="")
- RETURN_CODE [emv_removeCRL](#) (byte[] crlList, string ident="")
- RETURN_CODE [emv_removeAllCRL](#) (string ident="")
- RETURN_CODE [emv_removeTerminalData](#) (string ident="")
- RETURN_CODE [emv_retrieveAIDList](#) (ref byte[][] response, string ident="")
- RETURN_CODE [emv_retrieveApplicationData](#) (byte[] AID, ref byte[] tlv, string ident="")
- RETURN_CODE [emv_retrieveCAPK](#) (byte[] capk, ref byte[] key, string ident="")
- RETURN_CODE [emv_retrieveCAPKList](#) (ref byte[] keys, string ident="")
- RETURN_CODE [emv_retrieveCRLList](#) (ref byte[] list, string ident="")
- RETURN_CODE [emv_retrieveTerminalData](#) (ref byte[] tlv, string ident="")
- RETURN_CODE [emv_setApplicationData](#) (byte[] name, byte[] tlv, string ident="")
- RETURN_CODE [emv_setCAPK](#) (byte[] key, string ident="")
- RETURN_CODE [emv_setCRL](#) (byte[] list, string ident="")
- RETURN_CODE [emv_trySetTerminalData](#) (byte[] tlv, ref byte[] rejectedTLV, ref byte[] convertedTLV, bool overwrite=false, string ident="")
- RETURN_CODE [emv_setTerminalData](#) (byte[] tlv, string ident="")
- RETURN_CODE [emv_addTerminalData](#) (byte[] tlv, string ident="")
- string [device_getResponseCodeString](#) (RETURN_CODE eCode)
- RETURN_CODE [device_getSpectrumProKSN](#) (int type, ref byte[] KSN, string ident="")
- RETURN_CODE [icc_powerOffICC](#) (string ident="")
- RETURN_CODE [icc_powerOnICC](#) (ref byte[] ATR, string ident="")
- RETURN_CODE [icc_getICCReaderStatus](#) (ref byte status, string ident="")
- RETURN_CODE [icc_getICCStatus](#) (ref byte[] status, ref byte[] atr, string ident="")
- RETURN_CODE [device_StartRKI](#) (int type, string ident="")
- bool [createFastEMVData](#) (ref IDTTransactionData cData, string ident="")
- RETURN_CODE [device_getRKIStatus](#) (bool isProd, bool isMultiKey, ref string status, ref Dictionary< string, RKI_KEY_TYPE > keys, string ident="")
- RETURN_CODE [device_getAnyRKIStatus](#) (bool isMultiKey, ref string status, ref Dictionary< string, RKI_KEY_TYPE > keys, string ident="")
- RETURN_CODE [device_sendConfigurationFromZip](#) (byte[] zip, string filename, VC_OPERATION_TYPE type, bool matchFW, string ident="", bool isForeground=false)
- RETURN_CODE [device_sendConfiguration](#) (string filename, VC_OPERATION_TYPE type, bool matchFW, string ident="", bool isForeground=false)
- RETURN_CODE [device_readConfigurationToMemory](#) (string memo, Configs.InstallRules rules, List< Configs.DeviceCommand > cmds, string ident="", bool isForeground=false)
- RETURN_CODE [device_getConfigurationFromMemory](#) (ref string json, string ident="")
- RETURN_CODE [device_RemoteKeyInjection](#) (RKI_KEY_TYPE type, string keyName, string ident="", bool performOnForeground=false)

Static Public Member Functions

- static bool [useSerialPort](#) (int port)
- static bool [useSerialPort](#) (int port, int baud)
- static bool [useSerialPortLinux](#) (string path)
- static bool [useSerialPortLinux](#) (string path, int baud)
- static int [getCommandTimeout](#) (string ident="")
- static void [setCommandTimeout](#) (int milliseconds, string ident="")
- static void [initSC](#) ()
- static void [setCallback](#) (Callback my_Callback)
- static void [setCardNotificationCallback](#) (CardNotificationCallback my_Callback)
- static void [setCallback](#) (IntPtr my_Callback, SynchronizationContext context)
- static String [SDK_Version](#) ()
- static void [emv_autoAuthenticate](#) (bool authenticate, string ident="")
- static void [emv_autoAuthenticateTags](#) (bool authenticate, byte[] tags, string ident="")
- static void [emv_allowFallback](#) (bool allow, string ident="")
- static void [lcd_retrieveMessage](#) (DisplayMessages.DISPLAY_MESSAGE_LANGUAGE lang, DisplayMessages.DISPLAY_MESSAGE_IDENTIFIER id, ref string line1, ref string line2)

Properties

- static [IDT_SpectrumPro SharedController](#) [get]

21.14.1 Detailed Description

Class for [IDT_SpectrumPro](#) MSR/ICC reader

21.14.2 Member Function Documentation

21.14.2.1 **RETURN_CODE** IDTechSDK.IDT_SpectrumPro.config_getModelNumber (ref string *response*, string *ident* = " ")

Polls device for Model Number

Parameters

<i>response</i>	Returns Model Number
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString

21.14.2.2 **RETURN_CODE** IDTechSDK.IDT_SpectrumPro.config_getSerialNumber (ref string *response*, string *ident* = " ")

Polls device for Serial Number

Parameters

<i>response</i>	Returns Serial Number
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString

21.14.2.3 **bool** IDTechSDK.IDT_SpectrumPro.createFastEMVData (ref IDTTransactionData *cData*, string *ident* = " ")

Create Fast EMV Data

At the completion of a Fast EMV Transaction, after the final card decision is returned and the IDTTransactionData object is provided, sending that cData object to this method will populate the .fastEMV element with string data that represents the Fast EMV data that would be returned from an IDTech FastEMV over KB protocol

Parameters

<i>cData</i>	The IDTTransactionData object populated with card data.
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with `errorCode.getErrorString(string ident = "")`

21.14.2.4 RETURN_CODE IDTechSDK.IDT_SpectrumPro.device_activateTransaction (int *timeout*, byte[] *tags*, bool *forceOnline*, bool *isFastEMV* = false, string *ident* = " ")

Start EMV Transaction Request

Authorizes the EMV transaction for an ICC card

The tags will be returned in the callback routine.

Parameters

<i>timeout</i>	Timeout value in seconds.
<i>tags</i>	The tags to be included in the request. Passed as TLV Data. Example, tag 9F02 with amount 0x000000000100 would be 0x9F0206000000000100
<i>forceOnline</i>	TRUE = do not allow offline approval, FALSE = allow ICC to approve offline if terminal capable Note: To request tags to be included in default response, use tag DFEE1A, and specify tag list. Example four tags 9F02, 9F36, 95, 9F37 to be included in response = DFEE1A079F029F369F37
<i>isFastEMV</i>	If TRUE, it will populate the IDTTransactionData.fastEMV with ASCII data similar to IDTech FastEMV KB output, after performing an auto-authenticate and auto-complete with ResultCode = Could Not Contact Host
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with `device_getResponseCodeString`

21.14.2.5 RETURN_CODE IDTechSDK.IDT_SpectrumPro.device_cardNotification (bool *enable*, byte *option*, bool *captureMSR*, string *ident* = " ")

Card Notifications

Enable Insert/Removal notifications

Notification returned to CardNotificationCallback, use [setCardNotificationCallback\(CardNotificationCallback callback\)](#). CardNotificationCallback (byte status, string ident = "") 0x01 = bad msr 0x02 = good msr 0x03 = required track not captured 0x04 = front switch 0x08 = seat switch

Parameters

<i>enable</i>	TRUE = enabled
<i>option</i>	bit 1 = T1 required, b2 = T2 required, b3 = T2 required
<i>captureMSR</i>	TRUE = automatically capture MSR data and return as DeviceState.TransactionData to MessageCallback
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with `errorCode.getErrorString(string ident = "")`

21.14.2.6 RETURN_CODE IDTechSDK.IDT_SpectrumPro.device_controlUserInterface (byte[] values, string ident = " ")

Control User Interface

Controls the User Interface: Display, Beep, LED

```
@param values Four bytes to control the user interface
Byte[0] = LCD Message
Messages 00-07 are normally controlled by the reader.
- 00h: Idle Message (Welcome, string ident = "")
- 01h: Present card (Please Present Card, string ident = "")
- 02h: Time Out or Transaction cancel (No Card, string ident = "")
- 03h: Transaction between reader and card is in the middle (Processing...)
- 04h: Transaction Pass (Thank You, string ident = "")
- 05h: Transaction Fail (Fail, string ident = "")
- 06h: Amount (Amount $ 0.00 Tap Card, string ident = "")
- 07h: Balance or Offline Available funds (Balance $ 0.00) Messages 08-0B are controlled by the terminal
- 08h: Insert or Swipe card (Use Chip & PIN, string ident = "")
- 09h: Try Again(Tap Again, string ident = "")
- 0Ah: Tells the customer to present only one card (Present 1 card only)
- 0Bh: Tells the customer to wait for authentication/authorization (Wait)
- FFh: indicates the command is setting the LED/Buzzer only.
Byte[1] = Beep Indicator
- 00h: No beep
- 01h: Single beep
- 02h: Double beep
- 03h: Three short beeps
- 04h: Four short beeps
- 05h: One long beep of 200 ms
- 06h: One long beep of 400 ms
- 07h: One long beep of 600 ms
- 08h: One long beep of 800 ms
Byte[2] = LED Number
- 00h: LED 0 (Power LED) 01h: LED 1
- 02h: LED 2
- 03h: LED 3
- FFh: All LEDs
Byte[3] = LED Status
- 00h: LED Off
- 01h: LED On
```

Parameters

<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.
--------------	--

Returns

RETURN_CODE: Values can be parsed with `errorCode.getErrorString(string ident = "")`

21.14.2.7 RETURN_CODE IDTechSDK.IDT_SpectrumPro.device_getAnyRKIStatus (bool *isMultiKey*, ref string *status*, ref Dictionary< string, RKI_KEY_TYPE > *keys*, string *ident* = " ")

Get RKI Status Polls the RKI servers to validate device RKI eligibility. Note: if device type is known in advance (production or demo device), it is more efficient to use `device_getRKIStatus` instead

Parameters

<i>isMultiKey</i>	True = MultiKey, False = SingleKey
<i>status</i>	RKI status returned as a string
<i>keys</i>	Returns all available keys
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with `errorCode.getErrorString()`. When no data is available, return code = RETURN_CODE_NO_DATA_AVAILABLE

21.14.2.8 RETURN_CODE IDTechSDK.IDT_SpectrumPro.device_getConfigurationFromMemory (ref string *json*, string *ident* = " ")

Read Device Configuration From Memory Retrieves the ViVOconfig configuration data from memory after a successful device_readConfigurationToMemory

Parameters

<i>json</i>	Returns the .json file data that can be saved to disk
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with `errorCode.getErrorString()`. When no data is available, return code = RETURN_CODE_NO_DATA_AVAILABLE

21.14.2.9 RETURN_CODE IDTechSDK.IDT_SpectrumPro.device_getFirmwareVersion (ref string *response*, string *ident* = " ")

Polls device for Firmware Version

Parameters

<i>response</i>	Response returned of Firmware Version
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with `errorCode.getErrorString(string ident = "")`

21.14.2.10 RETURN_CODE IDTechSDK.IDT_SpectrumPro.device_getNonce (byte[] *hostnonce*, ref byte[] *devicenonce*, string *ident* = " ")

Get Device Nonce

Parameters

<i>hostnonce</i>	Nonce of host
<i>devicenonce</i>	Nonce of device
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with `errorCode.getErrorString(string ident = "")`

21.14.2.11 string IDTechSDK.IDT_SpectrumPro.device_getResponseCodeString (RETURN_CODE *eCode*)

Get the description of response result.

Parameters

<i>eCode</i>	the response result.
--------------	----------------------

Return values

<i>the</i>	string for description of response result
------------	---

- case 0: "no error, beginning task";
- case 1: "no response from reader";
- case 2: "invalid response data";
- case 3: "time out for task or CMD";
- case 4: "wrong parameter";
- case 5: "SDK is doing MSR or ICC task";
- case 6: "SDK is doing PINPad task";
- case 7: "SDK is doing CTLS task";
- case 8: "SDK is doing EMV task";
- case 9: "SDK is doing Other task";
- case 10: "err response or data";
- case 11: "no reader attached";
- case 12: "mono audio is enabled";
- case 13: "did connection";
- case 14: "audio volume is too low";
- case 15: "task or CMD be canceled";
- case 16: "UF wrong string format";
- case 17: "UF file not found";
- case 18: "UF wrong file format";
- case 19: "Attempt to contact online host failed";
- case 20: "Attempt to perform RKI failed";
- case 0x300: "Key Type(TDES) of Session Key is not same as the related Master Key.";
- case 0x400: "Related Key was not loaded.";
- case 0x500: "Key Same.";
- case 0x501: "Key is all zero";
- case 0x502: "TR-31 format error";
- case 0x702: "PAN is Error Key.";

- case 0x705: "No Internal MSR PAN (or Internal MSR PAN is erased timeout)";
- case 0X0C01: "Incorrect Frame Tag";
- case 0X0C02: "Incorrect Frame Type";
- case 0X0C03: "Unknown Frame Type";
- case 0X0C04: "Unknown Command";
- case 0X0C05: "Unknown Sub-Command";
- case 0X0C06: "CRC Error";
- case 0X0C07: "Failed";
- case 0X0C08: "Timeout";
- case 0X0C0A: "Incorrect Parameter";
- case 0X0C0B: "Command Not Supported";
- case 0X0C0C: "Sub-Command Not Supported";
- case 0X0C0D: "Parameter Not Supported / Status Abort Command";
- case 0X0C0F: "Sub-Command Not Allowed";
- case 0X0D01: "Incorrect Header Tag";
- case 0X0D02: "Unknown Command";
- case 0X0D03: "Unknown Sub-Command";
- case 0X0D04: "CRC Error in Frame";
- case 0X0D05: "Incorrect Parameter";
- case 0X0D06: "Parameter Not Supported";
- case 0X0D07: "Mal-formatted Data";
- case 0X0D08: "Timeout";
- case 0X0D0A: "Failed / NACK";
- case 0X0D0B: "Command not Allowed";
- case 0X0D0C: "Sub-Command not Allowed";
- case 0X0D0D: "Buffer Overflow (Data Length too large for reader buffer)";
- case 0X0D0E: "User Interface Event";
- case 0X0D11: "Communication type not supported, VT-1, burst, etc.";
- case 0X0D12: "Secure interface is not functional or is in an intermediate state.";
- case 0X0D13: "Data field is not mod 8";
- case 0X0D14: "Pad 0x80 not found where expected";
- case 0X0D15: "Specified key type is invalid";
- case 0X0D1: "Could not retrieve key from the SAM(InitSecureComm)";
- case 0X0D17: "Hash code problem";
- case 0X0D18: "Could not store the key into the SAM(InstallKey)";
- case 0X0D19: "Frame is too large";

- case 0X0D1A: "Unit powered up in authentication state but POS must resend the InitSecureComm command";
- case 0X0D1B: "The EEPROM may not be initialized because SecCommInterface does not make sense";
- case 0X0D1C: "Problem encoding APDU";
- case 0X0D20: "Unsupported Index(ILM) SAM Transceiver error – problem communicating with the SAM(Key Mgr)";
- case 0X0D2: "Unexpected Sequence Counter in multiple frames for single bitmap(ILM) Length error in data returned from the SAM(Key Mgr)";
- case 0X0D22: "Improper bit map(ILM)";
- case 0X0D23: "Request Online Authorization";
- case 0X0D24: "ViVOCard3 raw data read successful";
- case 0X0D25: "Message index not available(ILM) ViVOcomm activate transaction card type(ViVOcomm)";
- case 0X0D26: "Version Information Mismatch(ILM)";
- case 0X0D27: "Not sending commands in correct index message index(ILM)";
- case 0X0D28: "Time out or next expected message not received(ILM)";
- case 0X0D29: "ILM languages not available for viewing(ILM)";
- case 0X0D2A: "Other language not supported(ILM)";
- case 0X0D41: "Unknown Error from SAM";
- case 0X0D42: "Invalid data detected by SAM";
- case 0X0D43: "Incomplete data detected by SAM";
- case 0X0D44: "Reserved";
- case 0X0D45: "Invalid key hash algorithm";
- case 0X0D46: "Invalid key encryption algorithm";
- case 0X0D47: "Invalid modulus length";
- case 0X0D48: "Invalid exponent";
- case 0X0D49: "Key already exists";
- case 0X0D4A: "No space for new RID";
- case 0X0D4B: "Key not found";
- case 0X0D4C: "Crypto not responding";
- case 0X0D4D: "Crypto communication error";
- case 0X0D4E: "Module-specific error for Key Manager";
- case 0X0D4F: "All key slots are full (maximum number of keys has been installed)";
- case 0X0D50: "Auto-Switch OK";
- case 0X0D51: "Auto-Switch failed";
- case 0X0D90: "Account DUKPT Key not exist";
- case 0X0D91: "Account DUKPT Key KSN exhausted";
- case 0x0D00: "This Key had been loaded.";

- case 0x0E00: "Base Time was loaded.";
- case 0x0F00: "Encryption Or Decryption Failed.";
- case 0x1000: "Battery Low Warning (It is High Priority Response while Battery is Low.)";
- case 0x1800: "Send "Cancel Command" after send "Get Encrypted PIN" &"Get Numeric "& "Get Amount"";
- case 0x1900: "Press "Cancel" key after send "Get Encrypted PIN" &"Get Numeric "& "Get Amount"";
- case 0x30FF: "Security Chip is not connect";
- case 0x3000: "Security Chip is deactivation & Device is In Removal Legally State.";
- case 0x3101: "Security Chip is activation & Device is In Removal Legally State.";
- case 0x5500: "No Admin DUKPT Key.";
- case 0x5501: "Admin DUKPT Key STOP.";
- case 0x5502: "Admin DUKPT Key KSN is Error.";
- case 0x5503: "Get Authentication Code1 Failed.";
- case 0x5504: "Validate Authentication Code Error.";
- case 0x5505: "Encrypt or Decrypt data failed.";
- case 0x5506: "Not Support the New Key Type.";
- case 0x5507: "New Key Index is Error.";
- case 0x5508: "Step Error.";
- case 0x5509: "KSN Error";
- case 0x550A: "MAC Error.";
- case 0x550B: "Key Usage Error.";
- case 0x550C: "Mode Of Use Error.";
- case 0x550F: "Other Error.";
- case 0x6000: "Save or Config Failed / Or Read Config Error.";
- case 0x6200: "No Serial Number.";
- case 0x6900: "Invalid Command - Protocol is right, but task ID is invalid.";
- case 0x6A01: "Unsupported Command – Protocol and task ID are right, but command is invalid – In this State";
- case 0x6A00: "Unsupported Command - Protocol and task ID are right, but command is invalid.";
- case 0x6B00: "Unknown parameter in command - Protocol task ID and command are right, but parameter is invalid.";
- case 0x6C00: "Unknown parameter in command – Protocol task ID and command are right, but length is out of the requirement.";
- case 0x7200: "Device is suspend (MKS suspend or press password suspend).";
- case 0x7300: "PIN DUKPT is STOP (21 bit 1).";
- case 0x7400: "Device is Busy.";
- case 0xE100: "Can not enter sleep mode";
- case 0xE200: "File has existed";

- case 0xE300: "File has not existed";
- case 0xE313: "IO line low -- Card error after session start";
- case 0xE400: "Open File Error";
- case 0xE500: "SmartCard Error";
- case 0xE600: "Get MSR Card data is error";
- case 0xE700: "Command time out";
- case 0xE800: "File read or write is error";
- case 0xE900: "Active 1850 error!";
- case 0xEA00: "Load bootloader error";
- case 0xEF00: "Protocol Error- STX or ETX or check error.";
- case 0xEB00: "Picture is not exist";
- case 0x2C02: "No Microprocessor ICC seated";
- case 0x2C06: "no card seated to request ATR";
- case 0x2D01: "Card Not Supported,";
- case 0x2D03: "Card Not Supported, wants CRC";
- case 0x690D: "Command not supported on reader without ICC support";
- case 0x8100: "ICC error time out on power-up";
- case 0x8200: "invalid TS character received - Wrong operation step";
- case 0x8300: "Decode MSR Error";
- case 0x8400: "TriMagII no Response";
- case 0x8500: "No Swipe MSR Card";
- case 0x8510: "No Financial Card";
- case 0x8600: "Unsupported F, D, or combination of F and D";
- case 0x8700: "protocol not supported EMV TD1 out of range";
- case 0x8800: "power not at proper level";
- case 0x8900: "ATR length too long";
- case 0x8B01: "EMV invalid TA1 byte value";
- case 0x8B02: "EMV TB1 required";
- case 0x8B03: "EMV Unsupported TB1 only 00 allowed";
- case 0x8B04: "EMV Card Error, invalid BWI or CWI";
- case 0x8B06: "EMV TB2 not allowed in ATR";
- case 0x8B07: "EMV TC2 out of range";
- case 0x8B08: "EMV TC2 out of range";
- case 0x8B09: "per EMV96 TA3 must be > 0xF";
- case 0x8B10: "ICC error on power-up";
- case 0x8B11: "EMV T=1 then TB3 required";

- case 0x8B12: "Card Error, invalid BWI or CWI";
- case 0x8B13: "Card Error, invalid BWI or CWI";
- case 0x8B17: "EMV TC1/TB3 conflict-";
- case 0x8B20: "EMV TD2 out of range must be T=1";
- case 0x8C00: "TCK error";
- case 0xA304: "connector has no voltage setting";
- case 0xA305: "ICC error on power-up invalid (SBLK(IFSD) exchange";
- case 0xE301: "ICC error after session start";
- case 0xFF00: "Request to go online";
- case 0xFF01: "EMV: Accept the offline transaction";
- case 0xFF02: "EMV: Decline the offline transaction";
- case 0xFF03: "EMV: Accept the online transaction";
- case 0xFF04: "EMV: Decline the online transaction";
- case 0xFF05: "EMV: Application may fallback to magstripe technology";
- case 0xFF06: "EMV: ICC detected tah the conditions of use are not satisfied";
- case 0xFF07: "EMV: ICC didn't accept transaction";
- case 0xFF08: "EMV: Transaction was cancelled";
- case 0xFF09: "EMV: Application was not selected by kernel or ICC format error or ICC missing data error";
- case 0xFF0A: "EMV: Transaction is terminated";
- case 0xFF0B: "EMV: Other EMV Error";
- case 0xFFFF: "NO RESPONSE";
- case 0xF002: "ICC communication timeout";
- case 0xF003: "ICC communication Error";
- case 0xF00F: "ICC Card Seated and Highest Priority, disable MSR work request";
- case 0xF200: "AID List / Application Data is not exist";
- case 0xF201: "Terminal Data is not exist";
- case 0xF202: "TLV format is error";
- case 0xF203: "AID List is full";
- case 0xF204: "Any CA Key is not exist";
- case 0xF205: "CA Key RID is not exist";
- case 0xF206: "CA Key Index it not exist";
- case 0xF207: "CA Key is full";
- case 0xF208: "CA Key Hash Value is Error";
- case 0xF209: "Transaction format error";
- case 0xF20A: "The command will not be processing";
- case 0xF20B: "CRL is not exist";

- case 0xF20C: "CRL number exceed max number";
- case 0xF20D: "Amount,Other Amount,Trasaction Type are missing";
- case 0xF20E: "The Identification of algorithm is mistake";
- case 0xF20F: "No Financial Card";
- case 0xF210: "In Encrypt Result state, TLV total Length is greater than Max Length";
- case 0x1001: "INVALID ARG";
- case 0x1002: "FILE_OPEN_FAILED";
- case 0x1003: "FILE OPERATION_FAILED";
- case 0x2001: "MEMORY_NOT_ENOUGH";
- case 0x3002: "SMARTCARD_FAIL";
- case 0x3003: "SMARTCARD_INIT_FAILED";
- case 0x3004: "FALLBACK_SITUATION";
- case 0x3005: "SMARTCARD_ABSENT";
- case 0x3006: "SMARTCARD_TIMEOUT";
- case 0x5001: "EMV_PARSING_TAGS_FAILED";
- case 0x5002: "EMV_DUPLICATE_CARD_DATA_ELEMENT";
- case 0x5003: "EMV_DATA_FORMAT_INCORRECT";
- case 0x5004: "EMV_NO_TERM_APP";
- case 0x5005: "EMV_NO_MATCHING_APP";
- case 0x5006: "EMV_MISSING_MANDATORY_OBJECT";
- case 0x5007: "EMV_APP_SELECTION_RETRY";
- case 0x5008: "EMV_GET_AMOUNT_ERROR";
- case 0x5009: "EMV_CARD_REJECTED";
- case 0x5010: "EMV_AIP_NOT_RECEIVED";
- case 0x5011: "EMV_AFL_NOT_RECEIVED";
- case 0x5012: "EMV_AFL_LEN_OUT_OF_RANGE";
- case 0x5013: "EMV_SFI_OUT_OF_RANGE";
- case 0x5014: "EMV_AFL_INCORRECT";
- case 0x5015: "EMV_EXP_DATE_INCORRECT";
- case 0x5016: "EMV_EFF_DATE_INCORRECT";
- case 0x5017: "EMV_ISS_COD_TBL_OUT_OF_RANGE";
- case 0x5018: "EMV_CRYPTOGAM_TYPE_INCORRECT";
- case 0x5019: "EMV_PSE_NOT_SUPPORTED_BY_CARD";
- case 0x5020: "EMV_USER_SELECTED_LANGUAGE";
- case 0x5021: "EMV_SERVICE_NOT_ALLOWED";
- case 0x5022: "EMV_NO_TAG_FOUND";

- case 0x5023: "EMV_CARD_BLOCKED";
- case 0x5024: "EMV_LEN_INCORRECT";
- case 0x5025: "CARD_COM_ERROR";
- case 0x5026: "EMV_TSC_NOT_INCREASED";
- case 0x5027: "EMV_HASH_INCORRECT";
- case 0x5028: "EMV_NO_ARC";
- case 0x5029: "EMV_INVALID_ARC";
- case 0x5030: "EMV_NO_ONLINE_COMM";
- case 0x5031: "TRAN_TYPE_INCORRECT";
- case 0x5032: "EMV_APP_NO_SUPPORT";
- case 0x5033: "EMV_APP_NOT_SELECT";
- case 0x5034: "EMV_LANG_NOT_SELECT";
- case 0x5035: "EMV_NO_TERM_DATA";
- case 0x6001: "CVM_TYPE_UNKNOWN";
- case 0x6002: "CVM_AIP_NOT_SUPPORTED";
- case 0x6003: "CVM_TAG_8E_MISSING";
- case 0x6004: "CVM_TAG_8E_FORMAT_ERROR";
- case 0x6005: "CVM_CODE_IS_NOT_SUPPORTED";
- case 0x6006: "CVM_COND_CODE_IS_NOT_SUPPORTED";
- case 0x6007: "NO_MORE_CVM";
- case 0x6008: "PIN_BYPASSED_BEFORE";
- case 0x7001: "PK_BUFFER_SIZE_TOO_BIG";
- case 0x7002: "PK_FILE_WRITE_ERROR";
- case 0x7003: "PK_HASH_ERROR";
- case 0x8001: "NO_CARD_HOLDER_CONFIRMATION";
- case 0x8002: "GET_ONLINE_PIN";
- case 0xD000: "Data not exist";
- case 0xD001: "Data access error";
- case 0xD100: "RID not exist";
- case 0xD101: "RID existed";
- case 0xD102: "Index not exist";
- case 0xD200: "Maximum exceeded";
- case 0xD201: "Hash error";
- case 0xD205: "System Busy";
- case 0x0E01: "Unable to go online";
- case 0x0E02: "Technical Issue";

- case 0x0E03: "Declined";
- case 0x0E04: "Issuer Referral transaction";
- case 0x0F01: "Decline the online transaction";
- case 0x0F02: "Request to go online";
- case 0x0F03: "Transaction is terminated";
- case 0x0F05: "Application was not selected by kernel or ICC format error or ICC missing data error";
- case 0x0F07: "ICC didn't accept transaction";
- case 0x0F0A: "Application may fallback to magstripe technology";
- case 0x0F0C: "Transaction was cancelled";
- case 0x0F0D: "Timeout";
- case 0x0F0F: "Other EMV Error";
- case 0x0F10: "Accept the offline transaction";
- case 0x0F11: "Decline the offline transaction";
- case 0x0F21: "ICC detected tah the conditions of use are not satisfied";
- case 0x0F22: "No app were found on card matching terminal configuration";
- case 0x0F23: "Terminal file does not exist";
- case 0x0F24: "CAPK file does not exist";
- case 0x0F25: "CRL Entry does not exist";
- case 0x0FFE: "code when blocking is disabled";
- case 0x0FFF: "code when command is not applicable on the selected device";
- case 0xF005: "ICC Encrypted C-APDU Data Structure Length Error Or Format Error.";
- case 0xBBE0: "CM100 Success";
- case 0xBBE1: "CM100 Parameter Error";
- case 0xBBE2: "CM100 Low Output Buffer";
- case 0xBBE3: "CM100 Card Not Found";
- case 0xBBE4: "CM100 Collision Card Exists";
- case 0xBBE5: "CM100 Too Many Cards Exist";
- case 0xBBE6: "CM100 Saved Data Does Not Exist";
- case 0xBBE8: "CM100 No Data Available";
- case 0xBBE9: "CM100 Invalid CID Returned";
- case 0xBBEA: "CM100 Invalid Card Exists";
- case 0xBBEC: "CM100 Command Unsupported";
- case 0xBBED: "CM100 Error In Command Process";
- case 0xBBEE: "CM100 Invalid Command";
- case 0X9031: "Unknown command";
- case 0X9032: "Wrong parameter (such as the length of the command is incorrect)";

- case 0X9038: "Wait (the command couldn't be finished in BWT)";
- case 0X9039: "Busy (a previously command has not been finished)";
- case 0X903A: "Number of retries over limit";
- case 0X9040: "Invalid Manufacturing system data";
- case 0X9041: "Not authenticated";
- case 0X9042: "Invalid Master DUKPT Key";
- case 0X9043: "Invalid MAC Key";
- case 0X9044: "Reserved for future use";
- case 0X9045: "Reserved for future use";
- case 0X9046: "Invalid DATA DUKPT Key";
- case 0X9047: "Invalid PIN Pairing DUKPT Key";
- case 0X9048: "Invalid DATA Pairing DUKPT Key";
- case 0X9049: "No nonce generated";
- case 0X9949: "No GUID available. Perform getVersion first.";
- case 0X9950: "MAC Calculation unsuccessful. Check BDK value.";
- case 0X904A: "Not ready";
- case 0X904B: "Not MAC data";
- case 0X9050: "Invalid Certificate";
- case 0X9051: "Duplicate key detected";
- case 0X9052: "AT checks failed";
- case 0X9053: "TR34 checks failed";
- case 0X9054: "TR31 checks failed";
- case 0X9055: "MAC checks failed";
- case 0X9056: "Firmware download failed";
- case 0X9060: "Log is full";
- case 0X9061: "Removal sensor unengaged";
- case 0X9062: "Any hardware problems";
- case 0X9070: "ICC communication timeout";
- case 0X9071: "ICC data error (such check sum error)";
- case 0X9072: "Smart Card not powered up";

21.14.2.12 **RETURN_CODE** IDTechSDK.IDT_SpectrumPro.device_getRKIStatus (bool *isProd*, bool *isMultiKey*, ref string *status*, ref Dictionary< string, RKI_KEY_TYPE > *keys*, string *ident* = " ")

Get RKI Status From Specified Server Polls the RKI server to validate device RKI eligibility

Parameters

<i>isProd</i>	TRUE = poll production RKI server, FALSE = poll demo RKI server
<i>isMultiKey</i>	True = MultiKey, False = SingleKey
<i>status</i>	RKI status returned as a string
<i>keys</i>	Returns dictionary of available keys
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with `errorCode.getErrorString()`. When no data is available, return code = RETURN_CODE_NO_DATA_AVAILABLE

21.14.2.13 RETURN_CODE IDTechSDK.IDT_SpectrumPro.device_getSpectrumProKSN (int *type*, ref byte[] *KSN*, string *ident* = " ")

Get DUKPT KSN

Returns the KSN for the provided key index

Parameters

<i>type</i>	Key type: <ul style="list-style-type: none"> • 0: Key Encryption Key (Master Key or KEK, string ident = "") • 2: Data Encryption Key (DEK, string ident = "") • 5: MAC Key (MAK, string ident = "") • 10: RKL Key Encryption Key (REK, string ident = "") • 20: HSM DUKPT Key
<i>KSN</i>	Key Serial Number
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with `device_getResponseCodeString`

21.14.2.14 RETURN_CODE IDTechSDK.IDT_SpectrumPro.device_getVersions (ref SpectrumInfoExt *info*, string *ident* = " ")

Poll device for Version Info

Parameters

<i>info</i>	Version info
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with `errorCode.getErrorString(string ident = "")`

21.14.2.15 RETURN_CODE IDTechSDK.IDT_SpectrumPro.device_pollCardReader (ref byte[] status, string ident = " ")

Poll Card Reader

Provides information about the state of the Card Reader

@param status Six bytes indicating card reader information

Byte 0:

- Bit 0: Device Manufacturing CA data valid
- Bit 1: Device Manufacturing Secure data valid
- Bit 2: HOST_CR_MASTER_DUKPT Key valid
- Bit 3: HOST_CR_MAC Keys valid (Authenticated, string ident = "")
- Bit 4: RFU
- Bit 5: RFU
- Bit 6: DATA_DUKPT Key Valid
- Bit 7: Key is initialized (MFK and RSA Key pairs, string ident = "")

Byte 1:

- Bit 0: Firmware Key Valid
- Bit 1: RFU
- Bit 2: CR_PINPAD_MASTER_DUKPT Key valid
- Bit 3: CR_PINPAD_MAC Keys valid (Authenticated, string ident = "")
- Bit 4: DATA Pairing DUKPT Key valid
- Bit 5: PIN Pairing DUKPT Key Valid
- Bit 6: RFU
- Bit 7: RFU

Byte 2:

- Bit 0: RFU
- Bit 1: Tamper Switch #1 Error
- Bit 2: Battery Backup Error
- Bit 3: Temperature Error
- Bit 4: Voltage Sensor Error
- Bit 5: Firmware Authentication Error
- Bit 6: Tamper Switch #2 Error
- Bit 7: Removal Tamper Error

Byte 3:

- Battery Voltage (example 0x32 = 3.2V, 0x24 = 2.4V, string ident = "")

Byte 4:

- Bit 0: Log is Full
- Bit 1: Mag Data Present
- Bit 2: Card Insert
- Bit 3: Removal Sensor connected
- Bit 4: Card Seated
- Bit 5: Latch Mechanism Active
- Bit 6: Removal Sensor Active
- Bit 7: Tamper Detector Active

Byte 5:

- Bit 0: SAM Available
- Bit 1: Chip Card Reader Available
- Bit 2: Host Connected
- Bit 3: Contactless Available
- Bit 4: PINPAD connected
- Bit 5: MSR Header connected
- Bit 6: RFU
- Bit 7: Production Unit

Parameters

<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.
--------------	--

Returns

RETURN_CODE: Values can be parsed with `errorCode.getErrorString(string ident = "")`

21.14.2.16 **RETURN_CODE** IDTechSDK.IDT_SpectrumPro.device_readConfigurationToMemory (string *memo*, Configs.InstallRules *rules*, List< Configs.DeviceCommand > *cmds*, string *ident* = " ", bool *isForeground* = false)

Read Device Configuration To Memory Executes a ViVOconfig read of device settings and stores the results in memory

Once a ViVOconfig read starts, all messages will be returned to the callback as DeviceState.ViVOconfig. The ViVOconfig operation is complete when the return code from a DeviceState.ViVOconfig message is one of the following values:

- RETURN_CODE_SDK_VIVOCONFIG_READ_CONFIG_SUCCESS = Read process completed successfully. Execute device_getConfigurationFromMemory to retrieve the configuration data
- RETURN_CODE_SDK_VIVOCONFIG_READ_CONFIG_FAILED = Read process FAILED.

Parameters

<i>memo</i>	Optional: populates the memo field of the captured .json file
<i>rules</i>	Optional: populates the .json file with available install rules
<i>cmds</i>	Optional: populates the .json file with commands to execute
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.
<i>isForeground</i>	If TRUE, will block program until update complete. Otherwise, FALSE performs update on background.

Returns

RETURN_CODE: Values can be parsed with errorCode.getErrorString(). When no data is available, return code = RETURN_CODE_NO_DATA_AVAILABLE

21.14.2.17 **RETURN_CODE** IDTechSDK.IDT_SpectrumPro.device_rebootDevice (string *ident* = " ")

Reboot Device

Executes a command to restart the device.

- Card data is cleared, resetting card status bits.
- Response data of the previous command is cleared.
- Resetting firmware.

Parameters

<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.
--------------	--

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString

21.14.2.18 **RETURN_CODE** IDTechSDK.IDT_SpectrumPro.device_RemoteKeyInjection (RKI_KEY_TYPE *type*, string *keyName*, string *ident* = " ", bool *performOnForeground* = false)

Remote Key Injection Performs a remote key injection for the device

Parameters

<i>type</i>	Remote Key Injection Type
<i>keyName</i>	Name of key (optional)
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.
<i>performOnForeground</i>	TRUE = block during process (default is false)

Returns

RETURN_CODE: Values can be parsed with `errorCode.getErrorString()`. When no data is available, return code = RETURN_CODE_NO_DATA_AVAILABLE

21.14.2.19 RETURN_CODE IDTechSDK.IDT_SpectrumPro.device_sendConfiguration (string *filename*, VC_OPERATION_TYPE *type*, bool *matchFW*, string *ident* = " ", bool *isForeground* = false)

Send Configuration Executes a ViVOconfig update or verify to a device

Once a ViVOconfig write or verify starts, all messages will be returned to the callback as DeviceState.ViVOconfig. The ViVOconfig operation is complete when the return code from a DeviceState.ViVOconfig message is one of the following values:

- RETURN_CODE_SDK_VIVOCONFIG_VERIFY_SUCCESS = Verification process completed successfully. No differences found.
- RETURN_CODE_SDK_VIVOCONFIG_VERIFY_SUCCESS_WITH_WARNING = Verification process completed with warnings.
- RETURN_CODE_SDK_VIVOCONFIG_VERIFY_FAILED = Verification process FAILED
- RETURN_CODE_SDK_VIVOCONFIG_WRITE_SUCCESS = Write process completed successfully.
- RETURN_CODE_SDK_VIVOCONFIG_WRITE_SUCCESS_WITH_WARNING = Write process completed with warnings
- RETURN_CODE_SDK_VIVOCONFIG_WRITE_FAILED = Write process FAILED

Parameters

<i>filename</i>	The .json configuration file to update the device with NOTE: You can also use this parameter to pass the path of where the .json file is located at on local storage instead
<i>filename</i>	The name of the .json configuration file
<i>type</i>	VC_OPERATION_TYPE

- VC_OPERATION_TYPE_WRITE = Write File To Device, Hash must validate
- VC_OPERATION_TYPE_VERIFY = Verify Device With File
- VC_OPERATION_TYPE_WRITE_IGNORE_HASH = Write File To Device, Ignore Hash
- VC_OPERATION_TYPE_WRITE_FIX_HASH = Write File To Device, Fix Hash If Necessary

Parameters

<i>matchFW</i>	TRUE = Device FW must match file FW, FALSE = Don't validate device FW against file
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.
<i>isForeground</i>	If TRUE, will block program until update complete. Otherwise, FALSE performs update on background.

Returns

RETURN_CODE: Values can be parsed with `errorCode.getErrorString()`. When no data is available, return code = RETURN_CODE_NO_DATA_AVAILABLE

21.14.2.20 RETURN_CODE IDTechSDK.IDT_SpectrumPro.device_sendConfigurationFromZip (byte[] zip, string filename, VC_OPERATION_TYPE type, bool matchFW, string ident = " ", bool isForeground = false)

Send Configuration From Zip Executes a ViVOconfig update or verify to a device using files from a .zip archive

Once a ViVOconfig write or update starts, all messages will be returned to the callback as DeviceState.ViVOconfig. The ViVOconfig operation is complete when the return code from a DeviceState.ViVOconfig message is one of the following values:

- RETURN_CODE_SDK_VIVOCONFIG_VERIFY_SUCCESS = Verification process completed successfully. No differences found.
- RETURN_CODE_SDK_VIVOCONFIG_VERIFY_SUCCESS_WITH_WARNING = Verification process completed with warnings.
- RETURN_CODE_SDK_VIVOCONFIG_VERIFY_FAILED = Verification process FAILED
- RETURN_CODE_SDK_VIVOCONFIG_WRITE_SUCCESS = Write process completed successfully.
- RETURN_CODE_SDK_VIVOCONFIG_WRITE_SUCCESS_WITH_WARNING = Write process completed with warnings
- RETURN_CODE_SDK_VIVOCONFIG_WRITE_FAILED = Write process FAILED

Parameters

<i>zip</i>	.Zip archive containing the .json configuration file and also any associated image assets NOTE: You can also use this parameter to pass the path of where the .zip file is located at on local storage instead
<i>filename</i>	The name of the .json configuration file
<i>type</i>	VC_OPERATION_TYPE

- VC_OPERATION_TYPE_WRITE = Write File To Device, Hash must validate
- VC_OPERATION_TYPE_VERIFY = Verify Device With File
- VC_OPERATION_TYPE_WRITE_IGNORE_HASH = Write File To Device, Ignore Hash
- VC_OPERATION_TYPE_WRITE_FIX_HASH = Write File To Device, Fix Hash If Necessary

Parameters

<i>matchFW</i>	TRUE = Device FW must match file FW, FALSE = Don't validate device FW against file
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.
<i>isForeground</i>	If TRUE, will block program until update complete. Otherwise, FALSE performs update on background.

Returns

RETURN_CODE: Values can be parsed with `errorCode.getErrorString()`. When no data is available, return code = RETURN_CODE_NO_DATA_AVAILABLE

21.14.2.21 **RETURN_CODE** IDTechSDK.IDT_SpectrumPro.device_sendDataCommand (string *cmd*, bool *calcLRC*, ref byte[] *response*, string *ident* = " ")

Send a data command to the device

Sends a command to the device.

Parameters

<i>cmd</i>	String representation of command to execute
<i>calcLRC</i>	If TRUE, this will wrap command with start/length/lrc/sum/end: '{STX}{Len_Low}{Len_High} data {CheckLRC} {CheckSUM} {ETX}'
<i>response</i>	Response data
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString

21.14.2.22 **RETURN_CODE** IDTechSDK.IDT_SpectrumPro.device_sendDataCommand_ext (string *cmd*, bool *calcLRC*, ref byte[] *response*, int *timeout*, bool *noResponse*, string *ident* = " ")

Send a data command to the device - extended

Sends a command to the device.

Parameters

<i>cmd</i>	String representation of command to execute
<i>calcLRC</i>	If TRUE, this will wrap command as NGA with start/length/lrc/sum/end: '{STX}{Len_Low}{Len_High} data {CheckLRC} {CheckSUM} {ETX}'
<i>response</i>	Response data
<i>timeout</i>	Timeout value waiting for response, in milliseconds (1000 = 1 second, string ident = "")
<i>noResponse</i>	if TRUE, this will not wait for a response and immediately return SUCCESS
<i>calcITP</i>	If TRUE, this will wrap command as ITP with start/end/lrc: '{STX} data {ETX}{CheckLRC}'
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString

21.14.2.23 **RETURN_CODE** IDTechSDK.IDT_SpectrumPro.device_sendMacDataCommand (byte *taskId*, byte[] *functionID*, byte[] *data*, bool *macData*, ref byte[] *response*, string *ident* = " ")

Send a Command to device with MAC

Sends a command to the device with Mac calculation (if specified).

Parameters

<i>taskId</i>	Task ID to execute
<i>functionID</i>	Function ID to execute (one or more bytes, string ident = "")
<i>data</i>	Data to send (if any, string ident = "")

Parameters

<i>macData</i>	If TRUE, this will include the MAC with the command
<i>response</i>	Response data
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with `errorCode.getErrorString(string ident = "")`

21.14.2.24 RETURN_CODE IDTechSDK.IDT_SpectrumPro.device_sendPAE (string *command*, ref string *response*, int *timeout*, string *ident* = " ")

Send Payment Application Engine Command

Executes a PAE command

Parameters

<i>command</i>	ASCII command string, should start with "*PAE"
<i>response</i>	command response
<i>timeout</i>	timeout waiting for PAE response
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with `errorCode.getErrorString(string ident = "")`

21.14.2.25 RETURN_CODE IDTechSDK.IDT_SpectrumPro.device_setSpectrumProBDK (string *BDK*, string *ident* = " ")

Set Spectrum Pro BDK

Tells the SDK which BDK to use when generating MAC values for Spectrum Pro commands. SDK default value is "0123456789ABCDEFFEDCBA9876543210"

Parameters

<i>BDK</i>	Value of BDK, 32 ASCII characters representing 16 HEX values
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with `errorCode.getErrorString(string ident = "")`

21.14.2.26 RETURN_CODE IDTechSDK.IDT_SpectrumPro.device_setUID (string *UID*, int *keyType*, ref SpectrumInfo *info*, int *sessionTimeout* = 0x0064, string *ident* = " ")

Set Spectrum Pro UID

Sets the UID for the Spectrum Pro. SDK default value is "0001020304050607"

Parameters

<i>UID</i>	Value of UID, 16 ASCII characters representing 8 HEX values
<i>keyType</i>	Key type <ul style="list-style-type: none"> • 0 : Symmetric only • 1 : Asymmetric • 2 : Both Symmetric and Asymmetric
<i>info</i>	Spectrum Pro return info
<i>sessionTimeout</i>	Time, in seconds, that session data will be retained after data capture.
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with `errorCode.getErrorString(string ident = "")`

21.14.2.27 RETURN_CODE IDTechSDK.IDT_SpectrumPro.device_startRKI (bool *isTest*, string *ident* = " ")

Start Remote Key Injection

Starts a remote key injection request with IDTech RKI servers.

Parameters

<i>isTest</i>	TRUE = Demo Device, FALSE = Production Device
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with `device_getResponseCodeString`

21.14.2.28 RETURN_CODE IDTechSDK.IDT_SpectrumPro.device_StartRKI (int *type*, string *ident* = " ")

Start Remote Key Injection

Starts a remote key injection request with IDTech RKI servers. Set/Get RKI url with `IDT_Device.RKI_URL`.

Parameters

<i>type</i>	0 = Symmetric RKI Demo Unit 1 = Symmetric RKI Production Unit 2 = PKI-RKI Demo Unit 3 = PKI-RKI Production Unit
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString

21.14.2.29 RETURN_CODE IDTechSDK.IDT_SpectrumPro.device_startTransaction (double *amount*, double *amtOther*, int *exponent*, int *type*, int *timeout*, byte[] *tags*, bool *forceOnline*, bool *isFastEMV* = false, string *ident* = " ")

Start EMV Transaction Request

Authorizes the EMV transaction for an ICC card

The tags will be returned in the callback routine.

Parameters

<i>amount</i>	Transaction amount value (tag value 9F02, string ident = "")
<i>amtOther</i>	Other amount value, if any (tag value 9F03, string ident = "")
<i>exponent</i>	Number of characters after decimile point
<i>type</i>	Transaction type (tag value 9C).
<i>timeout</i>	Timeout value in seconds.
<i>tags</i>	Any other tags to be included in the request. Passed as TLV Data. Example, tag 9F02 with amount 0x000000000100 would be 0x9F0206000000000100 If tags 9F02 (amount), 9F03 (other amount), or 9C (transaction type) are included, they will take priority over these values supplied as individual parameters to this method.
<i>forceOnline</i>	TRUE = do not allow offline approval, FALSE = allow ICC to approve offline if terminal capable Note: To request tags to be included in default response, use tag DFEE1A, and specify tag list. Example four tags 9F02, 9F36, 95, 9F37 to be included in response = DFEE1A079F029F369F37
<i>isFastEMV</i>	If TRUE, it will populate the IDTTransactionData.fastEMV with ASCII data similar to IDTech FastEMV KB output, after performing an auto-authenticate and auto-complete with ResultCode = Could Not Contact Host
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString

21.14.2.30 RETURN_CODE IDTechSDK.IDT_SpectrumPro.device_updateFirmware (byte[] *firmwareData*, string *firmwareName*, int *encryptionType*, byte[] *keyBlob*, string *ident* = " ")

Update Firmware

Updates the firmware of the Spectrum Pro K21 HUB or Maxq1050.

Parameters

<i>firmwareData</i>	Signed binary data of a firmware file provided by IDTech
<i>firmwareName</i>	Firmware name. Must be one of the following two strings (with appropriate version information, string ident = "") <ul style="list-style-type: none"> • "SP K21 APP Vx.xx.xxx" • "SP MAX APP Vx.xx.xxx"

Parameters

<i>encryptionType</i>	Encryption type <ul style="list-style-type: none"> • 0 : Plaintext • 1 : TDES ECB, PKCS#5 padding • 2 : TDES CBC, PKCS#5, IV is all 0
<i>keyBlob</i>	Encrypted firmware session key blob, TR-31 Rev B, wrapped by FW Key (Optional, none if firmware is plaintext, string ident = "")
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with `errorCode.getErrorString(string ident = "")`

Firmware update status is returned in the callback with the following values: sender = SPECTRUM_PRO state = DeviceState.FirmwareUpdate data = File Progress. Two bytes, with `byte[0]` = current block, and `byte[1]` = total blocks. 0x0310 = block 3 of 16 transactionResultCode:

- RETURN_CODE_DO_SUCCESS = Firmware Update Completed Successfully
- RETURN_CODE_BLOCK_TRANSFER_SUCCESS = Current block transferred successfully
- Any other return code represents an error condition

21.14.2.31 RETURN_CODE IDTechSDK.IDT_SpectrumPro.emv_activateTransaction (int timeout, byte[] tags, bool forceOnline, bool isFastEMV = false, string ident = " ")

Start EMV Transaction Request

Authorizes the EMV transaction for an ICC card

The tags will be returned in the callback routine.

Parameters

<i>timeout</i>	Timeout value in seconds.
<i>tags</i>	The tags to be included in the request. Passed as TLV Data. Example, tag 9F02 with amount 0x000000000100 would be 0x9F0206000000000100
<i>forceOnline</i>	TRUE = do not allow offline approval, FALSE = allow ICC to approve offline if terminal capable Note: To request tags to be included in default response, use tag DFEE1A, and specify tag list. Example four tags 9F02, 9F36, 95, 9F37 to be included in response = DFEE1A079F029F369F37
<i>isFastEMV</i>	If TRUE, it will populate the IDTTransactionData.fastEMV with ASCII data similar to IDTech FastEMV KB output, after performing an auto-authenticate and auto-complete with ResultCode = Could Not Contact Host
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with `device_getResponseCodeString`

21.14.2.32 `RETURN_CODE IDTechSDK.IDT_SpectrumPro.emv_addTerminalData (byte[] tlv, string ident = " ")`

Add Terminal Data

Adds to the exosting Terminal Data

Parameters

<i>tlv</i>	TerminalData TLV Data
<i>ident</i>	Optional identifier

Return values

<i>RETURN_CODE</i>	Return codes listed as typedef enum in IDTCommon:RETURN_CODE. Values can be parsed with IDT_BTPay::device_getResponseCodeString:()
--------------------	--

21.14.2.33 `static void IDTechSDK.IDT_SpectrumPro.emv_allowFallback (bool allow, string ident = " ") [static]`

Allow fallback for EMV transactions. Default is TRUE

Parameters

<i>allow</i>	TRUE = allow fallback, FALSE = don't allow fallback
--------------	---

21.14.2.34 `RETURN_CODE IDTechSDK.IDT_SpectrumPro.emv_authenticateTransaction (byte[] updatedTLV, string ident = " ")`

Authenticate EMV Transaction Request

Authenticates the EMV transaction for an ICC card. Execute this after receiving response with result code 0x10 to `emv_startTransaction`

The tags will be returned in the callback routine.

Parameters

<i>updatedTLV</i>	TLV stream that can be used to update the following values: <ul style="list-style-type: none"> • 9F02: Amount • 9F03: Other amount • 9C: Transaction type • 5F57: Account type In addition tag DFEE1A can be sent to specify tag list to include in results. Example four tags 9F02, 9F36, 95, 9F37 to be included in response = DFEE1A079F029F369F9F37
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

`RETURN_CODE`: Values can be parsed with `device_getResponseCodeString`

21.14.2.35 static void IDTechSDK.IDT_SpectrumPro.emv_autoAuthenticate (bool *authenticate*, string *ident* = " ")
[static]

Enables authenticate for EMV transactions. If a startEMVTransaction results in code 0x0010 (start transaction success), then emv_authenticateEMVTransaction can automatically execute if parameter is set to TRUE

Parameters

<i>authenticate</i>	TRUE = auto authenticate, FALSE = manually authenticate
---------------------	---

21.14.2.36 static void IDTechSDK.IDT_SpectrumPro.emv_autoAuthenticateTags (bool *authenticate*, byte[] *tags*, string *ident* = " ") [static]

Enables authenticate for EMV transactions. If a startEMVTransaction results in code 0x0010 (start transaction success), then emv_authenticateEMVTransaction can automatically execute if parameter is set to TRUE

Parameters

<i>authenticate</i>	TRUE = auto authenticate, FALSE = manually authenticate
<i>tags</i>	Tags to pass during authentication stage;

21.14.2.37 RETURN_CODE IDTechSDK.IDT_SpectrumPro.emv_callbackResponseLCD (EMV_LCD_DISPLAY_MODE *type*, byte *selection*, string *ident* = " ")

Callback Response LCD Display

Provides menu selection responses to the kernel after a callback was received with DeviceState.EMVCallback, and callbackType = EMV_CALLBACK_TYPE.EMV_CALLBACK_TYPE_LCD, and lcd_displayMode = EMV_LCD_DISPLAY_MODE_MENU, EMV_LCD_DISPLAY_MODE_PROMPT, or EMV_LCD_DISPLAY_MODE_LANGUAGE_SELECT

Parameters

<i>type</i>	If Cancel key pressed during menu selection, then value is EMV_LCD_DISPLAY_MODE_CANCEL. Otherwise, value can be EMV_LCD_DISPLAY_MODE_MENU, EMV_LCD_DISPLAY_MODE_PROMPT, or EMV_LCD_DISPLAY_MODE_LANGUAGE_SELECT
<i>selection</i>	If type = EMV_LCD_DISPLAY_MODE_MENU or EMV_LCD_DISPLAY_MODE_LANGUAGE_SELECT, provide the selection ID line number. Otherwise, if type = EMV_LCD_DISPLAY_MODE_PROMPT supply either 0x43 ('C') for Cancel, or 0x45 ('E') for Enter/accept
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with errorCode.getErrorString()

21.14.2.38 RETURN_CODE IDTechSDK.IDT_SpectrumPro.emv_callbackResponseMSR (byte[] *MSR*, string *ident* = " ")

Callback Response MSR Entry

Provides MSR information to kernel after a callback was received with DeviceState.EMVCallback, and callbackType = EMV_CALLBACK_MSR

Parameters

<i>MSR</i>	Swiped track data
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with `errorCode.getErrorString()`

21.14.2.39 RETURN_CODE IDTechSDK.IDT_SpectrumPro.emv_callbackResponsePIN (EMV_PIN_MODE *type*, byte[] *KSN*, byte[] *PIN*, string *ident* = " ")

Callback Response PIN Entry

Provides (or cancels) PIN entry information to kernel after a callback was received with DeviceState.EMVCallback, and callbackType = EMV_CALLBACK_TYPE_PINPAD

Parameters

<i>type</i>	If Cancel key pressed during PIN entry, then value is EMV_PIN_MODE_CANCEL. Otherwise, value can be EMV_PIN_MODE_ONLINE_DUKPT, EMV_PIN_MODE_ONLINE_MKSK, or EMV_PIN_MODE_OFFLINE
<i>KSN</i>	If enciphered PIN, this is either the PIN DUKPT Key (EMV_PIN_MODE_ONLINE_DUKPT) or PIN Session Key (EMV_PIN_MODE_ONLINE_MKSK), or PIN Pairing DUKPT key (EMV_PIN_MODE_OFFLINE, string <i>ident</i> = "")
<i>PIN</i>	If enciphered PIN, this is encrypted PIN block. If device does not implement pairing function, this is plaintext PIN
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with `device_getResponseCodeString`

21.14.2.40 RETURN_CODE IDTechSDK.IDT_SpectrumPro.emv_cancelTransaction (string *ident* = " ")

Cancel EMV Transaction

Cancels the currently executing EMV transaction.

Parameters

<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.
--------------	--

Returns

RETURN_CODE: Values can be parsed with `errorCode.getErrorString()`

21.14.2.41 RETURN_CODE IDTechSDK.IDT_SpectrumPro.emv_clearTransactionLog (string *ident* = " ")

Clear Transaction Log

Clears the transaction log.

Parameters

<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.
--------------	--

Returns

RETURN_CODE: Values can be parsed with `errorCode.getErrorString()`

21.14.2.42 RETURN_CODE IDTechSDK.IDT_SpectrumPro.emv_completeTransaction (bool *commError*, byte[] *authCode*, byte[] *iad*, byte[] *tlvScripts*, byte[] *tlv*, string *ident* = " ")

Complete EMV Transaction Request

Completes the EMV transaction for an ICC card when online authorization request is received from `emv_authenticateTransaction`

The tags will be returned in the callback routine.

Parameters

<i>commError</i>	Communication error with host. Set to TRUE if host was unreachable, or FALSE if host response received. If Communication error, <i>authCode</i> , <i>iad</i> , <i>tlvScripts</i> can be null.
<i>authCode</i>	Authorization code from host. Two bytes. Example 0x3030. (Tag value 8A). Required
<i>iad</i>	Issuer Authentication Data, if any. Example 0x11223344556677883030 (tag value 91).
<i>tlvScripts</i>	71/72 scripts, if any
<i>tlv</i>	Additional TVL data to return with transaction results (if any, string <i>ident</i> = "")
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with `device_getResponseCodeString`

NOTE: There are three possible outcomes for Authorization Code: Approval, Refer To Bank, Decline. The kernel maps these three outcomes to valid authorization response codes using tag DFEE1B through 8 bytes: {2 bytes Approval Code}{2 bytes Referral Code}{2 bytes Decline Code}{2 bytes RFU} If your gateway uses "00" for Approval, "01" for Referral, and "05" for Decline, then DFEE1B 08 3030 3031 3035 0000 If you use an authorization code value that is not defined in DFEE1B, the kernel will use the DECLINE value of DFEE1B by default.

21.14.2.43 RETURN_CODE IDTechSDK.IDT_SpectrumPro.emv_getEMVConfigurationCheckValue (ref string *response*, string *ident* = " ")

Get EMV Kernel configuration check value info

Parameters

<i>response</i>	Response returned of Kernel configuration check value info
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with `device_getResponseCodeString`

21.14.2.44 **RETURN_CODE** IDTechSDK.IDT_SpectrumPro.emv_getEMVKernelCheckValue (ref string *response*, string *ident* = " ")

Get EMV Kernel check value info

Parameters

<i>response</i>	Response returned of Kernel check value info
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString

21.14.2.45 **RETURN_CODE** IDTechSDK.IDT_SpectrumPro.emv_getEMVKernelVersion (ref string *response*, string *ident* = " ")

Polls device for EMV Kernel Version

Parameters

<i>response</i>	Response returned of Kernel Version
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with errorCode.getErrorString(string ident = "")

21.14.2.46 **RETURN_CODE** IDTechSDK.IDT_SpectrumPro.emv_getTerminalID (ref string *response*, string *ident* = " ")

Gets the terminal ID as printable characters .

Parameters

<i>response</i>	Terminal ID
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with errorCode.getErrorString(string ident = "")

21.14.2.47 **RETURN_CODE** IDTechSDK.IDT_SpectrumPro.emv_getTerminalMajorConfiguration (ref int *configuration*, string *ident* = " ")

Gets the terminal major configuration in ICS .

Parameters

<i>configuration</i>	A configuration value, range 1-4 <ul style="list-style-type: none"> • 1 = 1C • 2 = 2C • 3 = 3C • 4 = 4C • 5 = 5C
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with `errorCode.getErrorString(string ident = "")`

21.14.2.48 RETURN_CODE IDTechSDK.IDT_SpectrumPro.emv_removeAllApplicationData (string *ident* = " ")

Remove All Application Data

Removes all the Application Data

Parameters

<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.
--------------	--

Returns

RETURN_CODE: Values can be parsed with `errorCode.getErrorString(string ident = "")`

21.14.2.49 RETURN_CODE IDTechSDK.IDT_SpectrumPro.emv_removeAllCAPK (string *ident* = " ")

Remove All Certificate Authority Public Key

Removes all the CAPK

Parameters

<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.
--------------	--

Returns

RETURN_CODE: Values can be parsed with `errorCode.getErrorString(string ident = "")`

21.14.2.50 RETURN_CODE IDTechSDK.IDT_SpectrumPro.emv_removeAllCRL (string *ident* = " ")

Remove All Certificate Revocation List Entries

Removes all CRLentry entries

Parameters

<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.
--------------	--

Returns

RETURN_CODE: Values can be parsed with `errorCode.getErrorString(string ident = "")`

21.14.2.51 RETURN_CODE IDTechSDK.IDT_SpectrumPro.emv_removeApplicationData (byte[] AID, string ident = " ")

Remove Application Data by AID

Removes the Application Data as specified by the AID name passed as a parameter

Parameters

<i>AID</i>	Name of ApplicationID Must be between 5 and 16 bytes
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with `errorCode.getErrorString(string ident = "")`

21.14.2.52 RETURN_CODE IDTechSDK.IDT_SpectrumPro.emv_removeCAPK (byte[] capk, string ident = " ")

Remove Certificate Authority Public Key

Removes the CAPK as specified by the RID/Index

Parameters

<i>capk</i>	6 byte CAPK = 5 bytes RID + 1 byte INDEX
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with `errorCode.getErrorString(string ident = "")`

21.14.2.53 RETURN_CODE IDTechSDK.IDT_SpectrumPro.emv_removeCRL (byte[] crlList, string ident = " ")

Remove Certificate Revocation List Entries

Removes CRLEntries as specified by the RID and Index and serial number passed as 9 bytes

Parameters

<i>crlList</i>	containing the list of CRL to remove: [CRL1][CRL2]...[CRLn] where each [CRL] is 9 bytes: [5 bytes RID][1 byte CAPK Index][3 bytes serial number]
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString

21.14.2.54 RETURN_CODE IDTechSDK.IDT_SpectrumPro.emv_removeTerminalData (string *ident* = " ")

Remove Terminal Data

Removes the Terminal Data

Parameters

<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.
--------------	--

Returns

RETURN_CODE: Values can be parsed with errorCode.getErrorString(string ident = "")

21.14.2.55 RETURN_CODE IDTechSDK.IDT_SpectrumPro.emv_retrieveAIDList (ref byte *response*[[], string *ident* = " ")

Retrieve AID list

Returns all the AID names installed on the terminal.

Parameters

<i>response</i>	array of AID name byte arrays
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with errorCode.getErrorString(string ident = "")

21.14.2.56 RETURN_CODE IDTechSDK.IDT_SpectrumPro.emv_retrieveApplicationData (byte[] *AID*, ref byte[] *tlv*, string *ident* = " ")

Retrieve Application Data by AID

Retrieves the Application Data as specified by the AID name passed as a parameter.

Parameters

<i>AID</i>	Name of ApplicationID. Must be between 5 and 16 bytes
<i>tlv</i>	The TLV elements of the requested AID
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with errorCode.getErrorString()

21.14.2.57 RETURN_CODE IDTechSDK.IDT_SpectrumPro.emv_retrieveCAPK (byte[] *capk*, ref byte[] *key*, string *ident* = " ")

Retrieve Certificate Authority Public Key

Retrieves the CAPK as specified by the RID/Index passed as a parameter.

Parameters

<i>capk</i>	6 bytes CAPK = 5 bytes RID + 1 byte Index
<i>key</i>	Response returned as a CAKey format: [5 bytes RID][1 byte Index][1 byte Hash Algorithm][1 byte Encryption Algorithm][20 bytes HashValue][4 bytes Public Key Exponent][2 bytes Modulus Length][Variable bytes Modulus] Where: <ul style="list-style-type: none"> • Hash Algorithm: The only algorithm supported is SHA-1. The value is set to 0x01 • Encryption Algorithm: The encryption algorithm in which this key is used. Currently support only one type: RSA. The value is set to 0x01. • HashValue: Which is calculated using SHA-1 over the following fields: RID & Index & Modulus & Exponent • Public Key Exponent: Actually, the real length of the exponent is either one byte or 3 bytes. It can have two values: 3 (Format is 0x00 00 00 03), or 65537 (Format is 0x00 01 00 01, string ident = "") • Modulus Length: LenL LenH Indicated the length of the next field. • Modulus: This is the modulus field of the public key. Its length is specified in the field above.
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with `errorCode.getErrorString()`

21.14.2.58 RETURN_CODE IDTechSDK.IDT_SpectrumPro.emv_retrieveCAPKList (ref byte[] keys, string ident = " ")

Retrieve the Certificate Authority Public Key list

Returns all the CAPK RID and Index installed on the terminal.

Parameters

<i>keys</i>	[key1][key2]...[keyn], each key 6 bytes where key = 5 bytes RID + 1 byte index
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with `errorCode.getErrorString()`

21.14.2.59 RETURN_CODE IDTechSDK.IDT_SpectrumPro.emv_retrieveCRLList (ref byte[] list, string ident = " ")

Retrieve the Certificate Revocation List

Returns the CRL entries on the terminal.

Parameters

<i>list</i>	[CRL1][CRL2]...[CRLn], each CRL 9 bytes where CRL = 5 bytes RID + 1 byte index + 3 bytes serial number
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with `errorCode.getErrorString()`

21.14.2.60 RETURN_CODE IDTechSDK.IDT_SpectrumPro.emv_retrieveTerminalData (ref byte[] *tlv*, string *ident* = " ")

Retrieve Terminal Data

Retrieves the Terminal Data.

Parameters

<i>tlv</i>	Response returned as a TLV
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with `errorCode.getErrorString()`

21.14.2.61 RETURN_CODE IDTechSDK.IDT_SpectrumPro.emv_retrieveTransactionResult (byte[] *tags*, ref IDTTransactionData *tlv*, string *ident* = " ")

Retrieve Transaction Results

Retrieves specified EMV tags from the currently executing transaction.

Parameters

<i>tags</i>	Tags to be retrieved. Example 0x9F028A will retrieve tags 9F02 and 8A
<i>tlv</i>	All requested tags returned as unencrypted, encrypted and masked TLV data in IDTTransactionData object
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with `device.getResponseCodeString`

21.14.2.62 RETURN_CODE IDTechSDK.IDT_SpectrumPro.emv_setApplicationData (byte[] *name*, byte[] *tlv*, string *ident* = " ")

Set Application Data by AID

Sets the Application Data as specified by the application name and TLV data

Parameters

<i>name</i>	Application name, 10-32 ASCII hex characters representing 5-16 bytes Example "a0000000031010"
<i>tlv</i>	Application data in TLV format
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with `errorCode.getErrorString()`

21.14.2.63 RETURN_CODE IDTechSDK.IDT_SpectrumPro.emv_setCAPK (byte[] key, string ident = " ")

Set Certificate Authority Public Key

Sets the CAPK as specified by the CAKey structure

Parameters

<i>key</i>	CAKey format: [5 bytes RID][1 byte Index][1 byte Hash Algorithm][1 byte Encryption Algorithm][20 bytes HashValue][4 bytes Public Key Exponent][2 bytes Modulus Length][Variable bytes Modulus] Where: <ul style="list-style-type: none"> • Hash Algorithm: The only algorithm supported is SHA-1. The value is set to 0x01 • Encryption Algorithm: The encryption algorithm in which this key is used. Currently support only one type: RSA. The value is set to 0x01. • HashValue: Which is calculated using SHA-1 over the following fields: RID & Index & Modulus & Exponent • Public Key Exponent: Actually, the real length of the exponent is either one byte or 3 bytes. It can have two values: 3 (Format is 0x00 00 00 03), or 65537 (Format is 0x00 01 00 01, string ident = "") • Modulus Length: LenL LenH Indicated the length of the next field. • Modulus: This is the modulus field of the public key. Its length is specified in the field above.
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with errorCode.getErrorString()

21.14.2.64 RETURN_CODE IDTechSDK.IDT_SpectrumPro.emv_setCRL (byte[] list, string ident = " ")

Set Certificate Revocation List

Sets the CRL

Parameters

<i>list</i>	CRL Entries containing the RID, Index, and serial numbers to set [CRL1][CRL2]...[CRLn] where each [CRL] is 9 bytes: [5 bytes RID][1 byte CAPK Index][3 bytes serial number]
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with errorCode.getErrorString()

21.14.2.65 RETURN_CODE IDTechSDK.IDT_SpectrumPro.emv_setTerminalData (byte[] tlv, string ident = " ")

Set Terminal Data

Sets the Terminal Data

Parameters

<i>tlv</i>	TerminalData TLV Data
<i>ident</i>	Optional identifier

Return values

<i>RETURN_CODE</i>	Return codes listed as typedef enum in IDTCommon:RETURN_CODE. Values can be parsed with IDT_BTPay::device_getResponseCodeString:()
--------------------	--

21.14.2.66 **RETURN_CODE** IDTechSDK.IDT_SpectrumPro.emv_setTerminalID (string *terminalID*, string *ident* = " ")

Sets the terminal ID as printable characters .

Parameters

<i>terminalID</i>	Terminal ID to set
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with errorCode.getErrorString(string ident = "")

21.14.2.67 **RETURN_CODE** IDTechSDK.IDT_SpectrumPro.emv_setTerminalMajorConfiguration (int *configuration*, string *ident* = " ")

Sets the terminal major configuration in ICS .

Parameters

<i>configuration</i>	A configuration value, range 1-4 <ul style="list-style-type: none"> • 1 = 1C • 2 = 2C • 3 = 3C • 4 = 4C • 5 = 5C
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with errorCode.getErrorString(string ident = "")

21.14.2.68 **RETURN_CODE** IDTechSDK.IDT_SpectrumPro.emv_startTransaction (double *amount*, double *amtOther*, int *exponent*, int *type*, int *timeout*, byte[] *tags*, bool *forceOnline*, bool *isFastEMV* = false, string *ident* = " ")

Start EMV Transaction Request

Authorizes the EMV transaction for an ICC card

The tags will be returned in the callback routine.

Parameters

<i>amount</i>	Transaction amount value (tag value 9F02, string ident = "")
---------------	---

Parameters

<i>amtOther</i>	Other amount value, if any (tag value 9F03, string ident = "")
<i>exponent</i>	Number of characters after decimile point
<i>type</i>	Transaction type (tag value 9C).
<i>timeout</i>	Timeout value in seconds.
<i>tags</i>	Any other tags to be included in the request. Passed as TLV Data. Example, tag 9F02 with amount 0x000000000100 would be 0x9F0206000000000100 If tags 9F02 (amount), 9F03 (other amount), or 9C (transaction type) are included, they will take priority over these values supplied as individual parameters to this method.
<i>forceOnline</i>	TRUE = do not allow offline approval, FALSE = allow ICC to approve offline if terminal capable Note: To request tags to be included in default response, use tag DFEE1A, and specify tag list. Example four tags 9F02, 9F36, 95, 9F37 to be included in response = DFEE1A079F029F369F37
<i>isFastEMV</i>	If TRUE, it will populate the IDTTransactionData.fastEMV with ASCII data similar to IDTech FastEMV KB output, after performing an auto-authenticate and auto-complete with ResultCode = Could Not Contact Host
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString

21.14.2.69 RETURN_CODE IDTechSDK.IDT_SpectrumPro.emv_trySetTerminalData (byte[] *tlv*, ref byte[] *rejectedTLV*, ref byte[] *convertedTLV*, bool *overwrite* = false, string *ident* = " ")

Try to Set Terminal Data

Attempts to set the Terminal Data. Use this functions to attempt to set the terminal data to the device. This function will allow the cross-device tag compatible writing of IDTech Legacy/New tag definitions

Parameters

<i>tlv</i>	TerminalData TLV Data
<i>rejectedTLV</i>	Contains the tags (if any) that were rejected by the firmware
<i>convertedTLV</i>	Contains the tags (if any) that were converted and accepted by the firmware
<i>overwrite</i>	TRUE = add TLV to existing tags, FALSE = replace existing tags with TLV
<i>ident</i>	Optional identifier

Return values

<i>RETURN_CODE</i>	Return codes listed as typedef enum in IDTCommon:RETURN_CODE. Values can be parsed with IDT_BTPay::device_getResponseCodeString:()
--------------------	--

21.14.2.70 static int IDTechSDK.IDT_SpectrumPro.getCommandTimeout (string *ident* = " ") [static]

Get Command Timeout

Gets the default timeout (in milliseconds) waiting for a blocking command response

Return values**Return values**

<i>time</i>	Time
-------------	------

21.14.2.71 RETURN_CODE IDTechSDK.IDT_SpectrumPro.icc_getICCReaderStatus (ref byte *status*, string *ident* = " ")

Get Reader Status

Returns the reader status

Parameters

<i>status</i>	Pointer that will return with the ICCReaderStatus results. bit 0: 0 = ICC Power Not Ready, 1 = ICC Powered bit 1: 0 = Card not seated, 1 = card seated
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString

21.14.2.72 RETURN_CODE IDTechSDK.IDT_SpectrumPro.icc_getICCStatus (ref byte[] *status*, ref byte[] *atr*, string *ident* = " ")

Get Reader Status

Returns the ICC ATR Data and the status of the Smart Card interface: Smart Card voltage, protocols and parameters

Parameters

<i>status</i>	<p>Pointer that will return with the ICC info. byte 0: Card Presence</p> <ul style="list-style-type: none"> • Bit 0: Card Inserted • Bit 1: Error communicating with ICC • Bit 2: Error activating/resetting ICC • Bit 5: Latch active • Bit 3,4,6,7: RFU 0 byte 1: Current setting and voltage of ICC <ul style="list-style-type: none"> – 0 = Card is not powered up – 1 = Card is powered with 1.8 V – 2 = Card is powered with 3 V – 3 = Card is powered with 5 V <p>byte 2: Protocol</p> <ul style="list-style-type: none"> – 0 = (T = 0, string ident = "") – 1 = (T = 1, string ident = "") bytes 3-5: T=0 Protocol parameters (if T=1, bytes 3-5 all 00, string ident = "") – Byte 3 = TA1 (FI/DI, string ident = "") – Byte 4 = TC1 (Guard Time, string ident = "") – Byte 5 = WI bytes 6-9: T=1 Protocol parameters (if T=0, bytes 6-9 all 00, string ident = "") – Byte 6 = TA1 (FI/DI, string ident = "") – Byte 7 = TC1 (EGT, string ident = "") – Byte 8 = TB3 (BWI/CWI, string ident = "") – Byte 9 = IFSC
<i>atr</i>	ATR Data is included when card is inserted and no other errors returned.
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString

21.14.2.73 RETURN_CODE IDTechSDK.IDT_SpectrumPro.icc_powerOffICC (string ident = " ")

Power Off ICC

Powers down the ICC

Parameters

<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.
--------------	--

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString

If Success, empty If Failure, ASCII encoded data of error string

21.14.2.74 **RETURN_CODE** IDTechSDK.IDT_SpectrumPro.icc_powerOnICC (ref byte[] *ATR*, string *ident* = " ")

Power On ICC

Power up the currently selected microprocessor card in the ICC reader

Parameters

<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.
--------------	--

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString

21.14.2.75 static void IDTechSDK.IDT_SpectrumPro.lcd_retrieveMessage (DisplayMessages.DISPLAY_MESSAGE_LANGUAGE *lang*, DisplayMessages.DISPLAY_MESSAGE_IDENTIFIER *id*, ref string *line1*, ref string *line2*) [static]

Retrieve LCD Message

Returns the string value for a message ID returned for LCD messaging

Parameters

<i>lang</i>	Language.
<i>id</i>	Message ID
<i>line1</i>	Line 1 string value
<i>line2</i>	Line 2 string value
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString

21.14.2.76 **RETURN_CODE** IDTechSDK.IDT_SpectrumPro.msr_cancelMSRSwipe (string *ident* = " ")

Disable MSR Swipe Cancels MSR swipe request.

Parameters

<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.
--------------	--

Returns

RETURN_CODE: Values can be parsed with errorCode.getErrorString(string ident = "")

21.14.2.77 **RETURN_CODE** IDTechSDK.IDT_SpectrumPro.msr_clearMSRData (string *ident* = " ")

Clear MSR Data

Clears the MSR Data buffer

Parameters

<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.
--------------	--

Returns

RETURN_CODE: Values can be parsed with `errorCode.getErrorString(string ident = "")`

21.14.2.78 RETURN_CODE IDTechSDK.IDT_SpectrumPro.msr_getMSRData (ref IDTTransactionData *card*, string *ident* = " ")

Get MSR Data

Reads the MSR Data buffer

@param *card* Card data

Parameters

<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.
--------------	--

Returns

RETURN_CODE: Values can be parsed with `errorCode.getErrorString(string ident = "")`

21.14.2.79 RETURN_CODE IDTechSDK.IDT_SpectrumPro.msr_startMSRSwipe (int *timeout*, string *ident* = " ")

Enable MSR Swipe

Enables MSR, waiting for swipe to occur. Allows track selection. Returns IDTMSRData instance to `deviceDelegate::swipeMSRData:(string ident = "")`

Parameters

<i>timeout</i>	Swipe Timeout Value
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with `errorCode.getErrorString(string ident = "")`

21.14.2.80 RETURN_CODE IDTechSDK.IDT_SpectrumPro.pin_cancelPINEntry (string *ident* = " ")

Cancel PIN Entry

Cancels PIN entry request

Parameters

<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.
--------------	--

Returns

RETURN_CODE: Values can be parsed with `errorCode.getErrorString(string ident = "")`

21.14.2.81 RETURN_CODE IDTechSDK.IDT_SpectrumPro.pin_getPIN (int *mode*, int *PANSource*, string *iccPAN*, int *startTimeout*, int *entryTimeout*, string *language*, string *ident* = " ")

Get Encrypted PIN

Requests PIN Entry

Parameters

<i>mode</i>	<ul style="list-style-type: none"> • 0x00- Cancel: Cancels PIN entry = also can execute pin_cancelPINEntry(). All other parameters for this method will be ignored • 0x01- Online PIN DUKPT • 0x02- Online PIN MKSK • 0x03- Offline PIN (No need to define PAN Source or ICC PAN, string ident = "")
<i>PANSource</i>	<ul style="list-style-type: none"> • 0x00- ICC: PAN Captured from ICC and must be provided in <i>iccPAN</i> parameter • 0x01- MSR: PAN Captured from MSR swipe and will be inserted by Spectrum Pro. No need to provide <i>iccPAN</i> parameter.
<i>iccPAN</i>	PAN captured from ICC. When PAN captured from MSR, this parameter will be ignored
<i>startTimeout</i>	The amount of time allowed to start PIN entry before timeout
<i>entryTimeout</i>	The amount of time to enter the PIN after first digit selected before timeout
<i>language</i>	Valid values "EN" for English, "ES" for Spanish, "ZH" for Chinese, "FR" for French
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with `errorCode.getErrorString(string ident = "")`

21.14.2.82 RETURN_CODE IDTechSDK.IDT_SpectrumPro.pin_passThroughMode (bool *enable*, string *ident* = " ")

Pin Pass Through

Enables all commands to be passed through the PINpad

Parameters

<i>enable</i>	TRUE = all commands passed through, FALSE = commands not passed through
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with `errorCode.getErrorString(string ident = "")`

21.14.2.83 `static String IDTechSDK.IDT_SpectrumPro.SDK_Version () [static]`

SDK Version

- All Devices

Returns the current version of SDK

Returns

Framework version

21.14.2.84 `static void IDTechSDK.IDT_SpectrumPro.setCallback (CallBack my_Callback) [static]`

Set Callback

Sets the class callback

21.14.2.85 `static void IDTechSDK.IDT_SpectrumPro.setCallback (IntPtr my_Callback, SynchronizationContext context) [static]`

Set Callback

Sets the class callback

Parameters

<i>my_Callback</i>	The callback function to receive the response message from device. defined as follows. public unsafe delegate void MFCCallBack(Parameters parameters, ident);
<i>context</i>	The context of the UI thread

21.14.2.86 `static void IDTechSDK.IDT_SpectrumPro.setCardNotificationCallback (CardNotificationCallback my_Callback) [static]`

Set Card Notification Callback

Sets the callback for card notification insert/remove/data

Notification returned to CardNotificationCallback, use [setCardNotificationCallback\(CardNotificationCallback callback\)](#). CardNotificationCallback (byte status, string ident = "") 0x01 = bad msr 0x02 = good msr 0x04 = front switch 0x08 = seat switch

enable with device_setCardNotification

21.14.2.87 `static void IDTechSDK.IDT_SpectrumPro.setCommandTimeout (int milliseconds, string ident = " ") [static]`

Set Command Timeout

Sets the default timeout (in milliseconds) waiting for a blocking command response

Parameters

<i>milliseconds</i>	Time
---------------------	------

21.14.2.88 `static bool IDTechSDK.IDT_SpectrumPro.useSerialPort (int port) [static]`

Use Serial Port Interface

Instructs SDK to attempt to use the Serial Port for communication with Spectrum Pro using default baud rate

Parameters

<i>port</i>	Serial Port to use. Example COM1 = 1.
-------------	---------------------------------------

Returns

bool TRUE=successful, FALSE=failure

21.14.2.89 `static bool IDTechSDK.IDT_SpectrumPro.useSerialPort (int port, int baud) [static]`

Use Serial Port Interface with baud rate

Instructs SDK to attempt to use the Serial Port for communication with Spectrum Pro

Parameters

<i>port</i>	Serial Port to use. Example COM1 = 1.
<i>baud</i>	Baud rate to override default. Example 115200;

Returns

bool TRUE=successful, FALSE=failure

21.14.2.90 `static bool IDTechSDK.IDT_SpectrumPro.useSerialPortLinux (string path) [static]`

Use Serial Port Interface on Linux

Instructs SDK to attempt to use the Serial Port for communication with BTMag using default baud rate on Linux implementations

Parameters

<i>path</i>	Path to use. Example /dev/ttyUSB*
-------------	-----------------------------------

Returns

bool TRUE=successful, FALSE=failure

21.14.2.91 `static bool IDTechSDK.IDT_SpectrumPro.useSerialPortLinux (string path, int baud) [static]`

Use Serial Port Interface on Linux with baud rate

Instructs SDK to attempt to use the Serial Port for communication with BTPay on Linux implementations

Parameters

<i>path</i>	Path to use. Example /dev/ttyUSB*
<i>baud</i>	Baud rate to override default. Example 115200;

Returns

bool TRUE=successful, FALSE=failure

21.14.3 Property Documentation**21.14.3.1 IDT_SpectrumPro** IDTechSDK.IDT_SpectrumPro.SharedController [static], [get]**Singleton Instance**

Establishes an singleton instance of Spectrum Pro class.

Returns

Instance of Spectrum Pro

The documentation for this class was generated from the following file:

- /Users/randy/Repo/universal-sdk/IDTechSDK_STD/DeviceHeaders/IDT_SpectrumPro.cs

21.15 IDTechSDK.IDT_SREDKey2 Class Reference**Public Member Functions**

- RETURN_CODE [msr_resetTerminalData](#) (string ident="")
- RETURN_CODE [msr_cancelTransaction](#) (string ident="")
- RETURN_CODE [msr_setConfiguration](#) (byte[] config, string ident="")
- RETURN_CODE [msr_getConfiguration](#) (ref byte[] config, string ident="")
- RETURN_CODE [msr_setTerminalData](#) (byte[] tlv, string ident="")
- RETURN_CODE [msr_getTerminalData](#) (byte[] tag, ref byte[] tlv, string ident="")
- RETURN_CODE [msr_startKeyedTransaction](#) (int timeout, bool ZIP, bool CSC, bool ADR, bool Mod10Check, string ident="")
- RETURN_CODE [msr_startSwipeTransaction](#) (int timeout, byte[] tlv, string ident="")
- RETURN_CODE [device_setTransArmorEncryption](#) (byte[] cert, string ident="")
- RETURN_CODE [device_setPollMode](#) (byte mode, string ident="")
- RETURN_CODE [device_getOutputType](#) (bool isSwipe, ref int response, string ident="")
- RETURN_CODE [device_setOutputType](#) (bool isSwipe, int value, string ident="")
- RETURN_CODE [config_getModelNumber](#) (ref string response, string ident="")
- RETURN_CODE [device_updateDeviceFromManifest](#) (string filepath, string ident="", bool isForeground=false)
- RETURN_CODE [device_certificateType](#) (ref int type, string ident="")
- RETURN_CODE [device_sendVivoCommandP3](#) (byte command, byte subCommand, byte[] data, ref byte[] response, string ident="")
- RETURN_CODE [device_sendVivoCommandP3_ext](#) (byte command, byte subCommand, byte[] data, ref byte[] response, int timeout, bool noResponse, string ident="")
- RETURN_CODE [device_sendVivoCommandP2](#) (byte command, byte subCommand, byte[] data, ref byte[] response, string ident="")
- RETURN_CODE [device_sendVivoCommandP2_ext](#) (byte command, byte subCommand, byte[] data, ref byte[] response, int timeout, bool noResponse, string ident="")
- RETURN_CODE [config_getSerialNumber](#) (ref string response, string ident="")
- RETURN_CODE [device_getFirmwareVersion](#) (ref string response, string ident="")
- string [device_getResponseCodeString](#) (RETURN_CODE eCode)
- RETURN_CODE [device_sendDataCommand](#) (string cmd, bool calcLRC, ref byte[] response, string ident="")
- RETURN_CODE [device_sendDataCommand_ext](#) (string cmd, bool calcLRC, ref byte[] response, int timeout, bool noResponse, string ident="")
- RETURN_CODE [msr_disable](#) (string ident="")

- RETURN_CODE [msr_enable](#) (string ident="")
- IDTechSDK.RETURN_CODE [device_setLanguage](#) (int val, string ident="")
- RETURN_CODE [msr_getSwipeEncryption](#) (ref byte encryption, string ident="")
- RETURN_CODE [device_getTransArmorID](#) (ref string TID, string ident="")
- RETURN_CODE [device_setTransArmorID](#) (string TID, string ident="")
- RETURN_CODE [device_enableAdminKey](#) (bool enable, string ident="")
- RETURN_CODE [msr_setExpirationMask](#) (bool mask, string ident="")
- RETURN_CODE [msr_getExpirationMask](#) (ref byte value, string ident="")
- RETURN_CODE [msr_setClearPANID](#) (byte val, string ident="")
- RETURN_CODE [msr_getClearPANID](#) (ref byte value, string ident="")
- RETURN_CODE [msr_getSwipeMaskOption](#) (ref byte option, string ident="")
- RETURN_CODE [msr_setSwipeMaskOption](#) (bool track1, bool track2, bool track3, string ident="")
- RETURN_CODE [msr_setSwipeForcedEncryptionOption](#) (bool track1, bool track2, bool track3, bool track3card0, string ident="")
- RETURN_CODE [msr_getSwipeForcedEncryptionOption](#) (ref byte option, string ident="")
- RETURN_CODE [msr_getFunctionStatus](#) (ref bool enabled, ref bool isBufferMode, ref bool withNotification, string ident="")
- RETURN_CODE [device_rebootDevice](#) (string ident="")
- RETURN_CODE [device_sendVivoCommandP4](#) (byte command, byte subCommand, byte[] data, ref byte[] response, string ident="")
- RETURN_CODE [device_sendVivoCommandP4_ext](#) (byte command, byte subCommand, byte[] data, ref byte[] response, int timeout, bool noResponse, string ident="")
- RETURN_CODE [config_getStatusKeySlots](#) (ref byte[] keyslots, string ident="")
- RETURN_CODE [device_updateDeviceFirmware](#) (byte[] firmwareData, string ident="")
- RETURN_CODE [msr_switchUSBInterfaceMode](#) (bool blsUSBKeyboardMode)
- RETURN_CODE [msr_switchUSBInterfaceMode](#) (bool blsUSBKeyboardMode, ref string ident)
- RETURN_CODE [device_startRKI](#) (string ident="")
- RETURN_CODE [device_getRKIStatus](#) (bool isProd, bool isMultiKey, ref string status, ref Dictionary< string, RKI_KEY_TYPE > keys, string ident="")
- RETURN_CODE [device_getAnyRKIStatus](#) (bool isMultiKey, ref string status, ref Dictionary< string, RKI_KEY_TYPE > keys, string ident="")
- RETURN_CODE [device_updateFirmwareFromZip](#) (byte[] zipfile, string ident="", bool isForeground=false)
- RETURN_CODE [device_sendConfigurationFromZip](#) (byte[] zip, string filename, VC_OPERATION_TYPE type, bool matchFW, string ident="", bool isForeground=false)
- RETURN_CODE [device_sendConfiguration](#) (string filename, VC_OPERATION_TYPE type, bool matchFW, string ident="", bool isForeground=false)
- RETURN_CODE [device_readConfigurationToMemory](#) (string memo, Configs.InstallRules rules, List< Configs.DeviceCommand > cmds, string ident="", bool isForeground=false)
- RETURN_CODE [device_getConfigurationFromMemory](#) (ref string json, string ident="")
- RETURN_CODE [device_RemoteKeyInjection](#) (RKI_KEY_TYPE type, string keyName, string ident="", bool performOnForeground=false)

Static Public Member Functions

- static void [initSC](#) ()
- static int [getCommandTimeout](#) (string ident="")
- static void [setCommandTimeout](#) (int milliseconds, string ident="")
- static void [setCallback](#) (Callback my_Callback)
- static void [setCallback](#) (IntPtr my_Callback, SynchronizationContext context)
- static RETURN_CODE [device_updateFirmwareType](#) (FIRMWARE_TYPE type, byte[] firmwareData, string ident="", UInt32 address=0, bool performOnForeground=false, int manifestScriptsCount=0, int processedScriptCount=0, string DT_PRJ_filename="")
- static String [SDK_Version](#) ()
- static void [lcd_retrieveMessage](#) (DisplayMessages.DISPLAY_MESSAGE_LANGUAGE lang, DisplayMessages.DISPLAY_MESSAGE_IDENTIFIER id, ref string line1, ref string line2)

Properties

- static [IDT_SREDKey2 SharedController](#) [get]

21.15.1 Detailed Description

Class for SREDKey2 MSR reader

21.15.2 Member Function Documentation

21.15.2.1 RETURN_CODE IDTechSDK.IDT_SREDKey2.config_getModelNumber (ref string *response*, string *ident* = " ")

Polls device for Model Number

Parameters

<i>response</i>	Returns Model Number
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString

21.15.2.2 RETURN_CODE IDTechSDK.IDT_SREDKey2.config_getSerialNumber (ref string *response*, string *ident* = " ")

Polls device for Serial Number

Parameters

<i>response</i>	Returns Serial Number
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString

21.15.2.3 RETURN_CODE IDTechSDK.IDT_SREDKey2.config_getStatusKeySlots (ref byte[] *keyslots*, string *ident* = " ")

Get status of the key slots

There are 10 keyslots for PIN Key at Index 1, slot #0-9 There are 10 keyslots for DATA Key at Index 2, slots #0-9
There is a keyslot for MAC Key at Index 5, slot 0 There is a keyslot for Key Encryption Key at Index 14, slot 0

This routine will return all the key slots values as one continuous byte[] of 4-byte key slot info.

- Byte 0 = Key Index Number
- Byte 1-2 = Key Slot Number
- Byte 3 = State: 0x00 = Unused, 0x01 = Valid, 0x02 = End of life, 0xFF = Not Available

Parameters

<i>keySlots</i>	Key slot data
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString

21.15.2.4 RETURN_CODE IDTechSDK.IDT_SREDKey2.device_certificateType (ref int *type*, string *ident* = " ")

Device Certificate Type

Returns the device certificate type

Parameters

<i>type</i>	0 = Unknown, 1 = Demo, 2 = Production
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString

21.15.2.5 RETURN_CODE IDTechSDK.IDT_SREDKey2.device_enableAdminKey (bool *enable*, string *ident* = " ")

Enable Admin Key

Parameters

<i>enable</i>	TRUE = enable, FALSE = disable
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with errorCode.getErrorString(string ident = "")

21.15.2.6 RETURN_CODE IDTechSDK.IDT_SREDKey2.device_getAnyRKIStatus (bool *isMultiKey*, ref string *status*, ref Dictionary< string, RKI_KEY_TYPE > *keys*, string *ident* = " ")

Get RKI Status Polls the RKI servers to validate device RKI eligibility. Note: if device type is known in advance (production or demo device), it is more efficient to use device_getRKIStatus instead

Parameters

<i>isMultiKey</i>	True = MultiKey, False = SingleKey
<i>status</i>	RKI status returned as a string
<i>keys</i>	Returns all available keys
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with `errorCode.getErrorString()`. When no data is available, return code = RETURN_CODE_NO_DATA_AVAILABLE

21.15.2.7 RETURN_CODE IDTechSDK.IDT_SREDKey2.device_getConfigurationFromMemory (ref string *json*, string *ident* = " ")

Read Device Configuration From Memory Retrieves the ViVOconfig configuration data from memory after a successful device_readConfigurationToMemory

Parameters

<i>json</i>	Returns the .json file data that can be saved to disk
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with `errorCode.getErrorString()`. When no data is available, return code = RETURN_CODE_NO_DATA_AVAILABLE

21.15.2.8 RETURN_CODE IDTechSDK.IDT_SREDKey2.device_getFirmwareVersion (ref string *response*, string *ident* = " ")

Polls device for Firmware Version

Parameters

<i>response</i>	Response returned of Firmware Version
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with `device_getResponseCodeString`

21.15.2.9 RETURN_CODE IDTechSDK.IDT_SREDKey2.device_getOutputType (bool *isSwipe*, ref int *response*, string *ident* = " ")

Get Captured Data Output Type

Parameters

<i>isSwipe</i>	TRUE = settings for Swipe, FALSE = settings for MSR (only applicable for Original/Enhanced formats, string ident = "") NOTE: IF XML is chosen, this parameter is ignored
<i>response</i>	<ul style="list-style-type: none"> • 0 : Original Format • 1 : Enhanced Format • 2 : XML Format
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString

21.15.2.10 string IDTechSDK.IDT_SREDKey2.device_getResponseCodeString (RETURN_CODE *eCode*)

Get the description of response result.

Parameters

<i>eCode</i>	the response result.
--------------	----------------------

Return values

<i>the</i>	string for description of response result
------------	---

- case 0: "no error, beginning task";
- case 1: "no response from reader";
- case 2: "invalid response data";
- case 3: "time out for task or CMD";
- case 4: "wrong parameter";
- case 5: "SDK is doing MSR or ICC task";
- case 6: "SDK is doing PINPad task";
- case 7: "SDK is doing CTLS task";
- case 8: "SDK is doing EMV task";
- case 9: "SDK is doing Other task";
- case 10: "err response or data";
- case 11: "no reader attached";
- case 12: "mono audio is enabled";
- case 13: "did connection";
- case 14: "audio volume is too low";
- case 15: "task or CMD be canceled";
- case 16: "UF wrong string format";
- case 17: "UF file not found";
- case 18: "UF wrong file format";
- case 19: "Attempt to contact online host failed";
- case 20: "Attempt to perform RKI failed";
- case 0x300: "Key Type(TDES) of Session Key is not same as the related Master Key.";
- case 0x400: "Related Key was not loaded.";
- case 0x500: "Key Same.";

- case 0x501: "Key is all zero";
- case 0x502: "TR-31 format error";
- case 0x702: "PAN is Error Key.";
- case 0x705: "No Internal MSR PAN (or Internal MSR PAN is erased timeout)";
- case 0X0C01: "Incorrect Frame Tag";
- case 0X0C02: "Incorrect Frame Type";
- case 0X0C03: "Unknown Frame Type";
- case 0X0C04: "Unknown Command";
- case 0X0C05: "Unknown Sub-Command";
- case 0X0C06: "CRC Error";
- case 0X0C07: "Failed";
- case 0X0C08: "Timeout";
- case 0X0C0A: "Incorrect Parameter";
- case 0X0C0B: "Command Not Supported";
- case 0X0C0C: "Sub-Command Not Supported";
- case 0X0C0D: "Parameter Not Supported / Status Abort Command";
- case 0X0C0F: "Sub-Command Not Allowed";
- case 0X0D01: "Incorrect Header Tag";
- case 0X0D02: "Unknown Command";
- case 0X0D03: "Unknown Sub-Command";
- case 0X0D04: "CRC Error in Frame";
- case 0X0D05: "Incorrect Parameter";
- case 0X0D06: "Parameter Not Supported";
- case 0X0D07: "Mal-formatted Data";
- case 0X0D08: "Timeout";
- case 0X0D0A: "Failed / NACK";
- case 0X0D0B: "Command not Allowed";
- case 0X0D0C: "Sub-Command not Allowed";
- case 0X0D0D: "Buffer Overflow (Data Length too large for reader buffer)";
- case 0X0D0E: "User Interface Event";
- case 0X0D11: "Communication type not supported, VT-1, burst, etc.";
- case 0X0D12: "Secure interface is not functional or is in an intermediate state.";
- case 0X0D13: "Data field is not mod 8";
- case 0X0D14: "Pad 0x80 not found where expected";
- case 0X0D15: "Specified key type is invalid";
- case 0X0D1: "Could not retrieve key from the SAM(InitSecureComm)";

- case 0X0D17: "Hash code problem";
- case 0X0D18: "Could not store the key into the SAM(InstallKey)";
- case 0X0D19: "Frame is too large";
- case 0X0D1A: "Unit powered up in authentication state but POS must resend the InitSecureComm command";
- case 0X0D1B: "The EEPROM may not be initialized because SecCommInterface does not make sense";
- case 0X0D1C: "Problem encoding APDU";
- case 0X0D20: "Unsupported Index(ILM) SAM Transceiver error – problem communicating with the SAM(Key Mgr)";
- case 0X0D2: "Unexpected Sequence Counter in multiple frames for single bitmap(ILM) Length error in data returned from the SAM(Key Mgr)";
- case 0X0D22: "Improper bit map(ILM)";
- case 0X0D23: "Request Online Authorization";
- case 0X0D24: "ViVOCard3 raw data read successful";
- case 0X0D25: "Message index not available(ILM) ViVOcomm activate transaction card type(ViVOcomm)";
- case 0X0D26: "Version Information Mismatch(ILM)";
- case 0X0D27: "Not sending commands in correct index message index(ILM)";
- case 0X0D28: "Time out or next expected message not received(ILM)";
- case 0X0D29: "ILM languages not available for viewing(ILM)";
- case 0X0D2A: "Other language not supported(ILM)";
- case 0X0D41: "Unknown Error from SAM";
- case 0X0D42: "Invalid data detected by SAM";
- case 0X0D43: "Incomplete data detected by SAM";
- case 0X0D44: "Reserved";
- case 0X0D45: "Invalid key hash algorithm";
- case 0X0D46: "Invalid key encryption algorithm";
- case 0X0D47: "Invalid modulus length";
- case 0X0D48: "Invalid exponent";
- case 0X0D49: "Key already exists";
- case 0X0D4A: "No space for new RID";
- case 0X0D4B: "Key not found";
- case 0X0D4C: "Crypto not responding";
- case 0X0D4D: "Crypto communication error";
- case 0X0D4E: "Module-specific error for Key Manager";
- case 0X0D4F: "All key slots are full (maximum number of keys has been installed)";
- case 0X0D50: "Auto-Switch OK";
- case 0X0D51: "Auto-Switch failed";

- case 0X0D90: "Account DUKPT Key not exist";
- case 0X0D91: "Account DUKPT Key KSN exhausted";
- case 0x0D00: "This Key had been loaded.";
- case 0x0E00: "Base Time was loaded.";
- case 0x0F00: "Encryption Or Decryption Failed.";
- case 0x1000: "Battery Low Warning (It is High Priority Response while Battery is Low.)";
- case 0x1800: "Send "Cancel Command" after send "Get Encrypted PIN" & "Get Numeric "& "Get Amount"";
- case 0x1900: "Press "Cancel" key after send "Get Encrypted PIN" & "Get Numeric "& "Get Amount"";
- case 0x30FF: "Security Chip is not connect";
- case 0x3000: "Security Chip is deactivation & Device is In Removal Legally State.";
- case 0x3101: "Security Chip is activation & Device is In Removal Legally State.";
- case 0x5500: "No Admin DUKPT Key.";
- case 0x5501: "Admin DUKPT Key STOP.";
- case 0x5502: "Admin DUKPT Key KSN is Error.";
- case 0x5503: "Get Authentication Code1 Failed.";
- case 0x5504: "Validate Authentication Code Error.";
- case 0x5505: "Encrypt or Decrypt data failed.";
- case 0x5506: "Not Support the New Key Type.";
- case 0x5507: "New Key Index is Error.";
- case 0x5508: "Step Error.";
- case 0x5509: "KSN Error";
- case 0x550A: "MAC Error.";
- case 0x550B: "Key Usage Error.";
- case 0x550C: "Mode Of Use Error.";
- case 0x550F: "Other Error.";
- case 0x6000: "Save or Config Failed / Or Read Config Error.";
- case 0x6200: "No Serial Number.";
- case 0x6900: "Invalid Command - Protocol is right, but task ID is invalid.";
- case 0x6A01: "Unsupported Command – Protocol and task ID are right, but command is invalid – In this State";
- case 0x6A00: "Unsupported Command - Protocol and task ID are right, but command is invalid.";
- case 0x6B00: "Unknown parameter in command - Protocol task ID and command are right, but parameter is invalid.";
- case 0x6C00: "Unknown parameter in command – Protocol task ID and command are right, but length is out of the requirement.";
- case 0x7200: "Device is suspend (MKSK suspend or press password suspend).";
- case 0x7300: "PIN DUKPT is STOP (21 bit 1).";

- case 0x7400: "Device is Busy.";
- case 0xE100: "Can not enter sleep mode";
- case 0xE200: "File has existed";
- case 0xE300: "File has not existed";
- case 0xE313: "IO line low -- Card error after session start";
- case 0xE400: "Open File Error";
- case 0xE500: "SmartCard Error";
- case 0xE600: "Get MSR Card data is error";
- case 0xE700: "Command time out";
- case 0xE800: "File read or write is error";
- case 0xE900: "Active 1850 error!";
- case 0xEA00: "Load bootloader error";
- case 0xEF00: "Protocol Error- STX or ETX or check error.";
- case 0xEB00: "Picture is not exist";
- case 0x2C02: "No Microprocessor ICC seated";
- case 0x2C06: "no card seated to request ATR";
- case 0x2D01: "Card Not Supported,";
- case 0x2D03: "Card Not Supported, wants CRC";
- case 0x690D: "Command not supported on reader without ICC support";
- case 0x8100: "ICC error time out on power-up";
- case 0x8200: "invalid TS character received - Wrong operation step";
- case 0x8300: "Decode MSR Error";
- case 0x8400: "TriMagII no Response";
- case 0x8500: "No Swipe MSR Card";
- case 0x8510: "No Financial Card";
- case 0x8600: "Unsupported F, D, or combination of F and D";
- case 0x8700: "protocol not supported EMV TD1 out of range";
- case 0x8800: "power not at proper level";
- case 0x8900: "ATR length too long";
- case 0x8B01: "EMV invalid TA1 byte value";
- case 0x8B02: "EMV TB1 required";
- case 0x8B03: "EMV Unsupported TB1 only 00 allowed";
- case 0x8B04: "EMV Card Error, invalid BWI or CWI";
- case 0x8B06: "EMV TB2 not allowed in ATR";
- case 0x8B07: "EMV TC2 out of range";
- case 0x8B08: "EMV TC2 out of range";

- case 0x8B09: "per EMV96 TA3 must be > 0xF";
- case 0x8B10: "ICC error on power-up";
- case 0x8B11: "EMV T=1 then TB3 required";
- case 0x8B12: "Card Error, invalid BWI or CWI";
- case 0x8B13: "Card Error, invalid BWI or CWI";
- case 0x8B17: "EMV TC1/TB3 conflict-";
- case 0x8B20: "EMV TD2 out of range must be T=1";
- case 0x8C00: "TCK error";
- case 0xA304: "connector has no voltage setting";
- case 0xA305: "ICC error on power-up invalid (SBLK(IFSD) exchange";
- case 0xE301: "ICC error after session start";
- case 0xFF00: "Request to go online";
- case 0xFF01: "EMV: Accept the offline transaction";
- case 0xFF02: "EMV: Decline the offline transaction";
- case 0xFF03: "EMV: Accept the online transaction";
- case 0xFF04: "EMV: Decline the online transaction";
- case 0xFF05: "EMV: Application may fallback to magstripe technology";
- case 0xFF06: "EMV: ICC detected tah the conditions of use are not satisfied";
- case 0xFF07: "EMV: ICC didn't accept transaction";
- case 0xFF08: "EMV: Transaction was cancelled";
- case 0xFF09: "EMV: Application was not selected by kernel or ICC format error or ICC missing data error";
- case 0xFF0A: "EMV: Transaction is terminated";
- case 0xFF0B: "EMV: Other EMV Error";
- case 0xFFFF: "NO RESPONSE";
- case 0xF002: "ICC communication timeout";
- case 0xF003: "ICC communication Error";
- case 0xF00F: "ICC Card Seated and Highest Priority, disable MSR work request";
- case 0xF200: "AID List / Application Data is not exist";
- case 0xF201: "Terminal Data is not exist";
- case 0xF202: "TLV format is error";
- case 0xF203: "AID List is full";
- case 0xF204: "Any CA Key is not exist";
- case 0xF205: "CA Key RID is not exist";
- case 0xF206: "CA Key Index it not exist";
- case 0xF207: "CA Key is full";
- case 0xF208: "CA Key Hash Value is Error";

- case 0xF209: "Transaction format error";
- case 0xF20A: "The command will not be processing";
- case 0xF20B: "CRL is not exist";
- case 0xF20C: "CRL number exceed max number";
- case 0xF20D: "Amount,Other Amount,Trasaction Type are missing";
- case 0xF20E: "The Identification of algorithm is mistake";
- case 0xF20F: "No Financial Card";
- case 0xF210: "In Encrypt Result state, TLV total Length is greater than Max Length";
- case 0x1001: "INVALID ARG";
- case 0x1002: "FILE_OPEN_FAILED";
- case 0x1003: "FILE OPERATION_FAILED";
- case 0x2001: "MEMORY_NOT_ENOUGH";
- case 0x3002: "SMARTCARD_FAIL";
- case 0x3003: "SMARTCARD_INIT_FAILED";
- case 0x3004: "FALLBACK_SITUATION";
- case 0x3005: "SMARTCARD_ABSENT";
- case 0x3006: "SMARTCARD_TIMEOUT";
- case 0x5001: "EMV_PARSING_TAGS_FAILED";
- case 0x5002: "EMV_DUPLICATE_CARD_DATA_ELEMENT";
- case 0x5003: "EMV_DATA_FORMAT_INCORRECT";
- case 0x5004: "EMV_NO_TERM_APP";
- case 0x5005: "EMV_NO_MATCHING_APP";
- case 0x5006: "EMV_MISSING_MANDATORY_OBJECT";
- case 0x5007: "EMV_APP_SELECTION_RETRY";
- case 0x5008: "EMV_GET_AMOUNT_ERROR";
- case 0x5009: "EMV_CARD_REJECTED";
- case 0x5010: "EMV_AIP_NOT_RECEIVED";
- case 0x5011: "EMV_AFL_NOT_RECEIVED";
- case 0x5012: "EMV_AFL_LEN_OUT_OF_RANGE";
- case 0x5013: "EMV_SFI_OUT_OF_RANGE";
- case 0x5014: "EMV_AFL_INCORRECT";
- case 0x5015: "EMV_EXP_DATE_INCORRECT";
- case 0x5016: "EMV_EFF_DATE_INCORRECT";
- case 0x5017: "EMV_ISS_COD_TBL_OUT_OF_RANGE";
- case 0x5018: "EMV_CRYPTOGAM_TYPE_INCORRECT";
- case 0x5019: "EMV_PSE_NOT_SUPPORTED_BY_CARD";

- case 0x5020: "EMV_USER_SELECTED_LANGUAGE";
- case 0x5021: "EMV_SERVICE_NOT_ALLOWED";
- case 0x5022: "EMV_NO_TAG_FOUND";
- case 0x5023: "EMV_CARD_BLOCKED";
- case 0x5024: "EMV_LEN_INCORRECT";
- case 0x5025: "CARD_COM_ERROR";
- case 0x5026: "EMV_TSC_NOT_INCREASED";
- case 0x5027: "EMV_HASH_INCORRECT";
- case 0x5028: "EMV_NO_ARC";
- case 0x5029: "EMV_INVALID_ARC";
- case 0x5030: "EMV_NO_ONLINE_COMM";
- case 0x5031: "TRAN_TYPE_INCORRECT";
- case 0x5032: "EMV_APP_NO_SUPPORT";
- case 0x5033: "EMV_APP_NOT_SELECT";
- case 0x5034: "EMV_LANG_NOT_SELECT";
- case 0x5035: "EMV_NO_TERM_DATA";
- case 0x6001: "CVM_TYPE_UNKNOWN";
- case 0x6002: "CVM_AIP_NOT_SUPPORTED";
- case 0x6003: "CVM_TAG_8E_MISSING";
- case 0x6004: "CVM_TAG_8E_FORMAT_ERROR";
- case 0x6005: "CVM_CODE_IS_NOT_SUPPORTED";
- case 0x6006: "CVM_COND_CODE_IS_NOT_SUPPORTED";
- case 0x6007: "NO_MORE_CVM";
- case 0x6008: "PIN_BYPASSED_BEFORE";
- case 0x7001: "PK_BUFFER_SIZE_TOO_BIG";
- case 0x7002: "PK_FILE_WRITE_ERROR";
- case 0x7003: "PK_HASH_ERROR";
- case 0x8001: "NO_CARD_HOLDER_CONFIRMATION";
- case 0x8002: "GET_ONLINE_PIN";
- case 0xD000: "Data not exist";
- case 0xD001: "Data access error";
- case 0xD100: "RID not exist";
- case 0xD101: "RID existed";
- case 0xD102: "Index not exist";
- case 0xD200: "Maximum exceeded";
- case 0xD201: "Hash error";

- case 0xD205: "System Busy";
- case 0x0E01: "Unable to go online";
- case 0x0E02: "Technical Issue";
- case 0x0E03: "Declined";
- case 0x0E04: "Issuer Referral transaction";
- case 0x0F01: "Decline the online transaction";
- case 0x0F02: "Request to go online";
- case 0x0F03: "Transaction is terminated";
- case 0x0F05: "Application was not selected by kernel or ICC format error or ICC missing data error";
- case 0x0F07: "ICC didn't accept transaction";
- case 0x0F0A: "Application may fallback to magstripe technology";
- case 0x0F0C: "Transaction was cancelled";
- case 0x0F0D: "Timeout";
- case 0x0F0F: "Other EMV Error";
- case 0x0F10: "Accept the offline transaction";
- case 0x0F11: "Decline the offline transaction";
- case 0x0F21: "ICC detected tah the conditions of use are not satisfied";
- case 0x0F22: "No app were found on card matching terminal configuration";
- case 0x0F23: "Terminal file does not exist";
- case 0x0F24: "CAPK file does not exist";
- case 0x0F25: "CRL Entry does not exist";
- case 0x0FFE: "code when blocking is disabled";
- case 0x0FFF: "code when command is not applicable on the selected device";
- case 0xF005: "ICC Encrypted C-APDU Data Structure Length Error Or Format Error.";
- case 0xBBE0: "CM100 Success";
- case 0xBBE1: "CM100 Parameter Error";
- case 0xBBE2: "CM100 Low Output Buffer";
- case 0xBBE3: "CM100 Card Not Found";
- case 0xBBE4: "CM100 Collision Card Exists";
- case 0xBBE5: "CM100 Too Many Cards Exist";
- case 0xBBE6: "CM100 Saved Data Does Not Exist";
- case 0xBBE8: "CM100 No Data Available";
- case 0xBBE9: "CM100 Invalid CID Returned";
- case 0xBBEA: "CM100 Invalid Card Exists";
- case 0xBBEC: "CM100 Command Unsupported";
- case 0xBBED: "CM100 Error In Command Process";

- case 0xBBEE: "CM100 Invalid Command";
- case 0X9031: "Unknown command";
- case 0X9032: "Wrong parameter (such as the length of the command is incorrect)";
- case 0X9038: "Wait (the command couldn't be finished in BWT)";
- case 0X9039: "Busy (a previously command has not been finished)";
- case 0X903A: "Number of retries over limit";
- case 0X9040: "Invalid Manufacturing system data";
- case 0X9041: "Not authenticated";
- case 0X9042: "Invalid Master DUKPT Key";
- case 0X9043: "Invalid MAC Key";
- case 0X9044: "Reserved for future use";
- case 0X9045: "Reserved for future use";
- case 0X9046: "Invalid DATA DUKPT Key";
- case 0X9047: "Invalid PIN Pairing DUKPT Key";
- case 0X9048: "Invalid DATA Pairing DUKPT Key";
- case 0X9049: "No nonce generated";
- case 0X9949: "No GUID available. Perform getVersion first.";
- case 0X9950: "MAC Calculation unsuccessful. Check BDK value.";
- case 0X904A: "Not ready";
- case 0X904B: "Not MAC data";
- case 0X9050: "Invalid Certificate";
- case 0X9051: "Duplicate key detected";
- case 0X9052: "AT checks failed";
- case 0X9053: "TR34 checks failed";
- case 0X9054: "TR31 checks failed";
- case 0X9055: "MAC checks failed";
- case 0X9056: "Firmware download failed";
- case 0X9060: "Log is full";
- case 0X9061: "Removal sensor unengaged";
- case 0X9062: "Any hardware problems";
- case 0X9070: "ICC communication timeout";
- case 0X9071: "ICC data error (such check sum error)";
- case 0X9072: "Smart Card not powered up";

21.15.2.11 RETURN_CODE IDTechSDK.IDT_SREDKey2.device_getRKIStatus (bool *isProd*, bool *isMultiKey*, ref string *status*, ref Dictionary< string, RKI_KEY_TYPE > *keys*, string *ident* = " ")

Get RKI Status From Specified Server Polls the RKI server to validate device RKI eligibility

Parameters

<i>isProd</i>	TRUE = poll production RKI server, FALSE = poll demo RKI server
<i>isMultiKey</i>	True = MultiKey, False = SingleKey
<i>status</i>	RKI status returned as a string
<i>keys</i>	Returns dictionary of available keys
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with `errorCode.getErrorString()`. When no data is available, return code = RETURN_CODE_NO_DATA_AVAILABLE

21.15.2.12 RETURN_CODE IDTechSDK.IDT_SREDKey2.device_getTransArmorID (ref string *TID*, string *ident* = " ")

Get TransArmor ID

Parameters

<i>TID</i>	TransArmor ID
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with `errorCode.getErrorString(string ident = "")`

21.15.2.13 RETURN_CODE IDTechSDK.IDT_SREDKey2.device_readConfigurationToMemory (string *memo*, Configs.InstallRules *rules*, List< Configs.DeviceCommand > *cmds*, string *ident* = " ", bool *isForeground* = false)

Read Device Configuration To Memory Executes a ViVOconfig read of device settings and stores the results in memory

Once a ViVOconfig read starts, all messages will be returned to the callback as DeviceState.ViVOconfig. The ViVOconfig operation is complete when the return code from a DeviceState.ViVOconfig message is one of the following values:

- RETURN_CODE_SDK_VIVOCONFIG_READ_CONFIG_SUCCESS = Read process completed successfully. Execute `device_getConfigurationFromMemory` to retrieve the configuration data
- RETURN_CODE_SDK_VIVOCONFIG_READ_CONFIG_FAILED = Read process FAILED.

Parameters

<i>memo</i>	Optional: populates the memo field of the captured .json file
<i>rules</i>	Optional: populates the .json file with available install rules
<i>cmds</i>	Optional: populates the .json file with commands to execute
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.
<i>isForeground</i>	If TRUE, will block program until update complete. Otherwise, FALSE performs update on background.

Returns

RETURN_CODE: Values can be parsed with `errorCode.getErrorString()`. When no data is available, return code = RETURN_CODE_NO_DATA_AVAILABLE

21.15.2.14 RETURN_CODE IDTechSDK.IDT_SREDKey2.device_rebootDevice (string *ident* = " ")

Reboot Device

Executes a command to restart the device.

Parameters

<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.
--------------	--

Returns

RETURN_CODE: Values can be parsed with `device_getResponseCodeString`

21.15.2.15 RETURN_CODE IDTechSDK.IDT_SREDKey2.device_RemoteKeyInjection (RKI_KEY_TYPE *type*, string *keyName*, string *ident* = " ", bool *performOnForeground* = false)

Remote Key Injection Performs a remote key injection for the device

Parameters

<i>type</i>	Remote Key Injection Type
<i>keyName</i>	Name of key (optional)
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.
<i>performOnForeground</i>	TRUE = block during process (default is false)

Returns

RETURN_CODE: Values can be parsed with `errorCode.getErrorString()`. When no data is available, return code = RETURN_CODE_NO_DATA_AVAILABLE

21.15.2.16 RETURN_CODE IDTechSDK.IDT_SREDKey2.device_sendConfiguration (string *filename*, VC_OPERATION_TYPE *type*, bool *matchFW*, string *ident* = " ", bool *isForeground* = false)

Send Configuration Executes a ViVOconfig update or verify to a device

Once a ViVOconfig write or verify starts, all messages will be returned to the callback as `DeviceState.ViVOconfig`. The ViVOconfig operation is complete when the return code from a `DeviceState.ViVOconfig` message is one of the following values:

- RETURN_CODE_SDK_VIVOCONFIG_VERIFY_SUCCESS = Verification process completed successfully. No differences found.
- RETURN_CODE_SDK_VIVOCONFIG_VERIFY_SUCCESS_WITH_WARNING = Verification process completed with warnings.
- RETURN_CODE_SDK_VIVOCONFIG_VERIFY_FAILED = Verification process FAILED
- RETURN_CODE_SDK_VIVOCONFIG_WRITE_SUCCESS = Write process completed successfully.

- RETURN_CODE_SDK_VIVOCONFIG_WRITE_SUCCESS_WITH_WARNING = Write process completed with warnings
- RETURN_CODE_SDK_VIVOCONFIG_WRITE_FAILED = Write process FAILED

Parameters

<i>filename</i>	The .json configuration file to update the device with NOTE: You can also use this parameter to pass the path of where the .json file is located at on local storage instead
<i>filename</i>	The name of the .json configuration file
<i>type</i>	VC_OPERATION_TYPE

- VC_OPERATION_TYPE_WRITE = Write File To Device, Hash must validate
- VC_OPERATION_TYPE_VERIFY = Verify Device With File
- VC_OPERATION_TYPE_WRITE_IGNORE_HASH = Write File To Device, Ignore Hash
- VC_OPERATION_TYPE_WRITE_FIX_HASH = Write File To Device, Fix Hash If Necessary

Parameters

<i>matchFW</i>	TRUE = Device FW must match file FW, FALSE = Don't validate device FW against file
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.
<i>isForeground</i>	If TRUE, will block program until update complete. Otherwise, FALSE performs update on background.

Returns

RETURN_CODE: Values can be parsed with `errorCode.getErrorString()`. When no data is available, return code = RETURN_CODE_NO_DATA_AVAILABLE

21.15.2.17 RETURN_CODE IDTechSDK.IDT_SREDKey2.device_sendConfigurationFromZip (byte[] zip, string filename, VC_OPERATION_TYPE type, bool matchFW, string ident = "", bool isForeground = false)

Send Configuration From Zip Executes a ViVOconfig update or verify to a device using files from a .zip archive

Once a ViVOconfig write or update starts, all messages will be returned to the callback as DeviceState.ViVOconfig. The ViVOconfig operation is complete when the return code from a DeviceState.ViVOconfig message is one of the following values:

- RETURN_CODE_SDK_VIVOCONFIG_VERIFY_SUCCESS = Verification process completed successfully. No differences found.
- RETURN_CODE_SDK_VIVOCONFIG_VERIFY_SUCCESS_WITH_WARNING = Verification process completed with warnings.
- RETURN_CODE_SDK_VIVOCONFIG_VERIFY_FAILED = Verification process FAILED
- RETURN_CODE_SDK_VIVOCONFIG_WRITE_SUCCESS = Write process completed successfully.
- RETURN_CODE_SDK_VIVOCONFIG_WRITE_SUCCESS_WITH_WARNING = Write process completed with warnings
- RETURN_CODE_SDK_VIVOCONFIG_WRITE_FAILED = Write process FAILED

Parameters

<i>zip</i>	.Zip archive containing the .json configuration file and also any associated image assets NOTE: You can also use this parameter to pass the path of where the .zip file is located at on local storage instead
<i>filename</i>	The name of the .json configuration file
<i>type</i>	VC_OPERATION_TYPE

- VC_OPERATION_TYPE_WRITE = Write File To Device, Hash must validate
- VC_OPERATION_TYPE_VERIFY = Verify Device With File
- VC_OPERATION_TYPE_WRITE_IGNORE_HASH = Write File To Device, Ignore Hash
- VC_OPERATION_TYPE_WRITE_FIX_HASH = Write File To Device, Fix Hash If Necessary

Parameters

<i>matchFW</i>	TRUE = Device FW must match file FW, FALSE = Don't validate device FW against file
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.
<i>isForeground</i>	If TRUE, will block program until update complete. Otherwise, FALSE performs update on background.

Returns

RETURN_CODE: Values can be parsed with `errorCode.getErrorString()`. When no data is available, return code = RETURN_CODE_NO_DATA_AVAILABLE

21.15.2.18 RETURN_CODE IDTechSDK.IDT_SREDKey2.device_sendDataCommand (string *cmd*, bool *calcLRC*, ref byte[] *response*, string *ident* = " ")

Send a data command to the device

Sends a command to the device.

Parameters

<i>cmd</i>	String representation of command to execute
<i>calcLRC</i>	If TRUE, this will wrap command with start/length/lrc/sum/end: '{STX}{Len_Low}{Len_High} data {CheckLRC} {CheckSUM} {ETX}'
<i>response</i>	Response data
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with `device_getResponseCodeString`

21.15.2.19 RETURN_CODE IDTechSDK.IDT_SREDKey2.device_sendDataCommand_ext (string *cmd*, bool *calcLRC*, ref byte[] *response*, int *timeout*, bool *noResponse*, string *ident* = " ")

Send a data command to the device - extended

Sends a command to the device.

Parameters

<i>cmd</i>	String representation of command to execute
<i>calcLRC</i>	If TRUE , this will wrap command as NGA with start/length/lrc/sum/end: '{STX}{Len_Low}{Len_High} data {CheckLRC} {CheckSUM} {ETX}'
<i>response</i>	Response data
<i>timeout</i>	Timeout value waiting for response, in milliseconds (1000 = 1 second, string ident = "")
<i>noResponse</i>	if TRUE , this will not wait for a response and immediately return SUCCESS
<i>calcITP</i>	If TRUE , this will wrap command as ITP with start/end/lrc: '{STX} data {ETX}{CheckLRC}'
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with `device_getResponseCodeString`

21.15.2.20 **RETURN_CODE** `IDTechSDK.IDT_SREDKey2.device_sendVivoCommandP2 (byte command, byte subCommand, byte[] data, ref byte[] response, string ident = " ")`

Send Vivo Command Protocol 2

Sends a protocol 2 command to Vivo readers (IDG/NEO, string ident = "")

Parameters

<i>command</i>	Command
<i>subCommand</i>	Sub-Command
<i>data</i>	Data. May be null
<i>response</i>	Response
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with `errorCode.getErrorString(string ident = "")`

21.15.2.21 **RETURN_CODE** `IDTechSDK.IDT_SREDKey2.device_sendVivoCommandP2_ext (byte command, byte subCommand, byte[] data, ref byte[] response, int timeout, bool noResponse, string ident = " ")`

Send Vivo Command Protocol 2 Extended

Sends a protocol 2 command to Vivo readers (IDG/NEO, string ident = "")

Parameters

<i>command</i>	Command
<i>subCommand</i>	Sub-Command
<i>data</i>	Data. May be null
<i>response</i>	Response
<i>timeout</i>	Timeout, in milliseconds (3000 = 3 seconds, string ident = "")
<i>noResponse</i>	TRUE = don't wait for response, FALSE = wait for response defined by timeout
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with `errorCode.getErrorString(string ident = "")`

21.15.2.22 RETURN_CODE IDTechSDK.IDT_SREDKey2.device_sendVivoCommandP3 (byte *command*, byte *subCommand*, byte[] *data*, ref byte[] *response*, string *ident* = " ")

Send Vivo Command Protocol 3

Sends a protocol 3 command to Vivo readers (IDG/NEO, string ident = "")

Parameters

<i>command</i>	Command
<i>subCommand</i>	Sub-Command
<i>data</i>	Data. May be null
<i>response</i>	Response
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with `errorCode.getErrorString(string ident = "")`

21.15.2.23 RETURN_CODE IDTechSDK.IDT_SREDKey2.device_sendVivoCommandP3_ext (byte *command*, byte *subCommand*, byte[] *data*, ref byte[] *response*, int *timeout*, bool *noResponse*, string *ident* = " ")

Send Vivo Command Protocol 3 Extended

Sends a protocol 3 command to Vivo readers (IDG/NEO, string ident = "")

Parameters

<i>command</i>	Command
<i>subCommand</i>	Sub-Command
<i>data</i>	Data. May be null
<i>response</i>	Response
<i>timeout</i>	Timeout, in milliseconds (3000 = 3 seconds, string ident = "")
<i>noResponse</i>	TRUE = don't wait for response, FALSE = wait for response defined by timeout
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with `errorCode.getErrorString(string ident = "")`

21.15.2.24 RETURN_CODE IDTechSDK.IDT_SREDKey2.device_sendVivoCommandP4 (byte *command*, byte *subCommand*, byte[] *data*, ref byte[] *response*, string *ident* = " ")

Send Vivo Command Protocol 4

Sends a protocol 4 command to Vivo readers (IDG/NEO, string ident = "")

Parameters

<i>command</i>	Command
<i>subCommand</i>	Sub-Command
<i>data</i>	Data. May be null
<i>response</i>	Response
<i>ip</i>	Optional IP
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with `errorCode.getErrorString(string ident = "")`

21.15.2.25 RETURN_CODE IDTechSDK.IDT_SREDKey2.device_sendVivoCommandP4_ext (byte *command*, byte *subCommand*, byte[] *data*, ref byte[] *response*, int *timeout*, bool *noResponse*, string *ident* = " ")

Send Vivo Command Protocol 4 Extended

Sends a protocol 4 command to Vivo readers (IDG/NEO, string *ident* = "")

Parameters

<i>command</i>	Command
<i>subCommand</i>	Sub-Command
<i>data</i>	Data. May be null
<i>response</i>	Response
<i>timeout</i>	Timeout, in milliseconds (3000 = 3 seconds, string <i>ident</i> = "")
<i>noResponse</i>	TRUE = don't wait for response, FALSE = wait for response defined by timeout
<i>ip</i>	Optional IP
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with `errorCode.getErrorString(string ident = "")`

21.15.2.26 IDTechSDK.RETURN_CODE IDTechSDK.IDT_SREDKey2.device_setLanguage (int *val*, string *ident* = " ")

Set Device Language.

Parameters

<i>val</i>	Language: 0 = English, 1 = Japanese
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with `device_getResponseCodeString`

21.15.2.27 RETURN_CODE IDTechSDK.IDT_SREDKey2.device_setOutputType (bool *isSwipe*, int *value*, string *ident* = " ")

Set Captured Data Output Type

Parameters

<i>isSwipe</i>	TRUE = settings for Swipe, FALSE = settings for MSR (only applicable for Original/Enhanced formats, string <i>ident</i> = "") NOTE: IF XML is chosen, this parameter is ignored
<i>value</i>	<ul style="list-style-type: none"> • 0 : Original Format • 1 : Enhanced Format • 2 : XML Format
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString

21.15.2.28 RETURN_CODE IDTechSDK.IDT_SREDKey2.device_setPollMode (byte *mode*, string *ident* = " ")

Send Poll Mode

Sets the poll mode for the device. Auto Poll keeps reader active, Poll On Demand only polls when requested by terminal. Valid only in HID Mode.

Parameters

<i>mode</i>	0 = Auto Poll, 1 = Poll On Demand
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with errorCode.getErrorString(string *ident* = "")

21.15.2.29 RETURN_CODE IDTechSDK.IDT_SREDKey2.device_setTransArmorEncryption (byte[] *cert*, string *ident* = " ")

Set TransArmor Encryption

Parameters

<i>cert</i>	Certificate in PEM format or DER format. PEM format must be string data (converted to binary) starting with "----". DER format is binary data
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with errorCode.getErrorString(string *ident* = "")

21.15.2.30 **RETURN_CODE** IDTechSDK.IDT_SREDKey2.device_setTransArmorID (string *TID*, string *ident* = " ")

Set TransArmor ID

Parameters

<i>TID</i>	TransArmor ID
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with `errorCode.getErrorString(string ident = "")`

21.15.2.31 **RETURN_CODE** IDTechSDK.IDT_SREDKey2.device_startRKI (string *ident* = " ")

Start Remote Key Injection

Starts a remote key injection request with IDTech RKI servers.

Parameters

<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.
--------------	--

Returns

RETURN_CODE: Values can be parsed with `device_getResponseCodeString`

21.15.2.32 **RETURN_CODE** IDTechSDK.IDT_SREDKey2.device_updateDeviceFirmware (byte[] *firmwareData*, string *ident* = " ")

Update K81 Firmware

Updates the firmware .

Parameters

<i>firmwareData</i>	Signed binary data of a firmware file provided by IDTech
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with `errorCode.getErrorString(string ident = "")`

After you pass the `firmwareData` file, a new thread will start to execute the firmware download. You will receive status of the progress through callbacks to the `IDTechSDK.Callback()` delegate. The following parameters will be passed back:

- `sender = IDT_DEVICE_Types.IDT_DEVICE_AUGUSTA`
- `state = DeviceState.FirmwareUpdate`
- `transactionResultCode` = status of the firmware update (starting, entering bootloader, applying update, block success, firmware success, string `ident` = "")

- data = File Progress. Four bytes, with bytes [0][1] = current block, and bytes [2][3] = total blocks. 0x00030010 = block 3 of 16

Example code starting a firmware update

```
OpenFileDialog diag = new OpenFileDialog( ident);
diag.Filter = "NGA FW Files|*.fm";

if (diag.ShowDialog() == DialogResult.OK, string ident = "")
{
    byte[] file = File.ReadAllBytes(diag.FileName, ident);
    RETURN_CODE rt = IDT_Device.SharedController.device_updateDeviceFirmware(file, ident);
    if (rt == RETURN_CODE.RETURN_CODE_DO_SUCCESS, string ident = "")
    {
        //Was a success
    }
    else
    {
        //Error starting firmware download
    }
}
```

Example monitoring firmware update status / success

```
private void MessageCallBack(IDTechSDK.IDT_DEVICE_Types type, DeviceState state, byte[] data,
    IDTTransactionData cardData, EMV_Callback emvCallback, RETURN_CODE transactionResultCode, string ident = "")
{
    switch (state, string ident = "")
    {
        case DeviceState.FirmwareUpdate:
            switch (transactionResultCode, string ident = "")
            {
                case RETURN_CODE.RETURN_CODE_FW_STARTING_UPDATE:
                    SetOutputText("Starting Firmware Update\n", ident);
                    break;
                case RETURN_CODE.RETURN_CODE_DO_SUCCESS:
                    SetOutputText("Firmware Update Successful\n", ident);
                    break;
                case RETURN_CODE.RETURN_CODE_APPLYING_FIRMWARE_UPDATE:
                    SetOutputText("Applying Firmware Update...\n", ident);
                    break;
                case RETURN_CODE.RETURN_CODE_ENTERING_BOOTLOADER_MODE:
                    SetOutputText("Entering Bootloader Mode...\n", ident);
                    break;
                case RETURN_CODE.RETURN_CODE_BLOCK_TRANSFER_SUCCESS:

                    int start = data[0] * 0x100 + data[1];
                    int end = data[2] * 0x100 + data[3];

                    SetOutputText("Sent block " + start.ToString() + " of " + end.ToString() + "\n", ident)
                    ;
                    break;
                default:
                    SetOutputText("Firmware Update Error Code: " + "0x" + String.Format("{0:X}", (ushort)
                        transactionResultCode) + ": " + IDTechSDK.errorCode.getErrorString(transactionResultCode) + "\r\n",
                        ident);
                    break;
            }
            break;
    }
}
```

21.15.2.33 RETURN_CODE IDTechSDK.IDT_SREDKey2.device_updateDeviceFromManifest (string filepath, string ident = " ", bool isForeground = false)

Update Device From Manifest

Initiates a device update using a manifest file.

Parameters

<i>filepath</i>	File path of the manifest file. It must be a valid .json manifest file, and must be located in the same folder as all the needed firmware update assets
-----------------	---

Parameters

<i>ip</i>	Optional Device Identifier
<i>isForeground</i>	Optional. If TRUE, function blocks until complete, other if FALSE, runs on background thread
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString

21.15.2.34 **RETURN_CODE** IDTechSDK.IDT_SREDKey2.device_updateFirmwareFromZip (byte[] *zipfile*, string *ident* = " ", bool *isForeground* = false)

Update Firmware From Zip

Performs one or more device firmware updates from firmware files passed as a compressed archive with a valid configuration file (.json format)

Parameters

<i>zipfile</i>	Zip file containing on or more firmware update files and a .json configuration file
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.
<i>isForeground</i>	If TRUE, will block program until firmware update complete. Otherwise, FALSE performs FW update on background.

Returns

RETURN_CODE: Values can be parsed with errorCode.getErrorString(string ident = "")

21.15.2.35 **static RETURN_CODE** IDTechSDK.IDT_SREDKey2.device_updateFirmwareType (FIRMWARE_TYPE *type*, byte[] *firmwareData*, string *ident* = " ", UInt32 *address* = 0, bool *performOnForeground* = false, int *manifestScriptsCount* = 0, int *processedScriptCount* = 0, string *DT_PRJ_filename* = " ") [static]

Update App Firmware

Updates the firmware

Parameters

<i>type</i>	FIRMWARE_TYPE. <ul style="list-style-type: none"> • FIRMWARE_TYPE_External_EMV_CTL2 • FIRMWARE_TYPE_EMV_CLL2_PPSE • FIRMWARE_TYPE_EMV_CLL2_VCPS • FIRMWARE_TYPE_EMV_CLL2_MChip • FIRMWARE_TYPE_EMV_CLL2_DPass • FIRMWARE_TYPE_EMV_CLL2_Amex • FIRMWARE_TYPE_EMV_CLL2_Interac • FIRMWARE_TYPE_EMV_CLL2_CUP • FIRMWARE_TYPE_EMV_CLL2_JCB • FIRMWARE_TYPE_APVAS • FIRMWARE_TYPE_SmartTap • FIRMWARE_TYPE_SCRP • FIRMWARE_TYPE_ADF_SDK • FIRMWARE_TYPE_ADF_App • FIRMWARE_TYPE_1050 • FIRMWARE_TYPE_1050_DEVICE_TREE FIRMWARE_TYPE_DEVICE_TREE_DEF FIRMWARE_TYPE_DEVICE_TREE_PRJ • FIRMWARE_TYPE_K81_BOOTLOADER_A • FIRMWARE_TYPE_K81_BOOTLOADER_B • FIRMWARE_TYPE_K81
<i>firmwareData</i>	Signed binary data of a firmware file provided by IDTech
<i>ip</i>	Optional ip address of device
<i>address</i>	Required for ADF_SDK and ADF_APP. Start address
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with `errorCode.getErrorString(string ident = "")`

After you pass the `firmwareData` file, a new thread will start to execute the firmware download. You will receive status of the progress through callbacks to the `IDTechSDK.Callback()` delegate. The following parameters will be passed back:

- `sender = IDT_DEVICE_Types.IDT_DEVICE_AUGUSTA`
- `state = DeviceState.FirmwareUpdate`
- `transactionResultCode = status of the firmware update (starting, entering bootloader, applying update, block success, firmware success, string ident = "")`
- `data = File Progress. Four bytes, with bytes [0][1] = current block, and bytes [2][3] = total blocks. 0x00030010 = block 3 of 16`

Example code starting a firmware update

```

OpenFileDialog diag = new OpenFileDialog( ident);
diag.Filter = "NGA FW Files|*.fm";

if (diag.ShowDialog() == DialogResult.OK, string ident = "")
{
    byte[] file = File.ReadAllBytes(diag.FileName, ident);
    RETURN_CODE rt = IDT_Device.SharedController.device_updateFirmware(FIRMWARE_TYPE.FIRMWARE_TYPE_1050,
        file, ident);
    if (rt == RETURN_CODE.RETURN_CODE_DO_SUCCESS, string ident = "")
    {
        //Was a success
    }
    else
    {
        //Error starting firmware download
    }
}

```

Example monitoring firmware update status / success

```

private void MessageCallBack(IDTechSDK.IDT_DEVICE_Types type, DeviceState state, byte[] data,
    IDTTransactionData cardData, EMV_Callback emvCallback, RETURN_CODE transactionResultCode, string ident = "")
{
    switch (state, string ident = "")
    {
        case DeviceState.FirmwareUpdate:
            switch (transactionResultCode, string ident = "")
            {
                case RETURN_CODE.RETURN_CODE_FW_STARTING_UPDATE:
                    SetOutputText("Starting Firmware Update\n", ident);
                    break;
                case RETURN_CODE.RETURN_CODE_DO_SUCCESS:
                    SetOutputText("Firmware Update Successful\n", ident);
                    break;
                case RETURN_CODE.RETURN_CODE_APPLYING_FIRMWARE_UPDATE:
                    SetOutputText("Applying Firmware Update...\n", ident);
                    break;
                case RETURN_CODE.RETURN_CODE_ENTERING_BOOTLOADER_MODE:
                    SetOutputText("Entering Bootloader Mode...\n", ident);
                    break;
                case RETURN_CODE.RETURN_CODE_BLOCK_TRANSFER_SUCCESS:

                    int start = data[0] * 0x100 + data[1];
                    int end = data[2] * 0x100 + data[3];

                    SetOutputText("Sent block " + start.ToString() + " of " + end.ToString() + "\n", ident)
                    ;
                    break;
                default:
                    SetOutputText("Firmware Update Error Code: " + "0x" + String.Format("{0:X}", (ushort)
                        transactionResultCode) + ": " + IDTechSDK.errorCode.getErrorString(transactionResultCode) + "\r\n",
                        ident);
                    break;
            }
        break;
    }
}

```

21.15.2.36 static int IDTechSDK.IDT_SREDKey2.getCommandTimeout(string ident = " ") [static]

Get Command Timeout

Gets the default timeout (in milliseconds) waiting for a blocking command response

Return values

<i>time</i>	Time
-------------	------

21.15.2.37 static void IDTechSDK.IDT_SREDKey2.lcd_retrieveMessage (DisplayMessages.DISPLAY_MESSAGE_LANGUAGE *lang*, DisplayMessages.DISPLAY_MESSAGE_IDENTIFIER *id*, ref string *line1*, ref string *line2*) [static]

Retrieve LCD Message

Returns the string value for a message ID returned for LCD messaging

Parameters

<i>lang</i>	Language.
<i>id</i>	Message ID
<i>line1</i>	Line 1 string value
<i>line2</i>	Line 2 string value
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString

21.15.2.38 RETURN_CODE IDTechSDK.IDT_SREDKey2.msr_cancelTransaction (string *ident* = " ")

Cancel MSR Transaction

Valid only in HID Mode

Parameters

<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.
--------------	--

Returns

RETURN_CODE: Values can be parsed with errorCode.getErrorString(string ident = "")

21.15.2.39 RETURN_CODE IDTechSDK.IDT_SREDKey2.msr_disable (string *ident* = " ")

Disable MSR function.

Parameters

<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.
--------------	--

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString

21.15.2.40 RETURN_CODE IDTechSDK.IDT_SREDKey2.msr_enable (string *ident* = " ")

Disable MSR function.

Parameters

<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.
--------------	--

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString

21.15.2.41 RETURN_CODE IDTechSDK.IDT_SREDKey2.msr_getClearPANID (ref byte *value*, string *ident* = " ")

Get Clear PAN Digits

Returns the number of digits that begin the PAN that will be in the clear

Parameters

<i>value</i>	Number of digits in clear. Values are char '0' - '6':
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString

21.15.2.42 RETURN_CODE IDTechSDK.IDT_SREDKey2.msr_getConfiguration (ref byte[] *config*, string *ident* = " ")

Set MSR Configuration

Gets msr configuration data.

Parameters

<i>config</i>	Configuration data to get
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with errorCode.getErrorString(string ident = "")

21.15.2.43 RETURN_CODE IDTechSDK.IDT_SREDKey2.msr_getExpirationMask (ref byte *value*, string *ident* = " ")

Get Expiration Masking

Get the flag that determines if to mask the expiration date

Parameters

<i>value</i>	'0' = masked, '1' = not-masked
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString

21.15.2.44 RETURN_CODE IDTechSDK.IDT_SREDKey2.msr_getFunctionStatus (ref bool *enabled*, ref bool *isBufferMode*, ref bool *withNotification*, string *ident* = " ")

Get MSR Function status Get MSR Function status about enable/disable and with or without seated notification

Parameters

<i>enabled</i>	<ul style="list-style-type: none"> • true: MSR Function enabled. • false: MSR Function disabled.
<i>isBufferMode</i>	<ul style="list-style-type: none"> • true: in the buffer mode. • false: in the auto mode.
<i>withNotification</i>	<ul style="list-style-type: none"> • true: with notification when swiped MSR Card. • false: without notification when swiped MSR Card.
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString

21.15.2.45 RETURN_CODE IDTechSDK.IDT_SREDKey2.msr_getSwipeEncryption (ref byte *encryption*, string *ident* = " ")

Get Swipe Data Encryption

For Non-SRED Augusta Only

Returns the encryption used for swipe data

Parameters

<i>encryption</i>	1 = TDES, 2 = AES, 0 = NONE
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString

21.15.2.46 RETURN_CODE IDTechSDK.IDT_SREDKey2.msr_getSwipeForcedEncryptionOption (ref byte *option*, string *ident* = " ")

Get Swipe Data Encryption

Gets the swipe force encryption options

Parameters

<i>option</i>	Byte using lower four bits as flags. 0 = Force Encryption Off, 1 = Force Encryption On bit0 = Track 1 bit1 = Track 2 bit2 = Track 3 bit4 = Track 3 Card Option 0
---------------	--

Example: Response 0x03 = Track1/Track2 Forced Encryption, Track3/Track3-0 no Forced Encryption

Parameters

<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.
--------------	--

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString

21.15.2.47 RETURN_CODE IDTechSDK.IDT_SREDKey2.msr_getSwipeMaskOption (ref byte *option*, string *ident* = " ")

Get Swipe Mask Option

Gets the swipe mask/clear data sending option

Parameters

<i>option</i>	Byte using lower three bits as flags. 0 = Mask Option Off, 1 = Mask Option On bit0 = Track 1 bit1 = Track 2 bit2 = Track 3
---------------	--

Example: Response 0x03 = Track1/Track2 Masked Option ON, Track3 Masked Option Off

Parameters

<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.
--------------	--

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString

21.15.2.48 RETURN_CODE IDTechSDK.IDT_SREDKey2.msr_getTerminalData (byte[] *tag*, ref byte[] *tlv*, string *ident* = " ")

Get MSR Terminal Data

Gets msr terminal data.

Parameters

<i>tag</i>	Tag value to retrieve (example new byte[]{0xDF, 0xEC, 0x09} to retrieve value for tag DFEC09
<i>tlv</i>	Configuration data to read in TLV format
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with errorCode.getErrorString(string ident = "")

21.15.2.49 RETURN_CODE IDTechSDK.IDT_SREDKey2.msr_resetTerminalData (string *ident* = " ")

Reset MSR Terminal Data

Resets msr terminal data to defaults.

Parameters

<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.
--------------	--

Returns

RETURN_CODE: Values can be parsed with `errorCode.getErrorString(string ident = "")`

21.15.2.50 RETURN_CODE IDTechSDK.IDT_SREDKey2.msr_setClearPANID (byte *val*, string *ident* = " ")

Set Clear PAN Digits

Sets the amount of digits shown in the clear (not masked) at the beginning of the returned PAN value

Parameters

<i>val</i>	Number of digits to show in clear. Range 0-6.
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with `device.getResponseCodeString`

21.15.2.51 RETURN_CODE IDTechSDK.IDT_SREDKey2.msr_setConfiguration (byte[] *config*, string *ident* = " ")

Set MSR Configuration

Sets msr configuration data.

Parameters

<i>config</i>	Configuration data to send
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with `errorCode.getErrorString(string ident = "")`

21.15.2.52 RETURN_CODE IDTechSDK.IDT_SREDKey2.msr_setExpirationMask (bool *mask*, string *ident* = " ")

Set Expiration Masking

Sets the flag to mask the expiration date

Parameters

<i>mask</i>	TRUE = mask expiration
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString

21.15.2.53 RETURN_CODE IDTechSDK.IDT_SREDKey2.msr_setSwipeForcedEncryptionOption (bool *track1*, bool *track2*, bool *track3*, bool *track3card0*, string *ident* = " ")

Set Swipe Force Encryption

Sets the swipe force encryption options

Parameters

<i>track1</i>	Force encrypt track 1
<i>track2</i>	Force encrypt track 2
<i>track3</i>	Force encrypt track 3
<i>track3card0</i>	Force encrypt track 3 when card type is 0
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString

21.15.2.54 RETURN_CODE IDTechSDK.IDT_SREDKey2.msr_setSwipeMaskOption (bool *track1*, bool *track2*, bool *track3*, string *ident* = " ")

Set Swipe Mask Option

Sets the swipe mask/clear data sending option

Parameters

<i>track1</i>	Mask track 1 allowed
<i>track2</i>	Mask track 2 allowed
<i>track3</i>	Mask track 3 allowed
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString

21.15.2.55 RETURN_CODE IDTechSDK.IDT_SREDKey2.msr_setTerminalData (byte[] *tlv*, string *ident* = " ")

Set MSR Terminal Data

Sets msr terminal data.

Parameters

<i>tlv</i>	Configuration data to set in TLV format
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with `errorCode.getErrorString(string ident = "")`

21.15.2.56 RETURN_CODE IDTechSDK.IDT_SREDKey2.msr_startKeyedTransaction (int *timeout*, bool *ZIP*, bool *CSC*, bool *ADR*, bool *Mod10Check*, string *ident* = " ")

Start Swipe Transaction

Enables keypad, awaiting card number input. Valid only when device is in HID mode, and poll on demand mode. Will prompt for PAN and Expiration Date.

```
@param timeout Swipe Timeout Value (0 - 255 seconds)
@ZIP Also Request ZIP
@CSC Also Request CSC
@ADR Also Request Address
@Mod10Check Verify PAN passes validation
```

Parameters

<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.
--------------	--

Returns

RETURN_CODE: Values can be parsed with `errorCode.getErrorString(string ident = "")`

21.15.2.57 RETURN_CODE IDTechSDK.IDT_SREDKey2.msr_startSwipeTransaction (int *timeout*, byte[] *tlv*, string *ident* = " ")

Start Swipe Transaction

Enables MSR, waiting for swipe to occur. Valid only when device is in HID mode, poll on demand mode, and MSR Function ID 0x1A = 0x31.

```
@param timeout Swipe Timeout Value (0 - 255 seconds)
@tlv TLV configuration settings to override existing device tlv configuration settings
```

Parameters

<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.
--------------	--

Returns

RETURN_CODE: Values can be parsed with `errorCode.getErrorString(string ident = "")`

21.15.2.58 RETURN_CODE IDTechSDK.IDT_SREDKey2.msr_switchUSBInterfaceMode (bool *blsUSBKeyboardMode*)

Switch the USB interface mode between USB HID and USB KB mode.

For Non-SRED Augusta Only

Parameters

<i>bIsUSBKeyboardMode</i>	USB interface mode <ul style="list-style-type: none"> • true: Enter into USB Keyboard mode • false: Enter into USB HID mode
---------------------------	---

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString

21.15.2.59 RETURN_CODE IDTechSDK.IDT_SREDKey2.msr_switchUSBInterfaceMode (bool *bIsUSBKeyboardMode*, ref string *ident*)

Switch the USB interface mode between USB HID and USB KB mode.

For Non-SRED Augusta Only

Parameters

<i>bIsUSBKeyboardMode</i>	USB interface mode <ul style="list-style-type: none"> • true: Enter into USB Keyboard mode • false: Enter into USB HID mode
<i>ident</i>	Device ID to send command to.

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString

21.15.2.60 static String IDTechSDK.IDT_SREDKey2.SDK_Version () [static]

SDK Version

- All Devices

Returns the current version of SDK

Returns

Framework version

21.15.2.61 static void IDTechSDK.IDT_SREDKey2.setCallback (Callback *my_Callback*) [static]

Set Callback

Sets the class callback

Parameters

<i>my_Callback</i>	The callback function to receive the response message from device. defined as follows. delegate void CallBack(IDT_DEVICE_Types sender, DeviceState state, byte[] data, IDTTransactionData card, EMV_Callback emvCallback, RETURN_CODE transactionResultCode, ident);
--------------------	---

21.15.2.62 static void IDTechSDK.IDT_SREDKey2.setCallback (IntPtr *my_Callback*, SynchronizationContext *context*)
[static]

Set Callback

Sets the class callback

Parameters

<i>my_Callback</i>	The callback function to receive the response message from device. defined as follows. public unsafe delegate void MFCCallBack(Parameters parameters, ident);
<i>context</i>	The context of the UI thread

21.15.2.63 static void IDTechSDK.IDT_SREDKey2.setCommandTimeout (int *milliseconds*, string *ident* = " ") [static]

Set Command Timeout

Sets the default timeout (in milliseconds) waiting for a blocking command response

Parameters

<i>milliseconds</i>	Time
---------------------	------

21.15.3 Property Documentation

21.15.3.1 IDT_SREDKey2 IDTechSDK.IDT_SREDKey2.SharedController [static],[get]

Singleton Instance

Establishes an singleton instance of [IDT_SREDKey2](#) class.

Returns

Instance of [IDT_SREDKey2](#)

The documentation for this class was generated from the following file:

- /Users/randy/Repo/universal-sdk/IDTechSDK_STD/DeviceHeaders/IDT_SREDKey2.cs

21.16 IDTechSDK.IDT_UniPay Class Reference

Public Member Functions

- RETURN_CODE [device_getFirmwareVersion](#) (ref string response, string ident="")
- RETURN_CODE [device_getBootloaderVersion](#) (ref string response, string ident="")

- RETURN_CODE [config_getModelNumber](#) (ref string response, string ident="")
- RETURN_CODE [config_getSerialNumber](#) (ref string response, string ident="")
- bool [config_setCmdTimeOutDuration](#) (int newTimeOut, string ident="")
- RETURN_CODE [device_getBatteryVoltage](#) (ref string voltage, string ident="")
- RETURN_CODE [device_sendPAE](#) (string command, ref string response, int timeout, string ident="")
- RETURN_CODE [device_rebootDevice](#) (string ident="")
- RETURN_CODE [device_sendDataCommand](#) (string cmd, bool calcLRC, ref byte[] response, string ident="")
- RETURN_CODE [device_startRKI](#) (string ident="")
- RETURN_CODE [icc_exchangeAPDU](#) (string c_APDU, ref byte[] response, string ident="")
- RETURN_CODE [icc_getAPDU_KSN](#) (ref byte[] ksn, string ident="")
- RETURN_CODE [icc_getICCReaderStatus](#) (ref byte status, string ident="")
- RETURN_CODE [icc_getKeyFormatForICCDUKPT](#) (ref byte format, string ident="")
- RETURN_CODE [icc_getKeyTypeForICCDUKPT](#) (ref byte type, string ident="")
- RETURN_CODE [icc_powerOffICC](#) (string ident="")
- RETURN_CODE [icc_powerOnICC](#) (ref byte[] ATR, string ident="")
- RETURN_CODE [icc_setKeyFormatForICCDUKPT](#) (byte encryption, string ident="")
- RETURN_CODE [icc_setKeyTypeForICCDUKPT](#) (byte encryption, string ident="")
- RETURN_CODE [msr_cancelMSRSwipe](#) (string ident="")
- RETURN_CODE [msr_getClearPANID](#) (ref byte value, string ident="")
- RETURN_CODE [msr_getExpirationMask](#) (ref byte value, string ident="")
- RETURN_CODE [msr_getSwipeEncryption](#) (ref byte encryption, string ident="")
- RETURN_CODE [msr_getSwipeForcedEncryptionOption](#) (ref byte option, string ident="")
- RETURN_CODE [msr_getSwipeMaskOption](#) (ref byte option, string ident="")
- RETURN_CODE [msr_setClearPANID](#) (byte val, string ident="")
- RETURN_CODE [msr_setExpirationMask](#) (bool mask, string ident="")
- RETURN_CODE [msr_setSwipeEncryption](#) (byte encryption, string ident="")
- RETURN_CODE [msr_setSwipeForcedEncryptionOption](#) (bool track1, bool track2, bool track3, bool track3card0, string ident="")
- RETURN_CODE [msr_setSetting](#) (byte setting, byte value, string ident="")
- RETURN_CODE [msr_getSetting](#) (byte setting, ref byte value, string ident="")
- RETURN_CODE [msr_setSwipeMaskOption](#) (bool track1, bool track2, bool track3, string ident="")
- RETURN_CODE [msr_startMSRSwipe](#) (int timeout, string ident="")
- RETURN_CODE [device_getRKIStatus](#) (bool isProd, bool isMultiKey, ref string status, ref Dictionary< string, RKI_KEY_TYPE > keys, string ident="")
- RETURN_CODE [device_getAnyRKIStatus](#) (bool isMultiKey, ref string status, ref Dictionary< string, RKI_KEY_TYPE > keys, string ident="")
- RETURN_CODE [device_updateFirmwareFromZip](#) (byte[] zipfile, string ident="", bool isForeground=false)
- RETURN_CODE [device_sendConfigurationFromZip](#) (byte[] zip, string filename, VC_OPERATION_TYPE type, bool matchFW, string ident="", bool isForeground=false)
- RETURN_CODE [device_sendConfiguration](#) (string filename, VC_OPERATION_TYPE type, bool matchFW, string ident="", bool isForeground=false)
- RETURN_CODE [device_readConfigurationToMemory](#) (string memo, Configs.InstallRules rules, List< Configs.DeviceCommand > cmds, string ident="", bool isForeground=false)
- RETURN_CODE [device_getConfigurationFromMemory](#) (ref string json, string ident="")
- RETURN_CODE [device_RemoteKeyInjection](#) (RKI_KEY_TYPE type, string keyName, string ident="", bool performOnForeground=false)

Static Public Member Functions

- static void [initSC](#) ()
- static int [getCommandTimeout](#) (string ident="")
- static void [setCommandTimeout](#) (int milliseconds, string ident="")
- static void [setCallback](#) (Callback my_Callback)
- static void [setCallback](#) (IntPtr my_Callback, SynchronizationContext context)
- static String [SDK_Version](#) ()

Properties

- static [IDT_UniPay SharedController](#) [get]

21.16.1 Detailed Description

Class for UniPay MSR/ICC reader

21.16.2 Member Function Documentation

21.16.2.1 RETURN_CODE IDTechSDK.IDT_UniPay.config_getModelNumber (ref string *response*, string *ident* = " ")

Polls device for Model Number

Parameters

<i>response</i>	Returns Model Number
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with `errorCode.getErrorString(string ident = "")`

21.16.2.2 RETURN_CODE IDTechSDK.IDT_UniPay.config_getSerialNumber (ref string *response*, string *ident* = " ")

Polls device for Serial Number

Parameters

<i>response</i>	Returns Serial Number
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with `errorCode.getErrorString(string ident = "")`

21.16.2.3 bool IDTechSDK.IDT_UniPay.config_setCmdTimeOutDuration (int *newTimeOut*, string *ident* = " ")

Command Acknowledgement Timeout

Sets the amount of seconds to wait for an {ACK} to a command before a timeout. Responses should normally be received under one second. Default is 3 seconds

Parameters

<i>newTimeOut</i>	Timeout value. Valid range 1 - 60 seconds
-------------------	---

Returns

Success flag. Determines if value was set and in range.

21.16.2.4 **RETURN_CODE** IDTechSDK.IDT_UniPay.device_getAnyRKIStatus (bool *isMultiKey*, ref string *status*, ref Dictionary< string, RKI_KEY_TYPE > *keys*, string *ident* = " ")

Get RKI Status Polls the RKI servers to validate device RKI eligibility. Note: if device type is known in advance (production or demo device), it is more efficient to use device_getRKIStatus instead

Parameters

<i>isMultiKey</i>	True = MultiKey, False = SingleKey
<i>status</i>	RKI status returned as a string
<i>keys</i>	Returns all available keys
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with errorCode.getErrorString(). When no data is available, return code = RETURN_CODE_NO_DATA_AVAILABLE

21.16.2.5 **RETURN_CODE** IDTechSDK.IDT_UniPay.device_getBatteryVoltage (ref string *voltage*, string *ident* = " ")

Polls device for Battery Voltage

Parameters

<i>voltage</i>	Returns Battery Voltage as 4-character string * 100. Example: "0186" = 1.86v. "1172" = 11.72v.
----------------	--

Return values

RETURN_CODE	<ul style="list-style-type: none"> • 0x0000: Success: no error - RETURN_CODE_DO_SUCCESS • 0x0001: Disconnect: no response from reader - RETURN_CODE_ERR_DISCONNECT • 0x0002: Invalid Response: invalid response data - RETURN_CODE_ERR_CMD_RESPONSE • 0x0003: Timeout: time out for task or CMD - RETURN_CODE_ERR_TIMEDOUT • 0x0004: Invalid Parameter: wrong parameter - RETURN_CODE_ERR_INVALID_PARAMETER • 0x0005: MSR Busy: SDK is doing MSR or ICC task - RETURN_CODE_SDK_BUSY_MSR • 0x0006: PINPad Busy: SDK is doing PINPad task - RETURN_CODE_SDK_BUSY_PINPAD • 0x0007: Unknown: Unknown error - RETURN_CODE_ERR_OTHER • 0x0100 through 0xFFFF refer to errorCode.getErrorString(string ident = "")
--------------------	--

21.16.2.6 RETURN_CODE IDTechSDK.IDT_UniPay.device_getBootloaderVersion (ref string response, string ident = " ")

Polls device for Bootloader Version

Puts device in bootloader mode so it can return bootloader version. Remains in bootloader mode until timeout of 30 seconds is reached. During this time, only command that may be executed is device_getFirmwareVersion, which will return bootloader version

Parameters

<i>response</i>	Response returned of Bootloader Version
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with errorCode.getErrorString(string ident = "")

21.16.2.7 RETURN_CODE IDTechSDK.IDT_UniPay.device_getConfigurationFromMemory (ref string json, string ident = " ")

Read Device Configuration From Memory Retrieves the ViVOconfig configuration data from memory after a successful device_readConfigurationToMemory

Parameters

<i>json</i>	Returns the .json file data that can be saved to disk
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with errorCode.getErrorString(). When no data is available, return code = RETURN_CODE_NO_DATA_AVAILABLE

21.16.2.8 RETURN_CODE IDTechSDK.IDT_UniPay.device_getFirmwareVersion (ref string response, string ident = " ")

Polls device for Firmware Version

Parameters

<i>response</i>	Response returned of Firmware Version
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with errorCode.getErrorString(string ident = "")

21.16.2.9 RETURN_CODE IDTechSDK.IDT_UniPay.device_getRKIStatus (bool isProd, bool isMultiKey, ref string status, ref Dictionary< string, RKI_KEY_TYPE > keys, string ident = " ")

Get RKI Status From Specified Server Polls the RKI server to validate device RKI eligibility

Parameters

<i>isProd</i>	TRUE = poll production RKI server, FALSE = poll demo RKI server
<i>isMultiKey</i>	True = MultiKey, False = SingleKey
<i>status</i>	RKI status returned as a string
<i>keys</i>	Returns dictionary of available keys
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with `errorCode.getErrorString()`. When no data is available, return code = RETURN_CODE_NO_DATA_AVAILABLE

21.16.2.10 RETURN_CODE IDTechSDK.IDT_UniPay.device_readConfigurationToMemory (string memo, Configs.InstallRules rules, List< Configs.DeviceCommand > cmds, string ident = " ", bool isForeground = false)

Read Device Configuration To Memory Executes a ViVOconfig read of device settings and stores the results in memory

Once a ViVOconfig read starts, all messages will be returned to the callback as `DeviceState.ViVOconfig`. The ViVOconfig operation is complete when the return code from a `DeviceState.ViVOconfig` message is one of the following values:

- RETURN_CODE_SDK_VIVOCONFIG_READ_CONFIG_SUCCESS = Read process completed successfully. Execute `device_getConfigurationFromMemory` to retrieve the configuration data
- RETURN_CODE_SDK_VIVOCONFIG_READ_CONFIG_FAILED = Read process FAILED.

Parameters

<i>memo</i>	Optional: populates the memo field of the captured .json file
<i>rules</i>	Optional: populates the .json file with available install rules
<i>cmds</i>	Optional: populates the .json file with commands to execute
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.
<i>isForeground</i>	If TRUE, will block program until update complete. Otherwise, FALSE performs update on background.

Returns

RETURN_CODE: Values can be parsed with `errorCode.getErrorString()`. When no data is available, return code = RETURN_CODE_NO_DATA_AVAILABLE

21.16.2.11 RETURN_CODE IDTechSDK.IDT_UniPay.device_rebootDevice (string ident = " ")

Reboot Device

Executes a command to restart the device.

Parameters

<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.
--------------	--

Returns

RETURN_CODE: Values can be parsed with `errorCode.getErrorString(string ident = "")`

21.16.2.12 `RETURN_CODE IDTechSDK.IDT_UniPay.device_RemoteKeyInjection (RKI_KEY_TYPE type, string keyName, string ident = " ", bool performOnForeground = false)`

Remote Key Injection Performs a remote key injection for the device

Parameters

<i>type</i>	Remote Key Injection Type
<i>keyName</i>	Name of key (optional)
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.
<i>performOnForeground</i>	TRUE = block during process (default is false)

Returns

RETURN_CODE: Values can be parsed with `errorCode.getErrorString()`. When no data is available, return code = RETURN_CODE_NO_DATA_AVAILABLE

21.16.2.13 `RETURN_CODE IDTechSDK.IDT_UniPay.device_sendConfiguration (string filename, VC_OPERATION_TYPE type, bool matchFW, string ident = " ", bool isForeground = false)`

Send Configuration Executes a ViVOconfig update or verify to a device

Once a ViVOconfig write or verify starts, all messages will be returned to the callback as `DeviceState.ViVOconfig`. The ViVOconfig operation is complete when the return code from a `DeviceState.ViVOconfig` message is one of the following values:

- RETURN_CODE_SDK_VIVOCONFIG_VERIFY_SUCCESS = Verification process completed successfully. No differences found.
- RETURN_CODE_SDK_VIVOCONFIG_VERIFY_SUCCESS_WITH_WARNING = Verification process completed with warnings.
- RETURN_CODE_SDK_VIVOCONFIG_VERIFY_FAILED = Verification process FAILED
- RETURN_CODE_SDK_VIVOCONFIG_WRITE_SUCCESS = Write process completed successfully.
- RETURN_CODE_SDK_VIVOCONFIG_WRITE_SUCCESS_WITH_WARNING = Write process completed with warnings
- RETURN_CODE_SDK_VIVOCONFIG_WRITE_FAILED = Write process FAILED

Parameters

<i>filename</i>	The .json configuration file to update the device with NOTE: You can also use this parameter to pass the path of where the .json file is located at on local storage instead
<i>filename</i>	The name of the .json configuration file
<i>type</i>	VC_OPERATION_TYPE

- VC_OPERATION_TYPE_WRITE = Write File To Device, Hash must validate
- VC_OPERATION_TYPE_VERIFY = Verify Device With File

- VC_OPERATION_TYPE_WRITE_IGNORE_HASH = Write File To Device, Ignore Hash
- VC_OPERATION_TYPE_WRITE_FIX_HASH = Write File To Device, Fix Hash If Necessary

Parameters

<i>matchFW</i>	TRUE = Device FW must match file FW, FALSE = Don't validate device FW against file
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.
<i>isForeground</i>	If TRUE, will block program until update complete. Otherwise, FALSE performs update on background.

Returns

RETURN_CODE: Values can be parsed with `errorCode.getErrorString()`. When no data is available, return code = RETURN_CODE_NO_DATA_AVAILABLE

21.16.2.14 RETURN_CODE IDTechSDK.IDT_UniPay.device_sendConfigurationFromZip (*byte[] zip*, *string filename*, *VC_OPERATION_TYPE type*, *bool matchFW*, *string ident* = " ", *bool isForeground* = false)

Send Configuration From Zip Executes a ViVOconfig update or verify to a device using files from a .zip archive

Once a ViVOconfig write or update starts, all messages will be returned to the callback as DeviceState.ViVOconfig. The ViVOconfig operation is complete when the return code from a DeviceState.ViVOconfig message is one of the following values:

- RETURN_CODE_SDK_VIVOCONFIG_VERIFY_SUCCESS = Verification process completed successfully. No differences found.
- RETURN_CODE_SDK_VIVOCONFIG_VERIFY_SUCCESS_WITH_WARNING = Verification process completed with warnings.
- RETURN_CODE_SDK_VIVOCONFIG_VERIFY_FAILED = Verification process FAILED
- RETURN_CODE_SDK_VIVOCONFIG_WRITE_SUCCESS = Write process completed successfully.
- RETURN_CODE_SDK_VIVOCONFIG_WRITE_SUCCESS_WITH_WARNING = Write process completed with warnings
- RETURN_CODE_SDK_VIVOCONFIG_WRITE_FAILED = Write process FAILED

Parameters

<i>zip</i>	.Zip archive containing the .json configuration file and also any associated image assets NOTE: You can also use this parameter to pass the path of where the .zip file is located at on local storage instead
<i>filename</i>	The name of the .json configuration file
<i>type</i>	VC_OPERATION_TYPE

- VC_OPERATION_TYPE_WRITE = Write File To Device, Hash must validate
- VC_OPERATION_TYPE_VERIFY = Verify Device With File
- VC_OPERATION_TYPE_WRITE_IGNORE_HASH = Write File To Device, Ignore Hash
- VC_OPERATION_TYPE_WRITE_FIX_HASH = Write File To Device, Fix Hash If Necessary

Parameters

<i>matchFW</i>	TRUE = Device FW must match file FW, FALSE = Don't validate device FW against file
----------------	--

Parameters

<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.
<i>isForeground</i>	If TRUE, will block program until update complete. Otherwise, FALSE performs update on background.

Returns

RETURN_CODE: Values can be parsed with `errorCode.getErrorString()`. When no data is available, return code = RETURN_CODE_NO_DATA_AVAILABLE

21.16.2.15 RETURN_CODE IDTechSDK.IDT_UniPay.device_sendDataCommand (string *cmd*, bool *calcLRC*, ref byte[] *response*, string *ident* = " ")

Send a NSData object to device

Sends a command represented by the provide NSData object to the device through the accessory protocol.

Parameters

<i>cmd</i>	NSData representation of command to execute
<i>calcLRC</i>	If TRUE, this will wrap command with start/length/lrc/sum/end: '{STX}{Len_Low}{Len_High} data {CheckLRC} {CheckSUM} {ETX}'
<i>response</i>	Response data
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with `errorCode.getErrorString(string ident = "")`

21.16.2.16 RETURN_CODE IDTechSDK.IDT_UniPay.device_sendPAE (string *command*, ref string *response*, int *timeout*, string *ident* = " ")

Send Payment Application Engine Command

Executes a PAE command

Parameters

<i>command</i>	ASCII command string, should start with "*PAE"
<i>response</i>	command response
<i>timeout</i>	timeout waiting for PAE response
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with `errorCode.getErrorString(string ident = "")`

21.16.2.17 RETURN_CODE IDTechSDK.IDT_UniPay.device_startRKI (string *ident* = " ")

Start Remote Key Injection

Starts a remote key injection request with IDTech RKI servers.

Parameters

<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.
--------------	--

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString

21.16.2.18 RETURN_CODE IDTechSDK.IDT_UniPay.device_updateFirmwareFromZip (byte[] *zipfile*, string *ident* = " ", bool *isForeground* = false)

Update Firmware From Zip

Performs one or more device firmware updates from firmware files passed as a compressed archive with a valid configuration file (.json format)

Parameters

<i>zipfile</i>	Zip file containing on or more firmware update files and a .json configuration file
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.
<i>isForeground</i>	If TRUE, will block program until firmware update complete. Otherwise, FALSE performs FW update on background.

Returns

RETURN_CODE: Values can be parsed with errorCode.getErrorString(string ident = "")

21.16.2.19 static int IDTechSDK.IDT_UniPay.getCommandTimeout (string *ident* = " ") [static]

Get Command Timeout

Gets the default timeout (in milliseconds) waiting for a blocking command response

Return values

<i>time</i>	Time
-------------	------

21.16.2.20 RETURN_CODE IDTechSDK.IDT_UniPay.icc_exchangeAPDU (string *c_APDU*, ref byte[] *response*, string *ident* = " ")

Exchange APDU

Sends an APDU packet to the ICC. If successful, response is returned in APDUResult class instance in response parameter.

Parameters

<i>c_APDU</i>	APDU data packet
<i>response</i>	Unencrypted/encrypted parsed APDU response
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with `errorCode.getErrorString(string ident = "")`

21.16.2.21 RETURN_CODE IDTechSDK.IDT_UniPay.icc_getAPDU_KSN (ref byte[] *k*sn, string *ident* = " ")

Get APDU KSN

Retrieves the KSN used in ICC Encrypted APDU usage

Parameters

<i>k</i> sn	Returns the encrypted APDU packet KSN
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with `errorCode.getErrorString(string ident = "")`

21.16.2.22 RETURN_CODE IDTechSDK.IDT_UniPay.icc_getICCReaderStatus (ref byte *status*, string *ident* = " ")

Get Reader Status

Returns the reader status

Parameters

<i>status</i>	Pointer that will return with the ICCReaderStatus results. bit 0: 0 = ICC Power Not Ready, 1 = ICC Powered bit 1: 0 = Card not seated, 1 = card seated
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with `errorCode.getErrorString(string ident = "")`

21.16.2.23 RETURN_CODE IDTechSDK.IDT_UniPay.icc_getKeyFormatForICCDUKPT (ref byte *format*, string *ident* = " ")

Get Key Format For ICC DUKPT

Specifies how data will be encrypted with Data Key or PIN key (if DUKPT key loaded, string *ident* = "")

Parameters

<i>format</i>	Response returned from method: <ul style="list-style-type: none"> • 'TDES': Encrypted card data with TDES if DUKPT Key had been loaded.(default, string <i>ident</i> = "") • 'AES': Encrypted card data with AES if DUKPT Key had been loaded. • 'NONE': No Encryption.
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with `errorCode.getErrorString(string ident = "")`

21.16.2.24 RETURN_CODE IDTechSDK.IDT_UniPay.icc_getKeyTypeForICCDUKPT (ref byte *type*, string *ident* = " ")

Get Key Type for ICC DUKPT

Specifies the key type used for ICC DUKPT encryption

Parameters

<i>type</i>	Response returned from method: <ul style="list-style-type: none"> 'DATA': Encrypted card data with Data Key DUKPT Key had been loaded.(default, string <i>ident</i> = "") 'PIN': Encrypted card data with PIN Key if DUKPT Key had been loaded.
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with `errorCode.getErrorString(string ident = "")`

21.16.2.25 RETURN_CODE IDTechSDK.IDT_UniPay.icc_powerOffICC (string *ident* = " ")

Power Off ICC

Powers down the ICC

Parameters

<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.
--------------	--

Returns

RETURN_CODE: Values can be parsed with `errorCode.getErrorString(string ident = "")`

If Success, empty If Failure, ASCII encoded data of error string

21.16.2.26 RETURN_CODE IDTechSDK.IDT_UniPay.icc_powerOnICC (ref byte[] *ATR*, string *ident* = " ")

Power On ICC

Power up the currently selected microprocessor card in the ICC reader

Parameters

<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.
--------------	--

Returns

RETURN_CODE: Values can be parsed with `errorCode.getErrorString(string ident = "")`

21.16.2.27 **RETURN_CODE** IDTechSDK.IDT_UniPay.icc_setKeyFormatForICCDUKPT (byte *encryption*, string *ident* = " ")

Set Key Format for ICC DUKPT

Sets how data will be encrypted, with either TDES or AES (if DUKPT key loaded, string *ident* = "")

Parameters

<i>encryption</i>	Encryption Type <ul style="list-style-type: none"> • 00: Encrypt with TDES • 01: Encrypt with AES
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with `errorCode.getErrorString(string ident = "")`

21.16.2.28 **RETURN_CODE** IDTechSDK.IDT_UniPay.icc_setKeyTypeForICCDUKPT (byte *encryption*, string *ident* = " ")

Set Key Type for ICC DUKPT Key

Sets which key the data will be encrypted with, with either Data Key or PIN key (if DUKPT key loaded, string *ident* = "")

Parameters

<i>encryption</i>	Encryption Type <ul style="list-style-type: none"> • 00: Encrypt with Data Key • 01: Encrypt with PIN Key
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with `errorCode.getErrorString(string ident = "")`

21.16.2.29 **RETURN_CODE** IDTechSDK.IDT_UniPay.msr_cancelMSRSwipe (string *ident* = " ")

Disable MSR Swipe Cancels MSR swipe request.

Parameters

<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.
--------------	--

Returns

RETURN_CODE: Values can be parsed with `errorCode.getErrorString(string ident = "")`

21.16.2.30 **RETURN_CODE** IDTechSDK.IDT_UniPay.msr_getClearPANID (ref byte *value*, string *ident* = " ")

Get Clear PAN Digits

Returns the number of digits that begin the PAN that will be in the clear

Parameters

<i>value</i>	Number of digits in clear. Values are char '0' - '6':
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with `errorCode.getErrorString(string ident = "")`

21.16.2.31 **RETURN_CODE** IDTechSDK.IDT_UniPay.msr_getExpirationMask (ref byte *value*, string *ident* = " ")

Get Expiration Masking

Get the flag that determines if to mask the expiration date

Parameters

<i>value</i>	'0' = masked, '1' = not-masked
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with `errorCode.getErrorString(string ident = "")`

21.16.2.32 **RETURN_CODE** IDTechSDK.IDT_UniPay.msr_getSetting (byte *setting*, ref byte *value*, string *ident* = " ")

Get MSR Setting value

Returns the encryption used for swipe data

Parameters

<i>setting</i>	MSR Setting to retrieve
<i>value</i>	MSR Setting value
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with `device_getResponseCodeString`

21.16.2.33 **RETURN_CODE** IDTechSDK.IDT_UniPay.msr_getSwipeEncryption (ref byte *encryption*, string *ident* = " ")

Get Swipe Data Encryption

Returns the encryption used for sweep data

Parameters

<i>encryption</i>	1 = TDES, 2 = AES, 0 = NONE
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with `errorCode.getErrorString(string ident = "")`

21.16.2.34 RETURN_CODE IDTechSDK.IDT_UniPay.msr_getSwipeForcedEncryptionOption (ref byte *option*, string *ident* = " ")

Get Swipe Data Encryption

Gets the swipe force encryption options

Parameters

<i>option</i>	Byte using lower four bits as flags. 0 = Force Encryption Off, 1 = Force Encryption On bit0 = Track 1 bit1 = Track 2 bit2 = Track 3 bit4 = Track 3 Card Option 0
---------------	--

Example: Response 0x03 = Track1/Track2 Forced Encryption, Track3/Track3-0 no Forced Encryption

Parameters

<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.
--------------	--

Returns

RETURN_CODE: Values can be parsed with `errorCode.getErrorString(string ident = "")`

21.16.2.35 RETURN_CODE IDTechSDK.IDT_UniPay.msr_getSwipeMaskOption (ref byte *option*, string *ident* = " ")

Get Swipe Mask Option

Gets the swipe mask/clear data sending option

Parameters

<i>option</i>	Byte using lower three bits as flags. 0 = Mask Option Off, 1 = Mask Option On bit0 = Track 1 bit1 = Track 2 bit2 = Track 3
---------------	--

Example: Response 0x03 = Track1/Track2 Masked Option ON, Track3 Masked Option Off

Parameters

<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.
--------------	--

Returns

RETURN_CODE: Values can be parsed with `errorCode.getErrorString(string ident = "")`

21.16.2.36 RETURN_CODE IDTechSDK.IDT_UniPay.msr_setClearPANID (byte *val*, string *ident* = " ")**Set Clear PAN Digits**

Sets the amount of digits shown in the clear (not masked) at the beginning of the returned PAN value

Parameters

<i>val</i>	Number of digits to show in clear. Range 0-6.
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with `errorCode.getErrorString(string ident = "")`

21.16.2.37 RETURN_CODE IDTechSDK.IDT_UniPay.msr_setExpirationMask (bool *mask*, string *ident* = " ")**Set Expiration Masking**

Sets the flag to mask the expiration date

Parameters

<i>mask</i>	TRUE = mask expiration
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with `errorCode.getErrorString(string ident = "")`

21.16.2.38 RETURN_CODE IDTechSDK.IDT_UniPay.msr_setSetting (byte *setting*, byte *value*, string *ident* = " ")**Set MSR Setting value****Parameters**

<i>setting</i>	MSR Setting to set
<i>value</i>	MSR Setting value
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with `device_getResponseCodeString`

21.16.2.39 RETURN_CODE IDTechSDK.IDT_UniPay.msr_setSwipeEncryption (byte *encryption*, string *ident* = " ")**Set Swipe Data Encryption**

Sets the swipe encryption method

Parameters

<i>encryption</i>	1 = TDES, 2 = AES
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with `errorCode.getErrorString(string ident = "")`

21.16.2.40 RETURN_CODE IDTechSDK.IDT_UniPay.msr_setSwipeForcedEncryptionOption (bool *track1*, bool *track2*, bool *track3*, bool *track3card0*, string *ident* = " ")

Set Swipe Force Encryption

Sets the swipe force encryption options

Parameters

<i>track1</i>	Force encrypt track 1
<i>track2</i>	Force encrypt track 2
<i>track3</i>	Force encrypt track 3
<i>track3card0</i>	Force encrypt track 3 when card type is 0
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with `errorCode.getErrorString(string ident = "")`

21.16.2.41 RETURN_CODE IDTechSDK.IDT_UniPay.msr_setSwipeMaskOption (bool *track1*, bool *track2*, bool *track3*, string *ident* = " ")

Set Swipe Mask Option

Sets the swipe mask/clear data sending option

Parameters

<i>track1</i>	Mask track 1 allowed
<i>track2</i>	Mask track 2 allowed
<i>track3</i>	Mask track 3 allowed
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with `errorCode.getErrorString(string ident = "")`

21.16.2.42 RETURN_CODE IDTechSDK.IDT_UniPay.msr_startMSRSwipe (int *timeout*, string *ident* = " ")

Enable MSR Swipe

Enables MSR, waiting for swipe to occur.

Parameters

<i>timeout</i>	Swipe Timeout Value
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with `errorCode.getErrorString(string ident = "")`

21.16.2.43 `static String IDTechSDK.IDT_UniPay.SDK_Version () [static]`

SDK Version

- All Devices

Returns the current version of SDK

Returns

Framework version

21.16.2.44 `static void IDTechSDK.IDT_UniPay.setCallback (Callback my_Callback) [static]`

Set Callback

Sets the class callback

21.16.2.45 `static void IDTechSDK.IDT_UniPay.setCallback (IntPtr my_Callback, SynchronizationContext context) [static]`

Set Callback

Sets the class callback

Parameters

<i>my_Callback</i>	The callback function to receive the response message from device. defined as follows. <code>public unsafe delegate void MFCCallBack(Parameters parameters, ident);</code>
<i>context</i>	The context of the UI thread

21.16.2.46 `static void IDTechSDK.IDT_UniPay.setCommandTimeout (int milliseconds, string ident = " ") [static]`

Set Command Timeout

Sets the default timeout (in milliseconds) waiting for a blocking command response

Parameters

<i>milliseconds</i>	Time
---------------------	------

21.16.3 Property Documentation

21.16.3.1 IDT_UniPay IDTechSDK.IDT_UniPay.SharedController [static], [get]

Singleton Instance

Establishes an singleton instance of [IDT_UniPay](#) class.

Returns

Instance of [IDT_UniPay](#)

The documentation for this class was generated from the following file:

- /Users/randy/Repo/universal-sdk/IDTechSDK_STD/DeviceHeaders/IDT_UniPay.cs

21.17 IDTechSDK.IDT_UniPayI_V Class Reference

Public Member Functions

- RETURN_CODE [config_getSerialNumber](#) (ref string response, string ident="")
- RETURN_CODE [device_getDRS](#) (ref byte[] codeDRS, string ident="")
- RETURN_CODE [device_sendPAE](#) (string command, ref string response, int timeout, string ident="")
- RETURN_CODE [device_getFirmwareVersion](#) (ref string response, string ident="")
- RETURN_CODE [device_selfCheck](#) (string ident="")
- RETURN_CODE [device_sendDataCommand](#) (string cmd, bool calcLRC, ref byte[] response, string ident="")
- RETURN_CODE [device_sendDataCommand_ext](#) (string cmd, bool calcLRC, ref byte[] response, int timeout, bool noResponse, string ident="")
- RETURN_CODE [msr_switchUSBInterfaceMode](#) (bool blsUSBKeyboardMode)
- RETURN_CODE [msr_switchUSBInterfaceMode](#) (bool blsUSBKeyboardMode, ref string ident)
- RETURN_CODE [msr_startMSRSwipe](#) (int timeout, string ident="")
- RETURN_CODE [msr_cancelMSRSwipe](#) (string ident="")
- RETURN_CODE [emv_cancelTransaction](#) (string ident="")
- RETURN_CODE [emv_startTransaction](#) (double amount, double amtOther, int exponent, int type, int timeout, byte[] tags, bool forceOnline, bool isFastEMV=false, string ident="")
- RETURN_CODE [emv_activateTransaction](#) (int timeout, byte[] tags, bool forceOnline, bool isFastEMV=false, string ident="")
- RETURN_CODE [device_startTransaction](#) (double amount, double amtOther, int exponent, int type, int timeout, byte[] tags, bool forceOnline, bool isFastEMV=false, string ident="")
- RETURN_CODE [device_activateTransaction](#) (int timeout, byte[] tags, bool forceOnline, bool isFastEMV=false, string ident="")
- RETURN_CODE [emv_authenticateTransaction](#) (byte[] updatedTLV, string ident="")
- RETURN_CODE [emv_completeTransaction](#) (bool commError, byte[] authCode, byte[] iad, byte[] tlvScripts, byte[] tlv, string ident="")
- RETURN_CODE [emv_callbackResponseLCD](#) (EMV_LCD_DISPLAY_MODE type, byte selection, string ident="")
- RETURN_CODE [emv_callbackResponsePIN](#) (EMV_PIN_MODE type, byte[] KSN, byte[] PIN, string ident="")
- RETURN_CODE [emv_callbackResponseMSR](#) (byte[] MSR, string ident="")
- RETURN_CODE [emv_getEMVKernelVersion](#) (ref string response, string ident="")
- RETURN_CODE [emv_retrieveTransactionResult](#) (byte[] tags, ref IDTTransactionData tlv, string ident="")
- RETURN_CODE [emv_removeApplicationData](#) (byte[] AID, string ident="")
- RETURN_CODE [emv_removeAllApplicationData](#) (string ident="")
- RETURN_CODE [emv_removeCAPK](#) (byte[] capk, string ident="")
- RETURN_CODE [emv_removeAllCAPK](#) (string ident="")
- RETURN_CODE [emv_removeCRL](#) (byte[] crlList, string ident="")

- RETURN_CODE [emv_removeAllCRL](#) (string ident="")
- RETURN_CODE [emv_retrieveApplicationData](#) (byte[] AID, ref byte[] tlv, string ident="")
- RETURN_CODE [emv_retrieveAIDList](#) (ref byte[][] response, string ident="")
- RETURN_CODE [emv_retrieveCAPK](#) (byte[] capk, ref byte[] key, string ident="")
- RETURN_CODE [emv_retrieveCAPKList](#) (ref byte[] keys, string ident="")
- RETURN_CODE [emv_retrieveCRLList](#) (ref byte[] list, string ident="")
- RETURN_CODE [emv_retrieveTerminalData](#) (ref byte[] tlv, string ident="")
- RETURN_CODE [emv_removeTerminalData](#) (string ident="")
- RETURN_CODE [emv_setApplicationData](#) (byte[] name, byte[] tlv, string ident="")
- RETURN_CODE [emv_setCAPK](#) (byte[] key, string ident="")
- RETURN_CODE [emv_setCRL](#) (byte[] list, string ident="")
- RETURN_CODE [emv_trySetTerminalData](#) (byte[] tlv, ref byte[] rejectedTLV, ref byte[] convertedTLV, bool overwrite=false, string ident="")
- RETURN_CODE [emv_setTerminalData](#) (byte[] tlv, string ident="")
- RETURN_CODE [emv_addTerminalData](#) (byte[] tlv, string ident="")
- RETURN_CODE [emv_setTerminalMajorConfiguration](#) (int configuration, string ident="")
- RETURN_CODE [device_pingDevice](#) (string ident="")
- RETURN_CODE [device_sendVivoCommandP2](#) (byte command, byte subCommand, byte[] data, ref byte[] response, string ident="")
- RETURN_CODE [device_sendVivoCommandP2_ext](#) (byte command, byte subCommand, byte[] data, ref byte[] response, int timeout, bool noResponse, string ident="")
- RETURN_CODE [device_startRKI](#) (string ident="")
- RETURN_CODE [device_setTerminalData](#) (byte[] tlv, string ident="")
- RETURN_CODE [device_retrieveTerminalData](#) (ref byte[] tlv, string ident="")
- RETURN_CODE [device_setBurstMode](#) (byte mode, string ident="")
- RETURN_CODE [device_setPollMode](#) (byte mode, string ident="")
- RETURN_CODE [device_updateDeviceFirmware](#) (byte[] firmwareData, string ident="")
- RETURN_CODE [icc_getICCRReaderStatus](#) (ref byte status, string ident="")
- RETURN_CODE [icc_powerOffICC](#) (string ident="")
- RETURN_CODE [icc_powerOnICC](#) (ref byte[] ATR, byte interfaces, string ident="")
- RETURN_CODE [icc_exchangeAPDU](#) (string c_APDU, ref byte[] response, string ident="")
- RETURN_CODE [device_enablePassThrough](#) (bool enablePassThrough, string ident="")
- RETURN_CODE [emv_getEMVKernelCheckValue](#) (ref string response, string ident="")
- RETURN_CODE [emv_getEMVConfigurationCheckValue](#) (ref string response, string ident="")
- string [device_getResponseCodeString](#) (RETURN_CODE eCode)
- RETURN_CODE [device_StartRKI](#) (int type, string ident="")
- RETURN_CODE [config_setEncryptionControl](#) (bool msr, bool icc, string ident="")
- RETURN_CODE [config_getEncryptionControl](#) (ref bool msr, ref bool icc, string ident="")
- RETURN_CODE [device_getRKIStatus](#) (bool isProd, bool isMultiKey, ref string status, ref Dictionary< string, RKI_KEY_TYPE > keys, string ident="")
- RETURN_CODE [device_getAnyRKIStatus](#) (bool isMultiKey, ref string status, ref Dictionary< string, RKI_KEY_TYPE > keys, string ident="")
- RETURN_CODE [emv_getEMVKernelVersionExt](#) (ref string response, string ident="")
- RETURN_CODE [device_updateFirmwareFromZip](#) (byte[] zipfile, string ident="", bool isForeground=false)
- RETURN_CODE [device_sendConfigurationFromZip](#) (byte[] zip, string filename, VC_OPERATION_TYPE type, bool matchFW, string ident="", bool isForeground=false)
- RETURN_CODE [device_sendConfiguration](#) (string filename, VC_OPERATION_TYPE type, bool matchFW, string ident="", bool isForeground=false)
- RETURN_CODE [device_readConfigurationToMemory](#) (string memo, Configs.InstallRules rules, List< Configs.DeviceCommand > cmds, string ident="", bool isForeground=false)
- RETURN_CODE [device_getConfigurationFromMemory](#) (ref string json, string ident="")
- RETURN_CODE [device_RemoteKeyInjection](#) (RKI_KEY_TYPE type, string keyName, string ident="", bool performOnForeground=false)

Static Public Member Functions

- static void **initSC** ()
- static int **getCommandTimeout** (string ident="")
- static void **setCommandTimeout** (int milliseconds, string ident="")
- static void **setCallback** (Callback my_Callback)
- static void **setCallback** (IntPtr my_Callback, SynchronizationContext context)
- static String **SDK_Version** ()
- static void **emv_autoAuthenticate** (bool authenticate, string ident="")
- static void **emv_autoAuthenticateTags** (bool authenticate, byte[] tags, string ident="")
- static void **emv_allowFallback** (bool allow, string ident="")
- static void **lcd_retrieveMessage** (DisplayMessages.DISPLAY_MESSAGE_LANGUAGE lang, DisplayMessages.DISPLAY_MESSAGE_IDENTIFIER id, ref string line1, ref string line2)

Properties

- static **IDT_UniPayI_V SharedController** [get]

21.17.1 Detailed Description

Class for UniPayI_V MSR/ICC reader

21.17.2 Member Function Documentation

21.17.2.1 **RETURN_CODE** IDTechSDK.IDT_UniPayI_V.config_getEncryptionControl (ref bool *msr*, ref bool *icc*, string *ident* = "")

Get Encryption Control

Get Encryption Control to switch status between MSR and ICC/EMV function. Following Encryption status supported:

- MSR ON, ICC/EMV ON,
- MSR ON, ICC/EMV OFF,
- MSR OFF, ICC/EMV OFF,

Parameters

<i>msr</i>	<ul style="list-style-type: none"> • true: enabled MSR with Encryption, • false: disabled MSR with Encryption,
<i>icc</i>	<ul style="list-style-type: none"> • true: enabled ICC with Encryption, • false: disabled ICC with Encryption,
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString

21.17.2.2 RETURN_CODE IDTechSDK.IDT_UniPayl_V.config_getSerialNumber (ref string *response*, string *ident* = " ")

Polls device for Serial Number

Parameters

<i>response</i>	Returns Serial Number
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString

21.17.2.3 RETURN_CODE IDTechSDK.IDT_UniPayl_V.config_setEncryptionControl (bool *msr*, bool *icc*, string *ident* = " ")

Set Encryption Control

Set Encryption Control to switch status between MSR and ICC/EMV function. Following Encryption status supported:

- MSR ON, ICC/EMV ON,
- MSR ON, ICC/EMV OFF,
- MSR OFF, ICC/EMV OFF,

Parameters

<i>msr</i>	<ul style="list-style-type: none"> • true: enable MSR with Encryption, • false: disable MSR with Encryption,
<i>icc</i>	<ul style="list-style-type: none"> • true: enable ICC with Encryption, • false: disable ICC with Encryption,
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString

21.17.2.4 RETURN_CODE IDTechSDK.IDT_UniPayl_V.device_activateTransaction (int *timeout*, byte[] *tags*, bool *forceOnline*, bool *isFastEMV* = false, string *ident* = " ")

Start EMV Transaction Request

Authorizes the EMV transaction for an ICC card

The tags will be returned in the callback routine.

Parameters

<i>timeout</i>	Timeout value in seconds.
<i>tags</i>	The tags to be included in the request. Passed as TLV Data. Example, tag 9F02 with amount 0x000000000100 would be 0x9F0206000000000100
<i>forceOnline</i>	TRUE = do not allow offline approval, FALSE = allow ICC to approve offline if terminal capable Note: To request tags to be included in default response, use tag DFEE1A, and specify tag list. Example four tags 9F02, 9F36, 95, 9F37 to be included in response = DFEE1A079F029F369f9F37
<i>isFastEMV</i>	If TRUE, it will populate the IDTTransactionData.fastEMV with ASCII data similar to IDTech FastEMV KB output, after performing an auto-authenticate and auto-complete with ResultCode = Could Not Contact Host
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString

21.17.2.5 RETURN_CODE IDTechSDK.IDT_UniPayl_V.device_enablePassThrough (bool *enablePassThrough*, string *ident* = " ")

Enable Pass Through

Enables Pass Through Mode for direct communication with L1 interface (power on icc, send apdu, etc).

Parameters

<i>enablePassThrough</i>	true = pass through ON, false = pass through OFF
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with errorCode.getErrorString(string ident = "")

21.17.2.6 RETURN_CODE IDTechSDK.IDT_UniPayl_V.device_getAnyRKIStatus (bool *isMultiKey*, ref string *status*, ref Dictionary< string, RKI_KEY_TYPE > *keys*, string *ident* = " ")

Get RKI Status Polls the RKI servers to validate device RKI eligibility. Note: if device type is known in advance (production or demo device), it is more efficient to use device_getRKIStatus instead

Parameters

<i>isMultiKey</i>	True = MultiKey, False = SingleKey
<i>status</i>	RKI status returned as a string
<i>keys</i>	Returns all available keys
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with errorCode.getErrorString(). When no data is available, return code = RETURN_CODE_NO_DATA_AVAILABLE

21.17.2.7 RETURN_CODE IDTechSDK.IDT_UniPayL_V.device_getConfigurationFromMemory (ref string json, string ident = " ")

Read Device Configuration From Memory Retrieves the ViVOconfig configuration data from memory after a successful device_readConfigurationToMemory

Parameters

<i>json</i>	Returns the .json file data that can be saved to disk
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with errorCode.getErrorString(). When no data is available, return code = RETURN_CODE_NO_DATA_AVAILABLE

21.17.2.8 RETURN_CODE IDTechSDK.IDT_UniPayL_V.device_getDRS (ref byte[] codeDRS, string ident = " ")

Get DRS Status

Gets the status of DRS(Destructive Reset). For TTK Version Only.

Parameters

<i>codeDRS</i>	the data format is [DRS SourceBlk Number] [SourceBlk1] ... [SourceBlkN] [DRS SourceBlk Number] is 2 bytes, format is NumL NumH. It is Number of [SourceBlkX] [SourceBlkX] is n bytes, Format is [SourceID] [SourceLen] [SourceData] [SourceID] is 1 byte [SourceLen] is 1 byte, it is length of [SourceData]
----------------	--

[SourceID] [SourceLen] [SourceData] 00 1 01 – Application Error 01 1 01 – Application Error 02 1 0x01 – EMV L2 Configuration Check Value Error 0x02 – Future Key Check Value Error 10 1 01 – Battery Error 11 1 Bit 0 – Tamper Switch 1 (0-No, 1-Error, string ident = "") Bit 1– Tamper Switch 2 (0-No, 1-Error, string ident = "") Bit 2– Tamper Switch 3 (0-No, 1-Error, string ident = "") Bit 3– Tamper Switch 4 (0-No, 1-Error, string ident = "") Bit 4– Tamper Switch 5 (0-No, 1-Error, string ident = "") Bit 5– Tamper Switch 6 (0-No, 1-Error, string ident = "")

12 1 01 –TemperatureHigh or Low 13 1 01 –Voltage High or Low 1F 4 Reg31~24bits, Reg23~16bits, Reg15~8bits, Reg7~0bits

Parameters

<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.
--------------	--

Returns

RETURN_CODE: Values can be parsed with errorCode.getErrorString(string ident = "")

21.17.2.9 RETURN_CODE IDTechSDK.IDT_UniPayL_V.device_getFirmwareVersion (ref string response, string ident = " ")

Polls device for Firmware Version

Parameters

<i>response</i>	Response returned of Firmware Version
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with `errorCode.getErrorString(string ident = "")`

21.17.2.10 `string IDTechSDK.IDT_UniPayl_V.device_getResponseCodeString (RETURN_CODE eCode)`

Get the description of response result.

Parameters

<i>eCode</i>	the response result.
--------------	----------------------

Return values

<i>the</i>	string for description of response result
------------	---

- case 0: "no error, beginning task";
- case 1: "no response from reader";
- case 2: "invalid response data";
- case 3: "time out for task or CMD";
- case 4: "wrong parameter";
- case 5: "SDK is doing MSR or ICC task";
- case 6: "SDK is doing PINPad task";
- case 7: "SDK is doing CTLS task";
- case 8: "SDK is doing EMV task";
- case 9: "SDK is doing Other task";
- case 10: "err response or data";
- case 11: "no reader attached";
- case 12: "mono audio is enabled";
- case 13: "did connection";
- case 14: "audio volume is too low";
- case 15: "task or CMD be canceled";
- case 16: "UF wrong string format";
- case 17: "UF file not found";
- case 18: "UF wrong file format";
- case 19: "Attempt to contact online host failed";
- case 20: "Attempt to perform RKI failed";
- case 0x300: "Key Type(TDES) of Session Key is not same as the related Master Key.";
- case 0x400: "Related Key was not loaded.";
- case 0x500: "Key Same.";

- case 0x501: "Key is all zero";
- case 0x502: "TR-31 format error";
- case 0x702: "PAN is Error Key.";
- case 0x705: "No Internal MSR PAN (or Internal MSR PAN is erased timeout)";
- case 0X0C01: "Incorrect Frame Tag";
- case 0X0C02: "Incorrect Frame Type";
- case 0X0C03: "Unknown Frame Type";
- case 0X0C04: "Unknown Command";
- case 0X0C05: "Unknown Sub-Command";
- case 0X0C06: "CRC Error";
- case 0X0C07: "Failed";
- case 0X0C08: "Timeout";
- case 0X0C0A: "Incorrect Parameter";
- case 0X0C0B: "Command Not Supported";
- case 0X0C0C: "Sub-Command Not Supported";
- case 0X0C0D: "Parameter Not Supported / Status Abort Command";
- case 0X0C0F: "Sub-Command Not Allowed";
- case 0X0D01: "Incorrect Header Tag";
- case 0X0D02: "Unknown Command";
- case 0X0D03: "Unknown Sub-Command";
- case 0X0D04: "CRC Error in Frame";
- case 0X0D05: "Incorrect Parameter";
- case 0X0D06: "Parameter Not Supported";
- case 0X0D07: "Mal-formatted Data";
- case 0X0D08: "Timeout";
- case 0X0D0A: "Failed / NACK";
- case 0X0D0B: "Command not Allowed";
- case 0X0D0C: "Sub-Command not Allowed";
- case 0X0D0D: "Buffer Overflow (Data Length too large for reader buffer)";
- case 0X0D0E: "User Interface Event";
- case 0X0D11: "Communication type not supported, VT-1, burst, etc.";
- case 0X0D12: "Secure interface is not functional or is in an intermediate state.";
- case 0X0D13: "Data field is not mod 8";
- case 0X0D14: "Pad 0x80 not found where expected";
- case 0X0D15: "Specified key type is invalid";
- case 0X0D1: "Could not retrieve key from the SAM(InitSecureComm)";

- case 0X0D17: "Hash code problem";
- case 0X0D18: "Could not store the key into the SAM(InstallKey)";
- case 0X0D19: "Frame is too large";
- case 0X0D1A: "Unit powered up in authentication state but POS must resend the InitSecureComm command";
- case 0X0D1B: "The EEPROM may not be initialized because SecCommInterface does not make sense";
- case 0X0D1C: "Problem encoding APDU";
- case 0X0D20: "Unsupported Index(ILM) SAM Transceiver error – problem communicating with the SAM(Key Mgr)";
- case 0X0D2: "Unexpected Sequence Counter in multiple frames for single bitmap(ILM) Length error in data returned from the SAM(Key Mgr)";
- case 0X0D22: "Improper bit map(ILM)";
- case 0X0D23: "Request Online Authorization";
- case 0X0D24: "ViVOCard3 raw data read successful";
- case 0X0D25: "Message index not available(ILM) ViVOcomm activate transaction card type(ViVOcomm)";
- case 0X0D26: "Version Information Mismatch(ILM)";
- case 0X0D27: "Not sending commands in correct index message index(ILM)";
- case 0X0D28: "Time out or next expected message not received(ILM)";
- case 0X0D29: "ILM languages not available for viewing(ILM)";
- case 0X0D2A: "Other language not supported(ILM)";
- case 0X0D41: "Unknown Error from SAM";
- case 0X0D42: "Invalid data detected by SAM";
- case 0X0D43: "Incomplete data detected by SAM";
- case 0X0D44: "Reserved";
- case 0X0D45: "Invalid key hash algorithm";
- case 0X0D46: "Invalid key encryption algorithm";
- case 0X0D47: "Invalid modulus length";
- case 0X0D48: "Invalid exponent";
- case 0X0D49: "Key already exists";
- case 0X0D4A: "No space for new RID";
- case 0X0D4B: "Key not found";
- case 0X0D4C: "Crypto not responding";
- case 0X0D4D: "Crypto communication error";
- case 0X0D4E: "Module-specific error for Key Manager";
- case 0X0D4F: "All key slots are full (maximum number of keys has been installed)";
- case 0X0D50: "Auto-Switch OK";
- case 0X0D51: "Auto-Switch failed";

- case 0X0D90: "Account DUKPT Key not exist";
- case 0X0D91: "Account DUKPT Key KSN exhausted";
- case 0x0D00: "This Key had been loaded.";
- case 0x0E00: "Base Time was loaded.";
- case 0x0F00: "Encryption Or Decryption Failed.";
- case 0x1000: "Battery Low Warning (It is High Priority Response while Battery is Low.)";
- case 0x1800: "Send "Cancel Command" after send "Get Encrypted PIN" & "Get Numeric" & "Get Amount";
- case 0x1900: "Press "Cancel" key after send "Get Encrypted PIN" & "Get Numeric" & "Get Amount";
- case 0x30FF: "Security Chip is not connect";
- case 0x3000: "Security Chip is deactivation & Device is In Removal Legally State.";
- case 0x3101: "Security Chip is activation & Device is In Removal Legally State.";
- case 0x5500: "No Admin DUKPT Key.";
- case 0x5501: "Admin DUKPT Key STOP.";
- case 0x5502: "Admin DUKPT Key KSN is Error.";
- case 0x5503: "Get Authentication Code1 Failed.";
- case 0x5504: "Validate Authentication Code Error.";
- case 0x5505: "Encrypt or Decrypt data failed.";
- case 0x5506: "Not Support the New Key Type.";
- case 0x5507: "New Key Index is Error.";
- case 0x5508: "Step Error.";
- case 0x5509: "KSN Error";
- case 0x550A: "MAC Error.";
- case 0x550B: "Key Usage Error.";
- case 0x550C: "Mode Of Use Error.";
- case 0x550F: "Other Error.";
- case 0x6000: "Save or Config Failed / Or Read Config Error.";
- case 0x6200: "No Serial Number.";
- case 0x6900: "Invalid Command - Protocol is right, but task ID is invalid.";
- case 0x6A01: "Unsupported Command – Protocol and task ID are right, but command is invalid – In this State";
- case 0x6A00: "Unsupported Command - Protocol and task ID are right, but command is invalid.";
- case 0x6B00: "Unknown parameter in command - Protocol task ID and command are right, but parameter is invalid.";
- case 0x6C00: "Unknown parameter in command – Protocol task ID and command are right, but length is out of the requirement.";
- case 0x7200: "Device is suspend (MKSK suspend or press password suspend).";
- case 0x7300: "PIN DUKPT is STOP (21 bit 1).";

- case 0x7400: "Device is Busy.";
- case 0xE100: "Can not enter sleep mode";
- case 0xE200: "File has existed";
- case 0xE300: "File has not existed";
- case 0xE313: "IO line low -- Card error after session start";
- case 0xE400: "Open File Error";
- case 0xE500: "SmartCard Error";
- case 0xE600: "Get MSR Card data is error";
- case 0xE700: "Command time out";
- case 0xE800: "File read or write is error";
- case 0xE900: "Active 1850 error!";
- case 0xEA00: "Load bootloader error";
- case 0xEF00: "Protocol Error- STX or ETX or check error.";
- case 0xEB00: "Picture is not exist";
- case 0x2C02: "No Microprocessor ICC seated";
- case 0x2C06: "no card seated to request ATR";
- case 0x2D01: "Card Not Supported,";
- case 0x2D03: "Card Not Supported, wants CRC";
- case 0x690D: "Command not supported on reader without ICC support";
- case 0x8100: "ICC error time out on power-up";
- case 0x8200: "invalid TS character received - Wrong operation step";
- case 0x8300: "Decode MSR Error";
- case 0x8400: "TriMagII no Response";
- case 0x8500: "No Swipe MSR Card";
- case 0x8510: "No Financial Card";
- case 0x8600: "Unsupported F, D, or combination of F and D";
- case 0x8700: "protocol not supported EMV TD1 out of range";
- case 0x8800: "power not at proper level";
- case 0x8900: "ATR length too long";
- case 0x8B01: "EMV invalid TA1 byte value";
- case 0x8B02: "EMV TB1 required";
- case 0x8B03: "EMV Unsupported TB1 only 00 allowed";
- case 0x8B04: "EMV Card Error, invalid BWI or CWI";
- case 0x8B06: "EMV TB2 not allowed in ATR";
- case 0x8B07: "EMV TC2 out of range";
- case 0x8B08: "EMV TC2 out of range";

- case 0x8B09: "per EMV96 TA3 must be > 0xF";
- case 0x8B10: "ICC error on power-up";
- case 0x8B11: "EMV T=1 then TB3 required";
- case 0x8B12: "Card Error, invalid BWI or CWI";
- case 0x8B13: "Card Error, invalid BWI or CWI";
- case 0x8B17: "EMV TC1/TB3 conflict-";
- case 0x8B20: "EMV TD2 out of range must be T=1";
- case 0x8C00: "TCK error";
- case 0xA304: "connector has no voltage setting";
- case 0xA305: "ICC error on power-up invalid (SBLK(IFSD) exchange";
- case 0xE301: "ICC error after session start";
- case 0xFF00: "Request to go online";
- case 0xFF01: "EMV: Accept the offline transaction";
- case 0xFF02: "EMV: Decline the offline transaction";
- case 0xFF03: "EMV: Accept the online transaction";
- case 0xFF04: "EMV: Decline the online transaction";
- case 0xFF05: "EMV: Application may fallback to magstripe technology";
- case 0xFF06: "EMV: ICC detected tah the conditions of use are not satisfied";
- case 0xFF07: "EMV: ICC didn't accept transaction";
- case 0xFF08: "EMV: Transaction was cancelled";
- case 0xFF09: "EMV: Application was not selected by kernel or ICC format error or ICC missing data error";
- case 0xFF0A: "EMV: Transaction is terminated";
- case 0xFF0B: "EMV: Other EMV Error";
- case 0xFFFF: "NO RESPONSE";
- case 0xF002: "ICC communication timeout";
- case 0xF003: "ICC communication Error";
- case 0xF00F: "ICC Card Seated and Highest Priority, disable MSR work request";
- case 0xF200: "AID List / Application Data is not exist";
- case 0xF201: "Terminal Data is not exist";
- case 0xF202: "TLV format is error";
- case 0xF203: "AID List is full";
- case 0xF204: "Any CA Key is not exist";
- case 0xF205: "CA Key RID is not exist";
- case 0xF206: "CA Key Index it not exist";
- case 0xF207: "CA Key is full";
- case 0xF208: "CA Key Hash Value is Error";

- case 0xF209: "Transaction format error";
- case 0xF20A: "The command will not be processing";
- case 0xF20B: "CRL is not exist";
- case 0xF20C: "CRL number exceed max number";
- case 0xF20D: "Amount,Other Amount,Trasaction Type are missing";
- case 0xF20E: "The Identification of algorithm is mistake";
- case 0xF20F: "No Financial Card";
- case 0xF210: "In Encrypt Result state, TLV total Length is greater than Max Length";
- case 0x1001: "INVALID ARG";
- case 0x1002: "FILE_OPEN_FAILED";
- case 0x1003: "FILE OPERATION_FAILED";
- case 0x2001: "MEMORY_NOT_ENOUGH";
- case 0x3002: "SMARTCARD_FAIL";
- case 0x3003: "SMARTCARD_INIT_FAILED";
- case 0x3004: "FALLBACK_SITUATION";
- case 0x3005: "SMARTCARD_ABSENT";
- case 0x3006: "SMARTCARD_TIMEOUT";
- case 0x5001: "EMV_PARSING_TAGS_FAILED";
- case 0x5002: "EMV_DUPLICATE_CARD_DATA_ELEMENT";
- case 0x5003: "EMV_DATA_FORMAT_INCORRECT";
- case 0x5004: "EMV_NO_TERM_APP";
- case 0x5005: "EMV_NO_MATCHING_APP";
- case 0x5006: "EMV_MISSING_MANDATORY_OBJECT";
- case 0x5007: "EMV_APP_SELECTION_RETRY";
- case 0x5008: "EMV_GET_AMOUNT_ERROR";
- case 0x5009: "EMV_CARD_REJECTED";
- case 0x5010: "EMV_AIP_NOT_RECEIVED";
- case 0x5011: "EMV_AFL_NOT_RECEIVED";
- case 0x5012: "EMV_AFL_LEN_OUT_OF_RANGE";
- case 0x5013: "EMV_SFI_OUT_OF_RANGE";
- case 0x5014: "EMV_AFL_INCORRECT";
- case 0x5015: "EMV_EXP_DATE_INCORRECT";
- case 0x5016: "EMV_EFF_DATE_INCORRECT";
- case 0x5017: "EMV_ISS_COD_TBL_OUT_OF_RANGE";
- case 0x5018: "EMV_CRYPTOGAM_TYPE_INCORRECT";
- case 0x5019: "EMV_PSE_NOT_SUPPORTED_BY_CARD";

- case 0x5020: "EMV_USER_SELECTED_LANGUAGE";
- case 0x5021: "EMV_SERVICE_NOT_ALLOWED";
- case 0x5022: "EMV_NO_TAG_FOUND";
- case 0x5023: "EMV_CARD_BLOCKED";
- case 0x5024: "EMV_LEN_INCORRECT";
- case 0x5025: "CARD_COM_ERROR";
- case 0x5026: "EMV_TSC_NOT_INCREASED";
- case 0x5027: "EMV_HASH_INCORRECT";
- case 0x5028: "EMV_NO_ARC";
- case 0x5029: "EMV_INVALID_ARC";
- case 0x5030: "EMV_NO_ONLINE_COMM";
- case 0x5031: "TRAN_TYPE_INCORRECT";
- case 0x5032: "EMV_APP_NO_SUPPORT";
- case 0x5033: "EMV_APP_NOT_SELECT";
- case 0x5034: "EMV_LANG_NOT_SELECT";
- case 0x5035: "EMV_NO_TERM_DATA";
- case 0x6001: "CVM_TYPE_UNKNOWN";
- case 0x6002: "CVM_AIP_NOT_SUPPORTED";
- case 0x6003: "CVM_TAG_8E_MISSING";
- case 0x6004: "CVM_TAG_8E_FORMAT_ERROR";
- case 0x6005: "CVM_CODE_IS_NOT_SUPPORTED";
- case 0x6006: "CVM_COND_CODE_IS_NOT_SUPPORTED";
- case 0x6007: "NO_MORE_CVM";
- case 0x6008: "PIN_BYPASSED_BEFORE";
- case 0x7001: "PK_BUFFER_SIZE_TOO_BIG";
- case 0x7002: "PK_FILE_WRITE_ERROR";
- case 0x7003: "PK_HASH_ERROR";
- case 0x8001: "NO_CARD_HOLDER_CONFIRMATION";
- case 0x8002: "GET_ONLINE_PIN";
- case 0xD000: "Data not exist";
- case 0xD001: "Data access error";
- case 0xD100: "RID not exist";
- case 0xD101: "RID existed";
- case 0xD102: "Index not exist";
- case 0xD200: "Maximum exceeded";
- case 0xD201: "Hash error";

- case 0xD205: "System Busy";
- case 0x0E01: "Unable to go online";
- case 0x0E02: "Technical Issue";
- case 0x0E03: "Declined";
- case 0x0E04: "Issuer Referral transaction";
- case 0x0F01: "Decline the online transaction";
- case 0x0F02: "Request to go online";
- case 0x0F03: "Transaction is terminated";
- case 0x0F05: "Application was not selected by kernel or ICC format error or ICC missing data error";
- case 0x0F07: "ICC didn't accept transaction";
- case 0x0F0A: "Application may fallback to magstripe technology";
- case 0x0F0C: "Transaction was cancelled";
- case 0x0F0D: "Timeout";
- case 0x0F0F: "Other EMV Error";
- case 0x0F10: "Accept the offline transaction";
- case 0x0F11: "Decline the offline transaction";
- case 0x0F21: "ICC detected tah the conditions of use are not satisfied";
- case 0x0F22: "No app were found on card matching terminal configuration";
- case 0x0F23: "Terminal file does not exist";
- case 0x0F24: "CAPK file does not exist";
- case 0x0F25: "CRL Entry does not exist";
- case 0x0FFE: "code when blocking is disabled";
- case 0x0FFF: "code when command is not applicable on the selected device";
- case 0xF005: "ICC Encrypted C-APDU Data Structure Length Error Or Format Error.";
- case 0xBBE0: "CM100 Success";
- case 0xBBE1: "CM100 Parameter Error";
- case 0xBBE2: "CM100 Low Output Buffer";
- case 0xBBE3: "CM100 Card Not Found";
- case 0xBBE4: "CM100 Collision Card Exists";
- case 0xBBE5: "CM100 Too Many Cards Exist";
- case 0xBBE6: "CM100 Saved Data Does Not Exist";
- case 0xBBE8: "CM100 No Data Available";
- case 0xBBE9: "CM100 Invalid CID Returned";
- case 0xBBEA: "CM100 Invalid Card Exists";
- case 0xBBEC: "CM100 Command Unsupported";
- case 0xBBED: "CM100 Error In Command Process";

- case 0xBBEE: "CM100 Invalid Command";
- case 0X9031: "Unknown command";
- case 0X9032: "Wrong parameter (such as the length of the command is incorrect)";
- case 0X9038: "Wait (the command couldn't be finished in BWT)";
- case 0X9039: "Busy (a previously command has not been finished)";
- case 0X903A: "Number of retries over limit";
- case 0X9040: "Invalid Manufacturing system data";
- case 0X9041: "Not authenticated";
- case 0X9042: "Invalid Master DUKPT Key";
- case 0X9043: "Invalid MAC Key";
- case 0X9044: "Reserved for future use";
- case 0X9045: "Reserved for future use";
- case 0X9046: "Invalid DATA DUKPT Key";
- case 0X9047: "Invalid PIN Pairing DUKPT Key";
- case 0X9048: "Invalid DATA Pairing DUKPT Key";
- case 0X9049: "No nonce generated";
- case 0X9949: "No GUID available. Perform getVersion first.";
- case 0X9950: "MAC Calculation unsuccessful. Check BDK value.";
- case 0X904A: "Not ready";
- case 0X904B: "Not MAC data";
- case 0X9050: "Invalid Certificate";
- case 0X9051: "Duplicate key detected";
- case 0X9052: "AT checks failed";
- case 0X9053: "TR34 checks failed";
- case 0X9054: "TR31 checks failed";
- case 0X9055: "MAC checks failed";
- case 0X9056: "Firmware download failed";
- case 0X9060: "Log is full";
- case 0X9061: "Removal sensor unengaged";
- case 0X9062: "Any hardware problems";
- case 0X9070: "ICC communication timeout";
- case 0X9071: "ICC data error (such check sum error)";
- case 0X9072: "Smart Card not powered up";

21.17.2.11 **RETURN_CODE** IDTechSDK.IDT_UniPayL_V.device_getRKIStatus (*bool isProd*, *bool isMultiKey*, *ref string status*,
ref Dictionary< string, RKI_KEY_TYPE > keys, *string ident = " "*)

Get RKI Status From Specified Server Polls the RKI server to validate device RKI eligibility

Parameters

<i>isProd</i>	TRUE = poll production RKI server, FALSE = poll demo RKI server
<i>isMultiKey</i>	True = MultiKey, False = SingleKey
<i>status</i>	RKI status returned as a string
<i>keys</i>	Returns dictionary of available keys
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with `errorCode.getErrorString()`. When no data is available, return code = RETURN_CODE_NO_DATA_AVAILABLE

21.17.2.12 RETURN_CODE IDTechSDK.IDT_UniPayl_V.device_pingDevice (string *ident* = " ")

Ping Device

Pings the reader. If connected, returns success. Otherwise, returns timeout.

Parameters

<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.
--------------	--

Returns

RETURN_CODE: Values can be parsed with `errorCode.getErrorString(string ident = "")`

21.17.2.13 RETURN_CODE IDTechSDK.IDT_UniPayl_V.device_readConfigurationToMemory (string *memo*, Configs.InstallRules *rules*, List< Configs.DeviceCommand > *cmds*, string *ident* = " ", bool *isForeground* = false)

Read Device Configuration To Memory Executes a ViVOconfig read of device settings and stores the results in memory

Once a ViVOconfig read starts, all messages will be returned to the callback as `DeviceState.ViVOconfig`. The ViV↔Oconfig operation is complete when the return code from a `DeviceState.ViVOconfig` message is one of the following values:

- RETURN_CODE_SDK_VIVOCONFIG_READ_CONFIG_SUCCESS = Read process completed successfully. Execute `device_getConfigurationFromMemory` to retrieve the configuration data
- RETURN_CODE_SDK_VIVOCONFIG_READ_CONFIG_FAILED = Read process FAILED.

Parameters

<i>memo</i>	Optional: populates the memo field of the captured .json file
<i>rules</i>	Optional: populates the .json file with available install rules
<i>cmds</i>	Optional: populates the .json file with commands to execute
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.
<i>isForeground</i>	If TRUE, will block program until update complete. Otherwise, FALSE performs update on background.

Returns

RETURN_CODE: Values can be parsed with `errorCode.getErrorString()`. When no data is available, return code = RETURN_CODE_NO_DATA_AVAILABLE

21.17.2.14 RETURN_CODE IDTechSDK.IDT_UniPayl_V.device_RemoteKeyInjection (RKI_KEY_TYPE *type*, string *keyName*, string *ident* = " ", bool *performOnForeground* = false)

Remote Key Injection Performs a remote key injection for the device

Parameters

<i>type</i>	Remote Key Injection Type
<i>keyName</i>	Name of key (optional)
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.
<i>performOnForeground</i>	TRUE = block during process (default is false)

Returns

RETURN_CODE: Values can be parsed with `errorCode.getErrorString()`. When no data is available, return code = RETURN_CODE_NO_DATA_AVAILABLE

21.17.2.15 RETURN_CODE IDTechSDK.IDT_UniPayl_V.device_retrieveTerminalData (ref byte[] *tlv*, string *ident* = " ")

Retrieve Terminal Data

Retrieves the Terminal Data for CTLS. This is configuration group 0 (Tag FFEE - > FFEE0100). The terminal data can also be retrieved by `ctls_getConfiguraitonGroup(0)`.

Parameters

<i>tlv</i>	Response returned as a TLV
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with `errorCode.getErrorString()`

21.17.2.16 RETURN_CODE IDTechSDK.IDT_UniPayl_V.device_selfCheck (string *ident* = " ")

Self check for TTK If Self-Test function Failed, then work into De-activation State. If device work into De-activation State, All Sensitive Data will be erased and it need be fixed in Manufacture.

Parameters

<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.
--------------	--

Returns

RETURN_CODE: Values can be parsed with `device_getResponseCodeString`

21.17.2.17 **RETURN_CODE** IDTechSDK.IDT_UniPayl_V.device_sendConfiguration (*string filename*, *VC_OPERATION_TYPE type*, *bool matchFW*, *string ident* = " ", *bool isForeground* = false)

Send Configuration Executes a ViVOconfig update or verify to a device

Once a ViVOconfig write or verify starts, all messages will be returned to the callback as DeviceState.ViVOconfig. The ViVOconfig operation is complete when the return code from a DeviceState.ViVOconfig message is one of the following values:

- RETURN_CODE_SDK_VIVOCONFIG_VERIFY_SUCCESS = Verification process completed successfully. No differences found.
- RETURN_CODE_SDK_VIVOCONFIG_VERIFY_SUCCESS_WITH_WARNING = Verification process completed with warnings.
- RETURN_CODE_SDK_VIVOCONFIG_VERIFY_FAILED = Verification process FAILED
- RETURN_CODE_SDK_VIVOCONFIG_WRITE_SUCCESS = Write process completed successfully.
- RETURN_CODE_SDK_VIVOCONFIG_WRITE_SUCCESS_WITH_WARNING = Write process completed with warnings
- RETURN_CODE_SDK_VIVOCONFIG_WRITE_FAILED = Write process FAILED

Parameters

<i>filename</i>	The .json configuration file to update the device with NOTE: You can also use this parameter to pass the path of where the .json file is located at on local storage instead
<i>filename</i>	The name of the .json configuration file
<i>type</i>	VC_OPERATION_TYPE

- VC_OPERATION_TYPE_WRITE = Write File To Device, Hash must validate
- VC_OPERATION_TYPE_VERIFY = Verify Device With File
- VC_OPERATION_TYPE_WRITE_IGNORE_HASH = Write File To Device, Ignore Hash
- VC_OPERATION_TYPE_WRITE_FIX_HASH = Write File To Device, Fix Hash If Necessary

Parameters

<i>matchFW</i>	TRUE = Device FW must match file FW, FALSE = Don't validate device FW against file
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.
<i>isForeground</i>	If TRUE, will block program until update complete. Otherwise, FALSE performs update on background.

Returns

RETURN_CODE: Values can be parsed with `errorCode.getErrorString()`. When no data is available, return code = RETURN_CODE_NO_DATA_AVAILABLE

21.17.2.18 **RETURN_CODE** IDTechSDK.IDT_UniPayl_V.device_sendConfigurationFromZip (*byte[] zip*, *string filename*, *VC_OPERATION_TYPE type*, *bool matchFW*, *string ident* = " ", *bool isForeground* = false)

Send Configuration From Zip Executes a ViVOconfig update or verify to a device using files from a .zip archive

Once a ViVOconfig write or update starts, all messages will be returned to the callback as DeviceState.ViVOconfig. The ViVOconfig operation is complete when the return code from a DeviceState.ViVOconfig message is one of the following values:

- `RETURN_CODE_SDK_VIVOCONFIG_VERIFY_SUCCESS` = Verification process completed successfully. No differences found.
- `RETURN_CODE_SDK_VIVOCONFIG_VERIFY_SUCCESS_WITH_WARNING` = Verification process completed with warnings.
- `RETURN_CODE_SDK_VIVOCONFIG_VERIFY_FAILED` = Verification process FAILED
- `RETURN_CODE_SDK_VIVOCONFIG_WRITE_SUCCESS` = Write process completed successfully.
- `RETURN_CODE_SDK_VIVOCONFIG_WRITE_SUCCESS_WITH_WARNING` = Write process completed with warnings
- `RETURN_CODE_SDK_VIVOCONFIG_WRITE_FAILED` = Write process FAILED

Parameters

<i>zip</i>	.Zip archive containing the .json configuration file and also any associated image assets NOTE: You can also use this parameter to pass the path of where the .zip file is located at on local storage instead
<i>filename</i>	The name of the .json configuration file
<i>type</i>	<code>VC_OPERATION_TYPE</code>

- `VC_OPERATION_TYPE_WRITE` = Write File To Device, Hash must validate
- `VC_OPERATION_TYPE_VERIFY` = Verify Device With File
- `VC_OPERATION_TYPE_WRITE_IGNORE_HASH` = Write File To Device, Ignore Hash
- `VC_OPERATION_TYPE_WRITE_FIX_HASH` = Write File To Device, Fix Hash If Necessary

Parameters

<i>matchFW</i>	<code>TRUE</code> = Device FW must match file FW, <code>FALSE</code> = Don't validate device FW against file
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.
<i>isForeground</i>	If <code>TRUE</code> , will block program until update complete. Otherwise, <code>FALSE</code> performs update on background.

Returns

`RETURN_CODE`: Values can be parsed with `errorCode.getErrorString()`. When no data is available, return code = `RETURN_CODE_NO_DATA_AVAILABLE`

21.17.2.19 `RETURN_CODE IDTechSDK.IDT_UniPayl_V.device_sendDataCommand (string cmd, bool calcLRC, ref byte[] response, string ident = " ")`

Send a data command to the device

Sends a command to the device.

Parameters

<i>cmd</i>	String representation of command to execute
<i>calcLRC</i>	If <code>TRUE</code> , this will wrap command with start/length/lrc/sum/end: '{STX}{Len_Low}{Len_High} data {CheckLRC} {CheckSUM} {ETX}'
<i>response</i>	Response data
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString

21.17.2.20 RETURN_CODE IDTechSDK.IDT_UniPayl_V.device_sendDataCommand_ext (string *cmd*, bool *calcLRC*, ref byte[] *response*, int *timeout*, bool *noResponse*, string *ident* = " ")

Send a data command to the device - extended

Sends a command to the device.

Parameters

<i>cmd</i>	String representation of command to execute
<i>calcLRC</i>	If TRUE, this will wrap command as NGA with start/length/lrc/sum/end: '{STX}{Len_Low}{Len_High} data {CheckLRC} {CheckSUM} {ETX}'
<i>response</i>	Response data
<i>timeout</i>	Timeout value waiting for response, in milliseconds (1000 = 1 second, string ident = "")
<i>noResponse</i>	if TRUE, this will not wait for a response and immediately return SUCCESS
<i>calcITP</i>	If TRUE, this will wrap command as ITP with start/end/lrc: '{STX} data {ETX}{CheckLRC}'
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString

21.17.2.21 RETURN_CODE IDTechSDK.IDT_UniPayl_V.device_sendPAE (string *command*, ref string *response*, int *timeout*, string *ident* = " ")

Send Payment Application Engine Command

Executes a PAE command

Parameters

<i>command</i>	ASCII command string, should start with "*PAE"
<i>response</i>	command response
<i>timeout</i>	timeout waiting for PAE response
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with errorCode.getErrorString(string ident = "")

21.17.2.22 RETURN_CODE IDTechSDK.IDT_UniPayl_V.device_sendVivoCommandP2 (byte *command*, byte *subCommand*, byte[] *data*, ref byte[] *response*, string *ident* = " ")

Send Vivo Command Protocol 2

Sends a protocol 2 command to Vivo readers (IDG/NEO, string ident = "")

Parameters

<i>command</i>	Command
<i>subCommand</i>	Sub-Command
<i>data</i>	Data. May be null
<i>response</i>	Response
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with `errorCode.getErrorString(string ident = "")`

21.17.2.23 RETURN_CODE IDTechSDK.IDT_UniPayl_V.device_sendVivoCommandP2_ext (byte *command*, byte *subCommand*, byte[] *data*, ref byte[] *response*, int *timeout*, bool *noResponse*, string *ident* = " ")

Send Vivo Command Protocol 2 Extended

Sends a protocol 2 command to Vivo readers (IDG/NEO, string *ident* = "")

Parameters

<i>command</i>	Command
<i>subCommand</i>	Sub-Command
<i>data</i>	Data. May be null
<i>response</i>	Response
<i>timeout</i>	Timeout, in milliseconds (3000 = 3 seconds, string <i>ident</i> = "")
<i>noResponse</i>	TRUE = don't wait for response, FALSE = wait for response defined by timeout
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with `errorCode.getErrorString(string ident = "")`

21.17.2.24 RETURN_CODE IDTechSDK.IDT_UniPayl_V.device_setBurstMode (byte *mode*, string *ident* = " ")

Send Burst Mode

Sets the burst mode for the device.

Parameters

<i>mode</i>	0 = OFF, 1 = Always On, 2 = Auto Exit
-------------	---------------------------------------

NOTE: USB-KB mode is identical to USB-HID mode except for the following:

- The device will appear to Windows as a Keyboard device instead of a USB-HID Device
- If the device is in Burst Mode, and the device is also in Auto Poll mode, swiped data will be output to the keyboard buffer instead of the SDK.

Parameters

<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.
--------------	--

Returns

RETURN_CODE: Values can be parsed with `errorCode.getErrorString(string ident = "")`

21.17.2.25 RETURN_CODE IDTechSDK.IDT_UniPayl_V.device_setPollMode (byte mode, string ident = " ")

Send Poll Mode

Sets the poll mode for the device. Auto Poll keeps reader active, Poll On Demand only polls when requested by terminal

Parameters

<i>mode</i>	0 = Auto Poll, 1 = Poll On Demand
-------------	-----------------------------------

NOTE: USB-KB mode is identical to USB-HID mode except for the following:

- The device will appear to Windows as a Keyboard device instead of a USB-HID Device
- If the device is in Burst Mode, and the device is also in Auto Poll mode, swiped data will be output to the keyboard buffer instead of the SDK.

Parameters

<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.
--------------	--

Returns

RETURN_CODE: Values can be parsed with `errorCode.getErrorString(string ident = "")`

21.17.2.26 RETURN_CODE IDTechSDK.IDT_UniPayl_V.device_setTerminalData (byte[] tlv, string ident = " ")

Set Terminal Data

Sets the Terminal Data for CTLS as specified by the TLV. The first TLV must be Configuration Group Number (Tag FFE4). The terminal global data is group 0, so the first TLV would be FFE40100. Other groups can be defined using this method (1 or greater), and those can be retrieved with `emv_getConfigurationGroup(int group)`, and deleted with `emv_removeConfigurationGroup(int group)`. You cannot delete group 0.

Parameters

<i>tlv</i>	TerminalData configuration data
------------	---------------------------------

Return values

<i>RETURN_CODE</i>	Return codes listed as typedef enum in IDTCommon:RETURN_CODE. Values can be parsed with <code>IDT_BTPay::device_getResponseCodeString:(string ident = "")</code>
--------------------	--

21.17.2.27 **RETURN_CODE** IDTechSDK.IDT_UniPayl_V.device_startRKI (string *ident* = " ")

Start Remote Key Injection

Starts a remote key injection request with IDTech RKI servers.

Parameters

<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.
--------------	--

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString

21.17.2.28 **RETURN_CODE** IDTechSDK.IDT_UniPayl_V.device_StartRKI (int *type*, string *ident* = " ")

Start Remote Key Injection

Starts a remote key injection request with IDTech RKI servers. Set/Get RKI url with IDT_Device.RKI_URL.

Parameters

<i>type</i>	0 = Symmetric RKI Demo Unit 1 = Symmetric RKI Production Unit 2 = PKI-RKI Demo Unit 3 = PKI-RKI Production Unit
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString

21.17.2.29 **RETURN_CODE** IDTechSDK.IDT_UniPayl_V.device_startTransaction (double *amount*, double *amtOther*, int *exponent*, int *type*, int *timeout*, byte[] *tags*, bool *forceOnline*, bool *isFastEMV* = false, string *ident* = " ")

Start EMV Transaction Request

Authorizes the EMV transaction for an ICC card

The tags will be returned in the callback routine.

Parameters

<i>amount</i>	Transaction amount value (tag value 9F02, string ident = "")
<i>amtOther</i>	Other amount value, if any (tag value 9F03, string ident = "")
<i>exponent</i>	Number of characters after decimile point
<i>type</i>	Transaction type (tag value 9C).
<i>timeout</i>	Timeout value in seconds.
<i>tags</i>	Any other tags to be included in the request. Passed as TLV Data. Example, tag 9F02 with amount 0x000000000100 would be 0x9F0206000000000100 If tags 9F02 (amount), 9F03 (other amount), or 9C (transaction type) are included, they will take priority over these values supplied as individual parameters to this method.
<i>forceOnline</i>	TRUE = do not allow offline approval, FALSE = allow ICC to approve offline if terminal capable Note: To request tags to be included in default response, use tag DFEE1A, and specify tag list. Example four tags 9F02, 9F36, 95, 9F37 to be included in response = DFEE1A079F029F369f9F37

Parameters

<i>isFastEMV</i>	If TRUE, it will populate the IDTTransactionData.fastEMV with ASCII data similar to IDTech FastEMV KB output, after performing an auto-authenticate and auto-complete with ResultCode = Could Not Contact Host
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString

21.17.2.30 RETURN_CODE IDTechSDK.IDT_UniPayl_V.device_updateDeviceFirmware (byte[] *firmwareData*, string *ident* = " ")

Update Firmware

Updates the firmware .

Parameters

<i>firmwareData</i>	Signed binary data of a firmware file provided by IDTech
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with errorCode.getErrorString(string ident = "")

After you pass the firmwareData file, a new thread will start to execute the firmware download. You will receive status of the progress through callbacks to the IDTechSDK.Callback() delegate. The following parameters will be passed back:

- sender = IDT_DEVICE_Types.IDT_DEVICE_AUGUSTA
- state = DeviceState.FirmwareUpdate
- transactionResultCode = status of the firmware update (starting, entering bootloader, applying update, block success, firmware success, string ident = "")
- data = File Progress. Four bytes, with bytes [0][1] = current block, and bytes [2][3] = total blocks. 0x00030010 = block 3 of 16

Example code starting a firmware update

```

OpenFileDialog diag = new OpenFileDialog( ident);
diag.Filter = "NGA FW Files|*.fm";

if (diag.ShowDialog() == DialogResult.OK, string ident = "")
{
    byte[] file = File.ReadAllBytes(diag.FileName, ident);
    RETURN_CODE rt = IDT_Device.SharedController.device_updateDeviceFirmware(file, ident);
    if (rt == RETURN_CODE.RETURN_CODE_DO_SUCCESS, string ident = "")
    {
        //Was a success
    }
    else
    {
        //Error starting firmware download
    }
}

```

Example monitoring firmware update status / success

```

private void MessageCallBack(IDTechSDK.IDT_DEVICE_Types type, DeviceState state, byte[] data,
    IDTTransactionData cardData, EMV_Callback emvCallback, RETURN_CODE transactionResultCode, string ident = "")
{
    switch (state, string ident = "")
    {
        case DeviceState.FirmwareUpdate:
            switch (transactionResultCode, string ident = "")
            {
                case RETURN_CODE.RETURN_CODE_FW_STARTING_UPDATE:
                    SetOutputText("Starting Firmware Update\n", ident);
                    break;
                case RETURN_CODE.RETURN_CODE_DO_SUCCESS:
                    SetOutputText("Firmware Update Successful\n", ident);
                    break;
                case RETURN_CODE.RETURN_CODE_APPLYING_FIRMWARE_UPDATE:
                    SetOutputText("Applying Firmware Update...\n", ident);
                    break;
                case RETURN_CODE.RETURN_CODE_ENTERING_BOOTLOADER_MODE:
                    SetOutputText("Entering Bootloader Mode...\n", ident);
                    break;
                case RETURN_CODE.RETURN_CODE_BLOCK_TRANSFER_SUCCESS:

                    int start = data[0] * 0x100 + data[1];
                    int end = data[2] * 0x100 + data[3];

                    SetOutputText("Sent block " + start.ToString() + " of " + end.ToString() + "\n", ident)
                    ;
                    break;
                default:
                    SetOutputText("Firmware Update Error Code: " + "0x" + String.Format("{0:X}", (ushort)
transactionResultCode) + ": " + IDTechSDK.errorCode.getErrorString(transactionResultCode) + "\r\n",
ident);
                    break;
            }
        break;
    }
}

```

21.17.2.31 RETURN_CODE IDTechSDK.IDT_UniPayl_V.device_updateFirmwareFromZip (byte[] *zipfile*, string *ident* = "", bool *isForeground* = false)

Update Firmware From Zip

Performs one or more device firmware updates from firmware files passed as a compressed archive with a valid configuration file (.json format)

Parameters

<i>zipfile</i>	Zip file containing on or more firmware update files and a .json configuration file
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.
<i>isForeground</i>	If TRUE, will block program until firmware update complete. Otherwise, FALSE performs FW update on background.

Returns

RETURN_CODE: Values can be parsed with `errorCode.getErrorString(string ident = "")`

21.17.2.32 RETURN_CODE IDTechSDK.IDT_UniPayl_V.emv_activateTransaction (int *timeout*, byte[] *tags*, bool *forceOnline*, bool *isFastEMV* = false, string *ident* = "")

Start EMV Transaction Request

Authorizes the EMV transaction for an ICC card

The tags will be returned in the callback routine.

Parameters

<i>timeout</i>	Timeout value in seconds.
<i>tags</i>	The tags to be included in the request. Passed as TLV Data. Example, tag 9F02 with amount 0x000000000100 would be 0x9F0206000000000100
<i>forceOnline</i>	TRUE = do not allow offline approval, FALSE = allow ICC to approve offline if terminal capable Note: To request tags to be included in default response, use tag DFEE1A, and specify tag list. Example four tags 9F02, 9F36, 95, 9F37 to be included in response = DFEE1A079F029F369f9F37
<i>isFastEMV</i>	If TRUE, it will populate the IDTTransactionData.fastEMV with ASCII data similar to IDTech FastEMV KB output, after performing an auto-authenticate and auto-complete with ResultCode = Could Not Contact Host
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString

21.17.2.33 RETURN_CODE IDTechSDK.IDT_UniPayl_V.emv_addTerminalData (byte[] tlv, string ident = " ")

Add Terminal Data

Adds to the exosting Terminal Data

Parameters

<i>tlv</i>	TerminalData TLV Data
<i>ident</i>	Optional identifier

Return values

<i>RETURN_CODE</i>	Return codes listed as typedef enum in IDTCommon:RETURN_CODE. Values can be parsed with IDT_BTPay::device_getResponseCodeString:()
--------------------	--

21.17.2.34 static void IDTechSDK.IDT_UniPayl_V.emv_allowFallback (bool allow, string ident = " ") [static]

Allow fallback for EMV transactions. Default is TRUE

Parameters

<i>allow</i>	TRUE = allow fallback, FALSE = don't allow fallback
--------------	---

21.17.2.35 RETURN_CODE IDTechSDK.IDT_UniPayl_V.emv_authenticateTransaction (byte[] updatedTLV, string ident = " ")

Authenticate EMV Transaction Request

Authenticates the EMV transaction for an ICC card. Execute this after receiving response with result code 0x10 to emv_startTransaction

The tags will be returned in the callback routine.

@param updatedTLV TLV stream that can be used to update the following values:

- 9F02: Amount
- 9F03: Other amount
- 9C: Transaction type
- 5F57: Account type

In addition tag DFEE1A can be sent to specify tag list to include in results. Example four tags 9F02, 9F36, 95

@param ident Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString

21.17.2.36 static void IDTechSDK.IDT_UniPayl_V.emv_autoAuthenticate (bool *authenticate*, string *ident* = " ") [static]

Enables authenticate for EMV transactions. If a startEMVTransaction results in code 0x0010 (start transaction success), then emv_authenticateEMVTransaction can automatically execute if parameter is set to TRUE

Parameters

<i>authenticate</i>	TRUE = auto authenticate, FALSE = manually authenticate
---------------------	---

21.17.2.37 static void IDTechSDK.IDT_UniPayl_V.emv_autoAuthenticateTags (bool *authenticate*, byte[] *tags*, string *ident* = " ") [static]

Enables authenticate for EMV transactions. If a startEMVTransaction results in code 0x0010 (start transaction success), then emv_authenticateEMVTransaction can automatically execute if parameter is set to TRUE

Parameters

<i>authenticate</i>	TRUE = auto authenticate, FALSE = manually authenticate
<i>tags</i>	Tags to pass during authentication stage;

21.17.2.38 RETURN_CODE IDTechSDK.IDT_UniPayl_V.emv_callbackResponseLCD (EMV_LCD_DISPLAY_MODE *type*, byte *selection*, string *ident* = " ")

Callback Response LCD Display

Provides menu selection responses to the kernel after a callback was received with DeviceState.EMVCallback, and callbackType = EMV_CALLBACK_TYPE.EMV_CALLBACK_TYPE_LCD, and lcd_displayMode = EMV_LCD_DISPLAY_MODE_MENU, EMV_LCD_DISPLAY_MODE_PROMPT, or EMV_LCD_DISPLAY_MODE_LANGUAGE_SELECT

Parameters

<i>type</i>	If Cancel key pressed during menu selection, then value is EMV_LCD_DISPLAY_MODE_CANCEL. Otherwise, value can be EMV_LCD_DISPLAY_MODE_MENU, EMV_LCD_DISPLAY_MODE_PROMPT, or EMV_LCD_DISPLAY_MODE_LANGUAGE_SELECT
<i>selection</i>	If type = EMV_LCD_DISPLAY_MODE_MENU or EMV_LCD_DISPLAY_MODE_LANGUAGE_SELECT, provide the selection ID line number. Otherwise, if type = EMV_LCD_DISPLAY_MODE_PROMPT supply either 0x43 ('C') for Cancel, or 0x45 ('E') for Enter/accept
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with `errorCode.getErrorString()`

21.17.2.39 RETURN_CODE IDTechSDK.IDT_UniPayl_V.emv_callbackResponseMSR (byte[] MSR, string ident = " ")

Callback Response MSR Entry

Provides MSR information to kernel after a callback was received with `DeviceState.EMVCallback`, and `callbackType = EMV_CALLBACK_MSR`

Parameters

<i>MSR</i>	Swiped track data
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with `errorCode.getErrorString()`

21.17.2.40 RETURN_CODE IDTechSDK.IDT_UniPayl_V.emv_callbackResponsePIN (EMV_PIN_MODE type, byte[] KSN, byte[] PIN, string ident = " ")

Callback Response PIN Entry

Provides (or cancels) PIN entry information to kernel after a callback was received with `DeviceState.EMVCallback`, and `callbackType = EMV_CALLBACK_TYPE_PINPAD`

Parameters

<i>type</i>	If Cancel key pressed during PIN entry, then value is <code>EMV_PIN_MODE_CANCEL</code> . Otherwise, value can be <code>EMV_PIN_MODE_ONLINE_DUKPT</code> , <code>EMV_PIN_MODE_ONLINE_MKSK</code> , or <code>EMV_PIN_MODE_OFFLINE</code>
<i>KSN</i>	If enciphered PIN, this is either the PIN DUKPT Key (<code>EMV_PIN_MODE_ONLINE_DUKPT</code>) or PIN Session Key (<code>EMV_PIN_MODE_ONLINE_MKSK</code>), or PIN Pairing DUKPT key (<code>EMV_PIN_MODE_OFFLINE</code> , <code>string ident = ""</code>)
<i>PIN</i>	If enciphered PIN, this is encrypted PIN block. If device does not implement pairing function, this is plaintext PIN
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with `device_getResponseCodeString`

21.17.2.41 RETURN_CODE IDTechSDK.IDT_UniPayl_V.emv_cancelTransaction (string ident = " ")

Cancel Transaction

Cancels the currently executing EMV transaction.

Parameters

<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.
--------------	--

Returns

RETURN_CODE: Values can be parsed with `errorCode.getErrorString()`

21.17.2.42 RETURN_CODE `IDTechSDK.IDT_UniPayl_V.emv_completeTransaction (bool commError, byte[] authCode, byte[] iad, byte[] tlvScripts, byte[] tlv, string ident = " ")`

Complete EMV Transaction Request

Completes the EMV transaction for an ICC card when online authorization request is received from `emv_authenticateTransaction`

The tags will be returned in the callback routine.

Parameters

<i>commError</i>	Communication error with host. Set to TRUE if host was unreachable, or FALSE if host response received. If Communication error, <i>authCode</i> , <i>iad</i> , <i>tlvScripts</i> can be null.
<i>authCode</i>	Authorization code from host. Two bytes. Example 0x3030. (Tag value 8A). Required
<i>iad</i>	Issuer Authentication Data, if any. Example 0x11223344556677883030 (tag value 91).
<i>tlvScripts</i>	71/72 scripts, if any
<i>tlv</i>	Additional TVL data to return with transaction results (if any, string <i>ident</i> = "")
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with `device_getResponseCodeString`

NOTE: There are three possible outcomes for Authorization Code: Approval, Refer To Bank, Decline. The kernel maps these three outcomes to valid authorization response codes using tag DFEE1B through 8 bytes: {2 bytes Approval Code}{2 bytes Referral Code}{2 bytes Decline Code}{2 bytes RFU} If your gateway uses "00" for Approval, "01" for Referral, and "05" for Decline, then DFEE1B 08 3030 3031 3035 0000 If you use an authorization code value that is not defined in DFEE1B, the kernel will use the DECLINE value of DFEE1B by default.

21.17.2.43 RETURN_CODE `IDTechSDK.IDT_UniPayl_V.emv_getEMVConfigurationCheckValue (ref string response, string ident = " ")`

Polls device for EMV Configuration Check Value

Parameters

<i>response</i>	Response returned of Check Value of Configuration
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with `errorCode.getErrorString(string ident = "")`

21.17.2.44 RETURN_CODE `IDTechSDK.IDT_UniPayl_V.emv_getEMVKernelCheckValue (ref string response, string ident = " ")`

Polls device for EMV Kernel Check Value

Parameters

<i>response</i>	Response returned of Check Value of Kernel
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with `errorCode.getErrorString(string ident = "")`

21.17.2.45 RETURN_CODE IDTechSDK.IDT_UniPayl_V.emv_getEMVKernelVersion (ref string *response*, string *ident* = " ")

Polls device for EMV Kernel Version

Parameters

<i>response</i>	Response returned of Kernel Version
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with `errorCode.getErrorString(string ident = "")`

21.17.2.46 RETURN_CODE IDTechSDK.IDT_UniPayl_V.emv_getEMVKernelVersionExt (ref string *response*, string *ident* = " ")

Polls device for Extended EMV Kernel Version

Parameters

<i>response</i>	Response returned of Kernel Version
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with `errorCode.getErrorString(string ident = "")`

21.17.2.47 RETURN_CODE IDTechSDK.IDT_UniPayl_V.emv_removeAllApplicationData (string *ident* = " ")

Remove All Application Data

Removes all the Application Data for EMV Kernel

Parameters

<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.
--------------	--

Returns

RETURN_CODE: Values can be parsed with `errorCode.getErrorString(string ident = "")`

21.17.2.48 RETURN_CODE IDTechSDK.IDT_UniPayl_V.emv_removeAllCAPK (string *ident* = " ")

Remove All Certificate Authority Public Key

Removes all the CAPK for EMV Kernel

Parameters

<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.
--------------	--

Returns

RETURN_CODE: Values can be parsed with `errorCode.getErrorString(string ident = "")`

21.17.2.49 RETURN_CODE IDTechSDK.IDT_UniPayl_V.emv_removeAllCRL (string *ident* = " ")

Remove All Certificate Revocation List Entries

Removes all CRLentry entries

Parameters

<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.
--------------	--

Returns

RETURN_CODE: Values can be parsed with `errorCode.getErrorString(string ident = "")`

21.17.2.50 RETURN_CODE IDTechSDK.IDT_UniPayl_V.emv_removeApplicationData (byte[] *AID*, string *ident* = " ")

Remove Application Data by AID

Removes the Application Data for EMV Kernel as specified by the AID name passed as a parameter

Parameters

<i>AID</i>	Name of ApplicationID Must be between 5 and 16 bytes
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with `errorCode.getErrorString(string ident = "")`

21.17.2.51 RETURN_CODE IDTechSDK.IDT_UniPayl_V.emv_removeCAPK (byte[] *capk*, string *ident* = " ")

Remove Certificate Authority Public Key

Removes the CAPK for EMV Kernel as specified by the RID/Index

Parameters

<i>capk</i>	6 byte CAPK = 5 bytes RID + 1 byte INDEX
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with `errorCode.getErrorString(string ident = "")`

21.17.2.52 RETURN_CODE IDTechSDK.IDT_UniPayl_V.emv_removeCRL (byte[] *crlList*, string *ident* = " ")

Remove Certificate Revocation List Entries

Removes CRL entries as specified by the RID and Index and serial number passed as 9 bytes

Parameters

<i>crlList</i>	containing the list of CRL to remove: [CRL1][CRL2]...[CRLn] where each [CRL] is 9 bytes: [5 bytes RID][1 byte CAPK Index][3 bytes serial number]
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with `device_getResponseCodeString`

21.17.2.53 RETURN_CODE IDTechSDK.IDT_UniPayl_V.emv_removeTerminalData (string *ident* = " ")

Remove Terminal Data

Removes the Terminal Data

Parameters

<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.
--------------	--

Returns

RETURN_CODE: Values can be parsed with `device_getResponseCodeString`

21.17.2.54 RETURN_CODE IDTechSDK.IDT_UniPayl_V.emv_retrieveAIDList (ref byte *response*[], string *ident* = " ")

Retrieve AID list

Returns all the AID names installed on the terminal for EMV Kernel.

Parameters

<i>response</i>	array of AID name byte arrays
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with `errorCode.getErrorString(string ident = "")`

21.17.2.55 RETURN_CODE IDTechSDK.IDT_UniPayl_V.emv_retrieveApplicationData (byte[] *AID*, ref byte[] *tlv*, string *ident* = " ")

Retrieve Application Data by AID

Retrieves the Application Data for EMV Kernel as specified by the AID name passed as a parameter.

Parameters

<i>AID</i>	Name of ApplicationID. Must be between 5 and 16 bytes
<i>tlv</i>	The TLV elements of the requested AID
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with `errorCode.getErrorString()`

21.17.2.56 RETURN_CODE IDTechSDK.IDT_UniPayl_V.emv_retrieveCAPK (byte[] *capk*, ref byte[] *key*, string *ident* = " ")

Retrieve Certificate Authority Public Key

Retrieves the CAPK for EMV Kernel as specified by the RID/Index passed as a parameter.

Parameters

<i>capk</i>	6 bytes CAPK = 5 bytes RID + 1 byte Index
<i>key</i>	Response returned as a CAKey format: [5 bytes RID][1 byte Index][1 byte Hash Algorithm][1 byte Encryption Algorithm][20 bytes HashValue][4 bytes Public Key Exponent][2 bytes Modulus Length][Variable bytes Modulus] Where: <ul style="list-style-type: none"> • Hash Algorithm: The only algorithm supported is SHA-1. The value is set to 0x01 • Encryption Algorithm: The encryption algorithm in which this key is used. Currently support only one type: RSA. The value is set to 0x01. • HashValue: Which is calculated using SHA-1 over the following fields: RID & Index & Modulus & Exponent • Public Key Exponent: Actually, the real length of the exponent is either one byte or 3 bytes. It can have two values: 3 (Format is 0x00 00 00 03), or 65537 (Format is 0x00 01 00 01, string <i>ident</i> = "") • Modulus Length: LenL LenH Indicated the length of the next field. • Modulus: This is the modulus field of the public key. Its length is specified in the field above.
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with `errorCode.getErrorString()`

21.17.2.57 RETURN_CODE IDTechSDK.IDT_UniPayl_V.emv_retrieveCAPKList (ref byte[] *keys*, string *ident* = " ")

Retrieve the Certificate Authority Public Key list

Returns all the CAPK RID and Index installed on the terminal for EMV Kernel.

Parameters

<i>keys</i>	[key1][key2]...[keyn], each key 6 bytes where key = 5 bytes RID + 1 byte index
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with `errorCode.getErrorString()`

21.17.2.58 RETURN_CODE IDTechSDK.IDT_UniPayl_V.emv_retrieveCRLList (ref byte[] list, string ident = " ")

Retrieve the Certificate Revocation List

Returns the CRL entries on the terminal.

Parameters

<i>list</i>	[CRL1][CRL2]...[CRLn], each CRL 9 bytes where CRL = 5 bytes RID + 1 byte index + 3 bytes serial number
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with `errorCode.getErrorString()`

21.17.2.59 RETURN_CODE IDTechSDK.IDT_UniPayl_V.emv_retrieveTerminalData (ref byte[] tlv, string ident = " ")

Retrieve Terminal Data

Retrieves the Terminal Data for EMV Kernel.

Parameters

<i>tlv</i>	Response returned as a TLV
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with `errorCode.getErrorString()`

21.17.2.60 RETURN_CODE IDTechSDK.IDT_UniPayl_V.emv_retrieveTransactionResult (byte[] tags, ref IDTTransactionData tlv, string ident = " ")

Retrieve Transaction Results

Retrieves specified EMV tags from the currently executing transaction.

Parameters

<i>tags</i>	Tags to be retrieved. Example 0x9F028A will retrieve tags 9F02 and 8A
<i>tlv</i>	All requested tags returned as unencrypted, encrypted and masked TLV data in IDTTransactionData object
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with `device.getResponseCodeString`

21.17.2.61 RETURN_CODE IDTechSDK.IDT_UniPayl_V.emv_setApplicationData (byte[] name, byte[] tlv, string ident = " ")

Set Application Data by AID

Sets the Application Data for EMV Kernel as specified by the application name and TLV data

Parameters

<i>name</i>	Application name, 10-32 ASCII hex characters representing 5-16 bytes Example "a0000000031010"
<i>tlv</i>	Application data in TLV format
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with `errorCode.getErrorString()`

21.17.2.62 RETURN_CODE IDTechSDK.IDT_UniPayl_V.emv_setCAPK (byte[] key, string ident = " ")

Set Certificate Authority Public Key

Sets the CAPK for EMV Kernel as specified by the CAKey structure

Parameters

<i>key</i>	CAKey format: [5 bytes RID][1 byte Index][1 byte Hash Algorithm][1 byte Encryption Algorithm][20 bytes HashValue][4 bytes Public Key Exponent][2 bytes Modulus Length][Variable bytes Modulus] Where: <ul style="list-style-type: none"> • Hash Algorithm: The only algorithm supported is SHA-1. The value is set to 0x01 • Encryption Algorithm: The encryption algorithm in which this key is used. Currently support only one type: RSA. The value is set to 0x01. • HashValue: Which is calculated using SHA-1 over the following fields: RID & Index & Modulus & Exponent • Public Key Exponent: Actually, the real length of the exponent is either one byte or 3 bytes. It can have two values: 3 (Format is 0x00 00 00 03), or 65537 (Format is 0x00 01 00 01, string ident = "") • Modulus Length: LenL LenH Indicated the length of the next field. • Modulus: This is the modulus field of the public key. Its length is specified in the field above.
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with `errorCode.getErrorString()`

21.17.2.63 RETURN_CODE IDTechSDK.IDT_UniPayl_V.emv_setCRL (byte[] list, string ident = " ")

Set Certificate Revocation List

Sets the CRL

Parameters

<i>list</i>	CRL Entries containing the RID, Index, and serial numbers to set [CRL1][CRL2]...[CRLn] where each [CRL] is 9 bytes: [5 bytes RID][1 byte CAPK Index][3 bytes serial number]
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with `errorCode.getErrorString()`

21.17.2.64 RETURN_CODE IDTechSDK.IDT_UniPayl_V.`emv_setTerminalData` (`byte[] tlv`, `string ident = ""`)

Set Terminal Data

Sets the Terminal Data

Parameters

<i>tlv</i>	TerminalData TLV Data
<i>ident</i>	Optional identifier

Return values

<i>RETURN_CODE</i>	Return codes listed as typedef enum in IDTCommon:RETURN_CODE. Values can be parsed with <code>IDT_BTPay::device_getResponseCodeString:()</code>
--------------------	---

21.17.2.65 RETURN_CODE IDTechSDK.IDT_UniPayl_V.`emv_setTerminalMajorConfiguration` (`int configuration`, `string ident = ""`)

Set Terminal Major Configuration

Sets the Terminal Data Major Configuration setting

Parameters

<i>configuration</i>	The configuration to set (1-5, string ident = "")
----------------------	---

Return values

<i>RETURN_CODE</i>	Return codes listed as typedef enum in IDTCommon:RETURN_CODE. Values can be parsed with IDT_BTPay::device_getResponseCodeString:(string ident = "")
--------------------	---

21.17.2.66 RETURN_CODE IDTechSDK.IDT_UniPayl_V.`emv_startTransaction` (`double amount`, `double amtOther`, `int exponent`, `int type`, `int timeout`, `byte[] tags`, `bool forceOnline`, `bool isFastEMV = false`, `string ident = ""`)

Start EMV Transaction Request

Authorizes the EMV transaction for an ICC card

The tags will be returned in the callback routine.

Parameters

<i>amount</i>	Transaction amount value (tag value 9F02, string ident = "")
<i>amtOther</i>	Other amount value, if any (tag value 9F03, string ident = "")
<i>exponent</i>	Number of characters after decimile point
<i>type</i>	Transaction type (tag value 9C).
<i>timeout</i>	Timeout value in seconds.
<i>tags</i>	Any other tags to be included in the request. Passed as TLV Data. Example, tag 9F02 with amount 0x000000000100 would be 0x9F0206000000000100 If tags 9F02 (amount), 9F03 (other amount), or 9C (transaction type) are included, they will take priority over these values supplied as individual parameters to this method.
<i>forceOnline</i>	TRUE = do not allow offline approval, FALSE = allow ICC to approve offline if terminal capable Note: To request tags to be included in default response, use tag DFEE1A, and specify tag list. Example four tags 9F02, 9F36, 95, 9F37 to be included in response = DFEE1A079F029F369F37
<i>isFastEMV</i>	If TRUE, it will populate the IDTTransactionData.fastEMV with ASCII data similar to IDTech FastEMV KB output, after performing an auto-authenticate and auto-complete with ResultCode = Could Not Contact Host
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString

21.17.2.67 **RETURN_CODE** IDTechSDK.IDT_UniPayl_V.emv_trySetTerminalData (byte[] *tlv*, ref byte[] *rejectedTLV*, ref byte[] *convertedTLV*, bool *overwrite* = false, string *ident* = " ")

Try to Set Terminal Data

Attempts to set the Terminal Data. Use this functions to attempt to set the terminal data to the device. This function will allow the cross-device tag compatible writing of IDTech Legacy/New tag definitions

Parameters

<i>tlv</i>	TerminalData TLV Data
<i>rejectedTLV</i>	Contains the tags (if any) that were rejected by the firmware
<i>convertedTLV</i>	Contains the tags (if any) that were converted and accepted by the firmware
<i>overwrite</i>	TRUE = add TLV to existing tags, FALSE = replace existing tags with TLV
<i>ident</i>	Optional identifier

Return values

RETURN_CODE	Return codes listed as typedef enum in IDTCommon:RETURN_CODE. Values can be parsed with IDT_BTPay::device_getResponseCodeString:()
--------------------	--

21.17.2.68 **static int** IDTechSDK.IDT_UniPayl_V.getCommandTimeout (string *ident* = " ") [static]

Get Command Timeout

Gets the default timeout (in milliseconds) waiting for a blocking command response

Return values

<i>time</i>	Time
-------------	------

21.17.2.69 **RETURN_CODE** IDTechSDK.IDT_UniPayl_V.icc_exchangeAPDU (string *c_APDU*, ref byte[] *response*, string *ident* = " ")

Exchange APDU

Sends an APDU packet to the ICC. If successful, response is returned in APDUResult class instance in response parameter.

Parameters

<i>c_APDU</i>	APDU data packet
<i>response</i>	Unencrypted/encrypted parsed APDU response
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with `errorCode.getErrorString(string ident = "")`

21.17.2.70 **RETURN_CODE** IDTechSDK.IDT_UniPayl_V.icc_getICCReaderStatus (ref byte *status*, string *ident* = " ")

Get Reader Status

Returns the reader status

Parameters

<i>status</i>	Pointer that will return with the ICCReaderStatus results. bit 0: 0 = ICC Power Not Ready, 1 = ICC Powered bit 1: 0 = Card not seated, 1 = card seated
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with `device_getResponseCodeString`

21.17.2.71 **RETURN_CODE** IDTechSDK.IDT_UniPayl_V.icc_powerOffICC (string *ident* = " ")

Power Off ICC

Powers down the ICC

Parameters

<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.
--------------	--

Returns

RETURN_CODE: Values can be parsed with `errorCode.getErrorString(string ident = "")`

If Success, empty If Failure, ASCII encoded data of error string

21.17.2.72 **RETURN_CODE** IDTechSDK.IDT_UniPayl_V.icc_powerOnICC (ref byte[] *ATR*, byte *interfaces*, string *ident* = " ")

Power On ICC

Power up the currently selected microprocessor card in the ICC reader

Parameters

<i>interfaces</i>	For NEO2 devices allowed interfaces for which to get the ATR. 0x20h = retrieve last ATR received from PICC 0x21h = SAM1 (SRED version only, string ident = "") 0x22h = SAM2 (SRED version only, string ident = "") For other devices interfaces euquals to 0s
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with `errorCode.getErrorString(string ident = "")`

21.17.2.73 **static void** IDTechSDK.IDT_UniPayl_V.lcd_retrieveMessage (DisplayMessages.DISPLAY_MESSAGE_LANGUAGE *lang*, DisplayMessages.DISPLAY_MESSAGE_IDENTIFIER *id*, ref string *line1*, ref string *line2*) [static]

Retrieve LCD Message

Returns the string value for a message ID returned for LCD messaging

Parameters

<i>lang</i>	Language.
<i>id</i>	Message ID
<i>line1</i>	Line 1 string value
<i>line2</i>	Line 2 string value
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with `device_getResponseCodeString`

21.17.2.74 **RETURN_CODE** IDTechSDK.IDT_UniPayl_V.msr_cancelMSRSwipe (string *ident* = " ")

Disable MSR Swipe Cancels MSR swipe request.

Parameters

<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.
--------------	--

Returns

RETURN_CODE: Values can be parsed with `errorCode.getErrorString(string ident = "")`

21.17.2.75 **RETURN_CODE** IDTechSDK.IDT_UniPayl_V.msr_startMSRSwipe (int *timeout*, string *ident* = " ")

Enable MSR Swipe

Enables MSR, waiting for swipe to occur.

Parameters

<i>timeout</i>	Swipe Timeout Value
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with `errorCode.getErrorString(string ident = "")`

21.17.2.76 RETURN_CODE IDTechSDK.IDT_UniPayl_V.msr_switchUSBInterfaceMode (bool *blsUSBKeyboardMode*)

Switch the USB interface mode between USB HID and USB KB mode.

For Non-SRED Augusta Only

Parameters

<i>blsUSBKeyboardMode</i>	USB interface mode <ul style="list-style-type: none"> • true: Enter into USB Keyboard mode • false: Enter into USB HID mode
---------------------------	---

Returns

RETURN_CODE: Values can be parsed with `device_getResponseCodeString`

21.17.2.77 RETURN_CODE IDTechSDK.IDT_UniPayl_V.msr_switchUSBInterfaceMode (bool *blsUSBKeyboardMode*, ref string *ident*)

Switch the USB interface mode between USB HID and USB KB mode.

For Non-SRED Augusta Only

Parameters

<i>blsUSBKeyboardMode</i>	USB interface mode <ul style="list-style-type: none"> • true: Enter into USB Keyboard mode • false: Enter into USB HID mode
<i>ident</i>	Device ID to send command to.

Returns

RETURN_CODE: Values can be parsed with `device_getResponseCodeString`

21.17.2.78 static String IDTechSDK.IDT_UniPayl_V.SDK_Version () [static]

SDK Version

- All Devices

Returns the current version of SDK

Returns

Framework version

21.17.2.79 `static void IDTechSDK.IDT_UniPayl_V.setCallback (Callback my_Callback) [static]`

Set Callback

Sets the class callback

21.17.2.80 `static void IDTechSDK.IDT_UniPayl_V.setCallback (IntPtr my_Callback, SynchronizationContext context) [static]`

Set Callback

Sets the class callback

Parameters

<i>my_Callback</i>	The callback function to receive the response message from device. defined as follows. public unsafe delegate void MFCCallBack(Parameters parameters, ident);
<i>context</i>	The context of the UI thread

21.17.2.81 `static void IDTechSDK.IDT_UniPayl_V.setCommandTimeout (int milliseconds, string ident = " ") [static]`

Set Command Timeout

Sets the default timeout (in milliseconds) waiting for a blocking command response

Parameters

<i>milliseconds</i>	Time
---------------------	------

21.17.3 Property Documentation

21.17.3.1 `IDT_UniPayl_V IDTechSDK.IDT_UniPayl_V.SharedController [static],[get]`

Singleton Instance

Establishes an singleton instance of [IDT_UniPayl_V](#) class.

Returns

Instance of [IDT_UniPayl_V](#)

The documentation for this class was generated from the following file:

- /Users/randy/Repo/universal-sdk/IDTechSDK_STD/DeviceHeaders/IDT_UniPayl_V.cs

21.18 IDTechSDK.IDT_Vendi Class Reference

Public Member Functions

- RETURN_CODE [device_controlLED](#) (byte indexLED, byte control, string ident="")
- RETURN_CODE [config_setBaudRate](#) (int baud, string ident="")
- RETURN_CODE [device_retrieveAIDList](#) (ref byte[] response, string ident="")
- RETURN_CODE [device_sendDataCommand](#) (string cmd, bool calcLRC, ref byte[] response, string ident="")
- RETURN_CODE [device_sendDataCommand_ext](#) (string cmd, bool calcLRC, ref byte[] response, int timeout, bool noResponse, string ident="")
- RETURN_CODE [config_getSerialNumber](#) (ref string response, string ident="")
- RETURN_CODE [device_setBurstMode](#) (byte mode, string ident="")
- RETURN_CODE [device_setMerchantRecord](#) (int index, bool enabled, string merchantID, string merchantURL, string ident="")
- RETURN_CODE [device_setPollMode](#) (byte mode, string ident="")
- RETURN_CODE [device_startRKI](#) (string ident="")
- RETURN_CODE [device_getFirmwareVersion](#) (ref string response, string ident="")
- RETURN_CODE [device_getMerchantRecord](#) (int index, ref byte[] record, string ident="")
- RETURN_CODE [device_getTransactionResults](#) (ref IDTTransactionData results, string ident="")
- RETURN_CODE [device_pingDevice](#) (string ident="")
- RETURN_CODE [device_controlUserInterface](#) (byte[] values, string ident="")
- RETURN_CODE [device_sendVivoCommandP2](#) (byte command, byte subCommand, byte[] data, ref byte[] response, string ident="")
- RETURN_CODE [device_sendVivoCommandP2_ext](#) (byte command, byte subCommand, byte[] data, ref byte[] response, int timeout, bool noResponse, string ident="")
- RETURN_CODE [ctls_retrieveAIDList](#) (ref byte[] response, string ident="")
- RETURN_CODE [ctls_getAllConfigurationGroups](#) (ref byte[] response, string ident="")
- RETURN_CODE [ctls_retrieveApplicationData](#) (byte[] AID, ref byte[] tlv, string ident="")
- RETURN_CODE [ctls_removeApplicationData](#) (byte[] AID, string ident="")
- RETURN_CODE [ctls_setApplicationData](#) (byte[] tlv, string ident="")
- RETURN_CODE [ctls_setDefaultConfiguration](#) (string ident="")
- RETURN_CODE [ctls_setConfigurationGroup](#) (byte[] tlv, string ident="")
- RETURN_CODE [device_sendPAE](#) (string command, ref string response, int timeout, string ident="")
- RETURN_CODE [ctls_retrieveTerminalData](#) (ref byte[] tlv, string ident="")
- RETURN_CODE [ctls_setTerminalData](#) (byte[] tlv, string ident="")
- RETURN_CODE [ctls_trySetTerminalData](#) (byte[] tlv, ref byte[] rejectedTLV, ref byte[] convertedTLV, string ident="")
- RETURN_CODE [ctls_getConfigurationGroup](#) (int group, ref byte[] tlv, string ident="")
- RETURN_CODE [ctls_removeConfigurationGroup](#) (int group, string ident="")
- RETURN_CODE [ctls_setCAPK](#) (byte[] key, string ident="")
- RETURN_CODE [ctls_retrieveCAPK](#) (byte[] capk, ref byte[] key, string ident="")
- RETURN_CODE [ctls_removeCAPK](#) (byte[] capk, string ident="")
- RETURN_CODE [ctls_removeAllCAPK](#) (string ident="")
- RETURN_CODE [ctls_retrieveCAPKList](#) (ref byte[] keys, string ident="")
- RETURN_CODE [ctls_startTransaction](#) (double amount, double amtOther, int exponent, int type, int timeout, byte[] tags, bool isFastEMV=false, string ident="")
- RETURN_CODE [ctls_activateTransaction](#) (int timeout, byte[] tags, bool isFastEMV=false, string ident="")
- RETURN_CODE [device_startTransaction](#) (double amount, double amtOther, int exponent, int type, int timeout, byte[] tags, bool isFastEMV=false, string ident="")
- RETURN_CODE [device_activateTransaction](#) (int timeout, byte[] tags, bool isFastEMV=false, string ident="")
- RETURN_CODE [ctls_cancelTransaction](#) (string ident="")
- RETURN_CODE [msr_startMSRSwipe](#) (int timeout, string ident="")
- RETURN_CODE [msr_cancelMSRSwipe](#) (string ident="")
- RETURN_CODE [device_getRKIStatus](#) (bool isProd, bool isMultiKey, ref string status, ref Dictionary< string, RKI_KEY_TYPE > keys, string ident="")
- RETURN_CODE [device_getAnyRKIStatus](#) (bool isMultiKey, ref string status, ref Dictionary< string, RKI_KEY_TYPE > keys, string ident="")

- RETURN_CODE [device_updateFirmwareFromZip](#) (byte[] zipfile, string ident="", bool isForeground=false)
- RETURN_CODE [device_sendConfigurationFromZip](#) (byte[] zip, string filename, VC_OPERATION_TYPE type, bool matchFW, string ident="", bool isForeground=false)
- RETURN_CODE [device_sendConfiguration](#) (string filename, VC_OPERATION_TYPE type, bool matchFW, string ident="", bool isForeground=false)
- RETURN_CODE [device_readConfigurationToMemory](#) (string memo, Configs.InstallRules rules, List< Configs.DeviceCommand > cmds, string ident="", bool isForeground=false)
- RETURN_CODE [device_getConfigurationFromMemory](#) (ref string json, string ident="")
- RETURN_CODE [device_RemoteKeyInjection](#) (RKI_KEY_TYPE type, string keyName, string ident="", bool performOnForeground=false)

Static Public Member Functions

- static bool [useSerialPort](#) (int port)
- static bool [useSerialPort](#) (int port, int baud)
- static bool [useSerialPortLinux](#) (string path)
- static bool [useSerialPortLinux](#) (string path, int baud)
- static int [getCommandTimeout](#) (string ident="")
- static void [setCommandTimeout](#) (int milliseconds, string ident="")
- static void [initSC](#) ()
- static void [setCallback](#) (Callback my_Callback)
- static void [setCallback](#) (IntPtr my_Callback, SynchronizationContext context)
- static String [SDK_Version](#) ()
- static RETURN_CODE [device_updateDeviceFirmware](#) (byte[] firmwareData, string ident="")
- static void [lcd_retrieveMessage](#) (DisplayMessages.DISPLAY_MESSAGE_LANGUAGE lang, DisplayMessages.DISPLAY_MESSAGE_IDENTIFIER id, ref string line1, ref string line2)

Properties

- static [IDT_Vendi SharedController](#) [get]

21.18.1 Member Function Documentation

21.18.1.1 RETURN_CODE IDTechSDK.IDT_Vendi.config_getSerialNumber (ref string *response*, string *ident* = " ")

Polls device for Serial Number

Parameters

<i>response</i>	Returns Serial Number
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with [device_getResponseCodeString](#)

21.18.1.2 RETURN_CODE IDTechSDK.IDT_Vendi.config_setBaudRate (int *baud*, string *ident* = " ")

Set Baud Rate

Sets the baud rate for RS-232 communication.

Parameters

<i>baud</i>	<ul style="list-style-type: none"> • 4 = 9600 • 6 = 19200 • 7 = 38400 • 8 = 57600 • 9 = 115200
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString

21.18.1.3 RETURN_CODE IDTechSDK.IDT_Vendi.ctls_activateTransaction (int *timeout*, byte[] *tags*, bool *isFastEMV* = false, string *ident* = " ")

Start CTLS Transaction Request

The tags will be returned in the callback routine.

Parameters

<i>timeout</i>	Timeout value in seconds.
<i>tags</i>	The tags to be included in the request. Passed as TLV Data. Example, tag 9F02 with amount 0x0000000000100 would be 0x9F02060000000000100
<i>isFastEMV</i>	If TRUE, it will populate the IDTTransactionData.fastEMV with ASCII data similar to IDTech FastEMV KB output, after performing an auto-authenticate and auto-complete with ResultCode = Could Not Contact Host
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with errorCode.getErrorString()

NOTE ON APPLEPAY VAS: To enable ApplePay VAS, first a merchant record must be defined in one of the six available index positions (1-6) using device_setMerchantRecord, then container tag FFEE06 must be sent as part of the additional tags parameter of ctls_startTransaction. Tag FFEE06 must contain tag 9F26 and 9F22, and can optionally contain tags 9F2B and DFO1. Example FFEE06189F220201009F2604000000009F2B050100000000DF010101 9F22 = two bytes = ApplePay Terminal Application Version Number. Hard defined as 0100 for now. (required, string ident = "") 9F26 = four bytes = ApplePay Terminal Capabilities Information (required, string ident = "")

- Byte 1 = RFU
- Byte 2 = Terminal Type
- - Bit 8 = VAS Support (1=on, 0 = off, string ident = "")
- - Bit 7 = Touch ID Required (1=on, 0 = off, string ident = "")
- - Bit 6 = RFU

- - Bit 5 = RFU
- - Bit 1,2,3,4
- - - 0 = Payment Terminal
- - - 1 = Transit Terminal
- - - 2 = Access Terminal
- - - 3 = Wireless Handoff Terminal
- - - 4 = App Handoff Terminal
- - - 15 = Other Terminal
- Byte 3 = RFU
- Byte 4 = Terminal Mode
- - 0 = ApplePay VAS OR ApplePay
- - 1 = ApplePay VAS AND ApplePay
- - 2 = ApplePay VAS ONLY
- - 3 = ApplePay ONLY 9F2B = 5 bytes = ApplePay VAS Filter. Each byte filters for that specific merchant index (optional, string ident = "") DF01 = 1 byte = ApplePay VAS Protocol. (optional, string ident = "")
- - Bit 1 : 1 = URL VAS, 0 = Full VAS
- - Bit 2 : 1 = VAS Beeps, 0 = No VAS Beeps
- - Bit 3 : 1 = Silent Comm Error, 2 = EMEA Comm Error
- - Bit 4-8 : RFU

21.18.1.4 RETURN_CODE IDTechSDK.IDT_Vendi.ctls_cancelTransaction (string *ident* = " ")

Cancel EMV Transaction

Cancels the currently executing EMV transaction.

Parameters

<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.
--------------	--

Returns

RETURN_CODE: Values can be parsed with `errorCode.getErrorString()`

21.18.1.5 RETURN_CODE IDTechSDK.IDT_Vendi.ctls_getAllConfigurationGroups (ref byte *response*[[]], string *ident* = " ")

Retrieve All Configuration Groups

Returns all the Configuration Groups installed on the terminal for CTLS

Parameters

<i>response</i>	array of CTLS groups as TLV bytes
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with `errorCode.getErrorString(string ident = "")`

21.18.1.6 RETURN_CODE IDTechSDK.IDT_Vendi.ctls_getConfigurationGroup (int *group*, ref byte[] *tlv*, string *ident* = " ")

Get Configuration Group

Retrieves the Configuration for the specified Group.

Parameters

<i>group</i>	Configuration Group
<i>tlv</i>	return data
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with `errorCode.getErrorString()`

21.18.1.7 RETURN_CODE IDTechSDK.IDT_Vendi.ctls_removeAllCAPK (string *ident* = " ")

Remove All Certificate Authority Public Key

Removes all the CAPK for CTLS

Parameters

<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.
--------------	--

Returns

RETURN_CODE: Values can be parsed with `errorCode.getErrorString(string ident = "")`

21.18.1.8 RETURN_CODE IDTechSDK.IDT_Vendi.ctls_removeApplicationData (byte[] *AID*, string *ident* = " ")

Remove Application Data by AID

Removes the Application Data for CTLS as specified by the AID name passed as a parameter

Parameters

<i>AID</i>	Name of ApplicationID Must be between 5 and 16 bytes
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with `errorCode.getErrorString(string ident = "")`

21.18.1.9 RETURN_CODE IDTechSDK.IDT_Vendi.ctls_removeCAPK (byte[] *capk*, string *ident* = " ")

Remove Certificate Authority Public Key

Removes the CAPK as specified by the RID/Index

Parameters

<i>capk</i>	6 byte CAPK = 5 bytes RID + 1 byte INDEX
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with `errorCode.getErrorString(string ident = "")`

21.18.1.10 RETURN_CODE IDTechSDK.IDT_Vendi.ctls_removeConfigurationGroup (int group, string ident = " ")

Remove Configuration Group

Removes the Configuration as specified by the Group. Must not by group 0

Parameters

<i>group</i>	Configuration Group
--------------	---------------------

Return values

<i>RETURN_CODE</i>	Return codes listed as typedef enum in IDTCommon:RETURN_CODE. Values can be parsed with IDT_BTPay::device_getResponseCodeString:(string ident = "")
--------------------	---

21.18.1.11 RETURN_CODE IDTechSDK.IDT_Vendi.ctls_retrieveAIDList (ref byte response[[]], string ident = " ")

Retrieve AID list

Returns all the AID names and their assigned groups installed on the terminal for CTLS.

Parameters

<i>response</i>	array of 2-tag TLV data objects: FFE4 (group name) followed by 9F06 (AID, string ident = "")
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with `errorCode.getErrorString(string ident = "")`

21.18.1.12 RETURN_CODE IDTechSDK.IDT_Vendi.ctls_retrieveApplicationData (byte[] AID, ref byte[] tlv, string ident = " ")

Retrieve Application Data by AID

Retrieves the CTLS Application Data as specified by the AID name passed as a parameter.

Parameters

<i>AID</i>	Name of ApplicationID. Must be between 5 and 16 bytes
<i>tlv</i>	The TLV elements of the requested AID
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with `errorCode.getErrorString()`

21.18.1.13 RETURN_CODE IDTechSDK.IDT_Vendi.ctls_retrieveCAPK (byte[] capk, ref byte[] key, string ident = " ")

Retrieve Certificate Authority Public Key

Retrieves the CAPK for CTLS as specified by the RID/Index passed as a parameter.

Parameters

<i>capk</i>	6 bytes CAPK = 5 bytes RID + 1 byte Index
<i>key</i>	<p>Response returned as a CAKey format: [5 bytes RID][1 byte Index][1 byte Hash Algorithm][1 byte Encryption Algorithm][20 bytes HashValue][4 bytes Public Key Exponent][2 bytes Modulus Length][Variable bytes Modulus] Where:</p> <ul style="list-style-type: none"> • Hash Algorithm: The only algorithm supported is SHA-1. The value is set to 0x01 • Encryption Algorithm: The encryption algorithm in which this key is used. Currently support only one type: RSA. The value is set to 0x01. • HashValue: Which is calculated using SHA-1 over the following fields: RID & Index & Modulus & Exponent • Public Key Exponent: Actually, the real length of the exponent is either one byte or 3 bytes. It can have two values: 3 (Format is 0x00 00 00 03), or 65537 (Format is 0x00 01 00 01, string ident = "") • Modulus Length: LenL LenH Indicated the length of the next field. • Modulus: This is the modulus field of the public key. Its length is specified in the field above.
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with `errorCode.getErrorString()`

21.18.1.14 RETURN_CODE IDTechSDK.IDT_Vendi.ctls_retrieveCAPKList (ref byte[] keys, string ident = " ")

Retrieve the Certificate Authority Public Key list

Returns all the CAPK RID and Index installed on the terminal for CTLS.

Parameters

<i>keys</i>	[key1][key2]...[keyn], each key 6 bytes where key = 5 bytes RID + 1 byte index
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with `errorCode.getErrorString()`

21.18.1.15 RETURN_CODE IDTechSDK.IDT_Vendi.ctls_retrieveTerminalData (ref byte[] tlv, string ident = " ")

Retrieve Terminal Data

Retrieves the Terminal Data for CTLS. This is configuration group 0 (Tag FFEE - > FFEE0100). The terminal data can also be retrieved by `ctls_getConfiguraitonGroup(0)`.

Parameters

<i>tlv</i>	Response returned as a TLV
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with `errorCode.getErrorString()`

21.18.1.16 RETURN_CODE IDTechSDK.IDT_Vendi.ctls_setApplicationData (byte[] tlv, string ident = " ")

Set Application Data by AID

Sets the Application Data for CTLS as specified by TLV data

Parameters

<i>tlv</i>	Application data in TLV format The first tag of the TLV data must be the group number (FFE4). The second tag of the TLV data must be the AID (9F06)
------------	---

Example valid TLV, for Group #2, AID a0000000035010: "ffe401029f0607a0000000051010ffe10101ffe50110ffe30114ffe20106"

Parameters

<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.
--------------	--

Returns

RETURN_CODE: Values can be parsed with `errorCode.getErrorString()`

21.18.1.17 RETURN_CODE IDTechSDK.IDT_Vendi.ctls_setCAPK (byte[] key, string ident = " ")

Set Certificate Authority Public Key

Sets the CAPK for CTLS as specified by the CAKey structure

Parameters

<i>key</i>	CAKey format: [5 bytes RID][1 byte Index][1 byte Hash Algorithm][1 byte Encryption Algorithm][20 bytes HashValue][4 bytes Public Key Exponent][2 bytes Modulus Length][Variable bytes Modulus] Where: <ul style="list-style-type: none"> • Hash Algorithm: The only algorithm supported is SHA-1. The value is set to 0x01 • Encryption Algorithm: The encryption algorithm in which this key is used. Currently support only one type: RSA. The value is set to 0x01. • HashValue: Which is calculated using SHA-1 over the following fields: RID & Index & Modulus & Exponent • Public Key Exponent: Actually, the real length of the exponent is either one byte or 3 bytes. It can have two values: 3 (Format is 0x00 00 00 03), or 65537 (Format is 0x00 01 00 01, string ident = "") • Modulus Length: LenL LenH Indicated the length of the next field. • Modulus: This is the modulus field of the public key. Its length is specified in the field above.
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with `errorCode.getErrorString()`

21.18.1.18 RETURN_CODE IDTechSDK.IDT_Vendi.ctls_setConfigurationGroup (byte[] tlv, string ident = " ")

Set Configuration Group

Sets the Configuration Group for CTLS as specified by the TLV data

Parameters

<i>tlv</i>	Configuration Group Data in TLV format The first tag of the TLV data must be the group number (FFE4). A second tag must exist
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with `errorCode.getErrorString()`

21.18.1.19 RETURN_CODE IDTechSDK.IDT_Vendi.ctls_setDefaultConfiguration (string ident = " ")

Set Default Configuration Group

Resets the device to default CTLS configuration group settings

Parameters

<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.
--------------	--

Returns

RETURN_CODE: Values can be parsed with `errorCode.getErrorString()`

21.18.1.20 RETURN_CODE IDTechSDK.IDT_Vendi.ctls_setTerminalData (byte[] tlv, string ident = " ")

Set Terminal Data

Sets the Terminal Data for CTLS as specified by the TLV. The first TLV must be Configuration Group Number (Tag FFE4). The terminal global data is group 0, so the first TLV would be FFE40100. Other groups can be defined using this method (1 or greater), and those can be retrieved with `emv_getConfigurationGroup(int group)`, and deleted with `emv_removeConfigurationGroup(int group)`. You cannot delete group 0.

Parameters

<i>tlv</i>	TerminalData configuration data
------------	---------------------------------

Return values

<i>RETURN_CODE</i>	Return codes listed as typedef enum in IDTCommon:RETURN_CODE. Values can be parsed with IDT_BTPay::device_getResponseCodeString:(string ident = "")
--------------------	---

21.18.1.21 RETURN_CODE IDTechSDK.IDT_Vendi.ctls_startTransaction (double amount, double amtOther, int exponent, int type, int timeout, byte[] tags, bool isFastEMV = false, string ident = " ")

Start CTLS Transaction Request

The tags will be returned in the callback routine.

Parameters

<i>amount</i>	Transaction amount value (tag value 9F02, string ident = "")
<i>amtOther</i>	Other amount value, if any (tag value 9F03, string ident = "")
<i>type</i>	Transaction type (tag value 9C).
<i>timeout</i>	Timeout value in seconds.
<i>tags</i>	Any other tags to be included in the request. Passed as TLV Data. Example, tag 9F02 with amount 0x000000000100 would be 0x9F0206000000000100 If tags 9F02 (amount), 9F03 (other amount), or 9C (transaction type) are included, they will take priority over these values supplied as individual parameters to this method.
<i>isFastEMV</i>	If TRUE, it will populate the IDTTransactionData.fastEMV with ASCII data similar to IDTech FastEMV KB output, after performing an auto-authenticate and auto-complete with ResultCode = Could Not Contact Host
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with `errorCode.getErrorString()`

NOTE ON APPLEPAY VAS: To enable ApplePay VAS, first a merchant record must be defined in one of the six available index positions (1-6) using `device_setMerchantRecord`, then container tag FFEE06 must be sent as part of the additional tags parameter of `ctls_startTransaction`. Tag FFEE06 must contain tag 9F26 and 9F22, and can optionally contain tags 9F2B and DFO1. Example FFEE06189F220201009F2604000000009F2B050100000000DF010101 9F22 = two bytes = ApplePay Terminal Application Version Number. Hard defined as 0100 for now.

(required, string ident = "") 9F26 = four bytes = ApplePay Terminal Capabilities Information (required, string ident = "")

- Byte 1 = RFU
- Byte 2 = Terminal Type
 - - Bit 8 = VAS Support (1=on, 0 = off, string ident = "")
 - - Bit 7 = Touch ID Required (1=on, 0 = off, string ident = "")
 - - Bit 6 = RFU
 - - Bit 5 = RFU
 - - Bit 1,2,3,4
 - - - 0 = Payment Terminal
 - - - 1 = Transit Terminal
 - - - 2 = Access Terminal
 - - - 3 = Wireless Handoff Terminal
 - - - 4 = App Handoff Terminal
 - - - 15 = Other Terminal
- Byte 3 = RFU
- Byte 4 = Terminal Mode
 - - 0 = ApplePay VAS OR ApplePay
 - - 1 = ApplePay VAS AND ApplePay
 - - 2 = ApplePay VAS ONLY
 - - 3 = ApplePay ONLY 9F2B = 5 bytes = ApplePay VAS Filter. Each byte filters for that specific merchant index (optional, string ident = "") DF01 = 1 byte = ApplePay VAS Protocol. (optional, string ident = "")
 - - Bit 1 : 1 = URL VAS, 0 = Full VAS
 - - Bit 2 : 1 = VAS Beeps, 0 = No VAS Beeps
 - - Bit 3 : 1 = Silent Comm Error, 2 = EMEA Comm Error
 - - Bit 4-8 : RFU

21.18.1.22 `RETURN_CODE IDTechSDK.IDT_Vendi.ctls_trySetTerminalData (byte[] tlv, ref byte[] rejectedTLV, ref byte[] convertedTLV, string ident = " ")`

Try to Set CTLS Terminal Data

Attempts to set the CTLS Terminal Data. Use this functions to attempt to set the terminal data to the device. This function will allow the cross-device tag compatible writing of IDTech Legacy/New tag definitions

Parameters

<i>tlv</i>	TerminalData TLV Data
<i>rejectedTLV</i>	Contains the tags (if any) that were rejected by the firmware
<i>convertedTLV</i>	Contains the tags (if any) that were converted and accepted by the firmware
<i>ident</i>	Optional identifier

Return values

<i>RETURN_CODE</i>	Return codes listed as typedef enum in IDTCommon:RETURN_CODE. Values can be parsed with IDT_BTPay::device_getResponseCodeString:()
--------------------	--

21.18.1.23 `RETURN_CODE IDTechSDK.IDT_Vendi.device_activateTransaction (int timeout, byte[] tags, bool isFastEMV = false, string ident = " ")`

Start CTLS Transaction Request

The tags will be returned in the callback routine.

Parameters

<i>timeout</i>	Timeout value in seconds.
<i>tags</i>	The tags to be included in the request. Passed as TLV Data. Example, tag 9F02 with amount 0x000000000100 would be 0x9F0206000000000100
<i>isFastEMV</i>	If TRUE, it will populate the IDTTransactionData.fastEMV with ASCII data similar to IDTech FastEMV KB output, after performing an auto-authenticate and auto-complete with ResultCode = Could Not Contact Host
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with errorCode.getErrorString()

NOTE ON APPLEPAY VAS: To enable ApplePay VAS, first a merchant record must be defined in one of the six available index positions (1-6) using device_setMerchantRecord, then container tag FFEE06 must be sent as part of the additional tags parameter of ctls_startTransaction. Tag FFEE06 must contain tag 9F26 and 9F22, and can optionally contain tags 9F2B and DFO1. Example FFEE06189F220201009F2604000000009F2B050100000000↵DF010101 9F22 = two bytes = ApplePay Terminal Application Version Number. Hard defined as 0100 for now. (required, string ident = "") 9F26 = four bytes = ApplePay Terminal Capabilities Information (required, string ident = "")

- Byte 1 = RFU
- Byte 2 = Terminal Type
- - Bit 8 = VAS Support (1=on, 0 = off, string ident = "")
- - Bit 7 = Touch ID Required (1=on, 0 = off, string ident = "")
- - Bit 6 = RFU
- - Bit 5 = RFU
- - Bit 1,2,3,4
- - - 0 = Payment Terminal
- - - 1 = Transit Terminal
- - - 2 = Access Terminal
- - - 3 = Wireless Handoff Terminal
- - - 4 = App Handoff Terminal
- - - 15 = Other Terminal

- Byte 3 = RFU
- Byte 4 = Terminal Mode
- - 0 = ApplePay VAS OR ApplePay
- - 1 = ApplePay VAS AND ApplePay
- - 2 = ApplePay VAS ONLY
- - 3 = ApplePay ONLY 9F2B = 5 bytes = ApplePay VAS Filter. Each byte filters for that specific merchant index (optional, string ident = "") DF01 = 1 byte = ApplePay VAS Protocol. (optional, string ident = "")
- - Bit 1 : 1 = URL VAS, 0 = Full VAS
- - Bit 2 : 1 = VAS Beeps, 0 = No VAS Beeps
- - Bit 3 : 1 = Silent Comm Error, 2 = EMEA Comm Error
- - Bit 4-8 : RFU

21.18.1.24 RETURN_CODE IDTechSDK.IDT_Vendi.device_controlLED (byte *indexLED*, byte *control*, string *ident* = " ")

Control LED

Controls the LED for the reader. This command will only operate in pass-through mode

Parameters

<i>indexLED</i>	For LED <ul style="list-style-type: none"> • 00: LED 0 (Power LED, leftmost LED) • 01: LED 1 • 02: LED 2 • 03: LED 3 • FF: All LED's
<i>control</i>	LED Status: <ul style="list-style-type: none"> • 00: LED Off • 01: LED On
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString

21.18.1.25 RETURN_CODE IDTechSDK.IDT_Vendi.device_controlUserInterface (byte[] *values*, string *ident* = " ")

Control User Interface

Controls the User Interface: Display, Beep, LED

@param values Four bytes to control the user interface
Byte[0] = LCD Message
Messages 00-07 are normally controlled by the reader.

- 00h: Idle Message (Welcome, string ident = "")
- 01h: Present card (Please Present Card, string ident = "")
- 02h: Time Out or Transaction cancel (No Card, string ident = "")
- 03h: Transaction between reader and card is in the middle (Processing...)
- 04h: Transaction Pass (Thank You, string ident = "")
- 05h: Transaction Fail (Fail, string ident = "")
- 06h: Amount (Amount \$ 0.00 Tap Card, string ident = "")
- 07h: Balance or Offline Available funds (Balance \$ 0.00) Messages 08-0B are controlled by the terminal
- 08h: Insert or Swipe card (Use Chip & PIN, string ident = "")
- 09h: Try Again(Tap Again, string ident = "")
- 0Ah: Tells the customer to present only one card (Present 1 card only)
- 0Bh: Tells the customer to wait for authentication/authorization (Wait)
- FFh: indicates the command is setting the LED/Buzzer only.

Byte[1] = Beep Indicator

- 00h: No beep
- 01h: Single beep
- 02h: Double beep
- 03h: Three short beeps
- 04h: Four short beeps
- 05h: One long beep of 200 ms
- 06h: One long beep of 400 ms
- 07h: One long beep of 600 ms
- 08h: One long beep of 800 ms

Byte[2] = LED Number

- 00h: LED 0 (Power LED) 01h: LED 1
- 02h: LED 2
- 03h: LED 3
- FFh: All LEDs

Byte[3] = LED Status

- 00h: LED Off
- 01h: LED On

Parameters

<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.
--------------	--

Returns

RETURN_CODE: Values can be parsed with `errorCode.getErrorString(string ident = "")`

21.18.1.26 `RETURN_CODE IDTechSDK.IDT_Vendi.device_getAnyRKIStatus (bool isMultiKey, ref string status, ref Dictionary< string, RKI_KEY_TYPE > keys, string ident = " ")`

Get RKI Status Polls the RKI servers to validate device RKI eligibility. Note: if device type is known in advance (production or demo device), it is more efficient to use `device_getRKIStatus` instead

Parameters

<i>isMultiKey</i>	True = MultiKey, False = SingleKey
<i>status</i>	RKI status returned as a string
<i>keys</i>	Returns all available keys
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with `errorCode.getErrorString()`. When no data is available, return code = RETURN_CODE_NO_DATA_AVAILABLE

21.18.1.27 RETURN_CODE IDTechSDK.IDT_Vendi.device_getConfigurationFromMemory (ref string *json*, string *ident* = " ")

Read Device Configuration From Memory Retrieves the ViVOconfig configuration data from memory after a successful device_readConfigurationToMemory

Parameters

<i>json</i>	Returns the .json file data that can be saved to disk
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with `errorCode.getErrorString()`. When no data is available, return code = RETURN_CODE_NO_DATA_AVAILABLE

21.18.1.28 RETURN_CODE IDTechSDK.IDT_Vendi.device_getFirmwareVersion (ref string *response*, string *ident* = " ")

Polls device for Firmware Version

Parameters

<i>response</i>	Response returned of Firmware Version
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with `errorCode.getErrorString(string ident = "")`

21.18.1.29 RETURN_CODE IDTechSDK.IDT_Vendi.device_getMerchantRecord (int *index*, ref byte[] *record*, string *ident* = " ")

Get Merchant Record Gets the merchant record for ApplePay VAS

Parameters

<i>index</i>	Merchant Record index, valid values 1-6
<i>record</i>	Data returned containing 99 bytes: Byte 0 = Merchand Index Byte 1 = Merchant Enabled (1 = enabled, string ident = "") Byte 2 - 33 = Merchant Protocol Hash-256 value Byte 34 = Length of Merchant URL Bytes 35 - 99 = URL
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with `errorCode.getErrorString(string ident = "")`

21.18.1.30 RETURN_CODE IDTechSDK.IDT_Vendi.device_getRKIStatus (bool *isProd*, bool *isMultiKey*, ref string *status*, ref Dictionary< string, RKI_KEY_TYPE > *keys*, string *ident* = " ")

Get RKI Status From Specified Server Polls the RKI server to validate device RKI eligibility

Parameters

<i>isProd</i>	TRUE = poll production RKI server, FALSE = poll demo RKI server
<i>isMultiKey</i>	True = MultiKey, False = SingleKey
<i>status</i>	RKI status returned as a string
<i>keys</i>	Returns dictionary of available keys
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with `errorCode.getErrorString()`. When no data is available, return code = RETURN_CODE_NO_DATA_AVAILABLE

21.18.1.31 RETURN_CODE IDTechSDK.IDT_Vendi.device_getTransactionResults (ref IDTTTransactionData *results*, string *ident* = " ")

Get Transaction Results Gets the transaction results when the reader is functioning in "Auto Poll" mode

Parameters

<i>results</i>	The transaction results
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with `errorCode.getErrorString()`. When no data is available, return code = RETURN_CODE_NO_DATA_AVAILABLE

21.18.1.32 RETURN_CODE IDTechSDK.IDT_Vendi.device_pingDevice (string *ident* = " ")

Ping Device

Pings the reader. If connected, returns success. Otherwise, returns timeout.

Parameters

<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.
--------------	--

Returns

RETURN_CODE: Values can be parsed with `errorCode.getErrorString(string ident = "")`

21.18.1.33 RETURN_CODE IDTechSDK.IDT_Vendi.device_readConfigurationToMemory (string *memo*, Configs.InstallRules *rules*, List< Configs.DeviceCommand > *cmds*, string *ident* = " ", bool *isForeground* = false)

Read Device Configuration To Memory Executes a ViVOconfig read of device settings and stores the results in memory

Once a ViVOconfig read starts, all messages will be returned to the callback as `DeviceState.ViVOconfig`. The ViVOconfig operation is complete when the return code from a `DeviceState.ViVOconfig` message is one of the following values:

- RETURN_CODE_SDK_VIVOCONFIG_READ_CONFIG_SUCCESS = Read process completed successfully. Execute device_getConfigurationFromMemory to retrieve the configuration data
- RETURN_CODE_SDK_VIVOCONFIG_READ_CONFIG_FAILED = Read process FAILED.

Parameters

<i>memo</i>	Optional: populates the memo field of the captured .json file
<i>rules</i>	Optional: populates the .json file with available install rules
<i>cmds</i>	Optional: populates the .json file with commands to execute
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.
<i>isForeground</i>	If TRUE, will block program until update complete. Otherwise, FALSE performs update on background.

Returns

RETURN_CODE: Values can be parsed with `errorCode.getErrorString()`. When no data is available, return code = RETURN_CODE_NO_DATA_AVAILABLE

21.18.1.34 RETURN_CODE IDTechSDK.IDT_Vendi.device_RemoteKeyInjection (RKL_KEY_TYPE *type*, string *keyName*, string *ident* = " ", bool *performOnForeground* = false)

Remote Key Injection Performs a remote key injection for the device

Parameters

<i>type</i>	Remote Key Injection Type
<i>keyName</i>	Name of key (optional)
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.
<i>performOnForeground</i>	TRUE = block during process (default is false)

Returns

RETURN_CODE: Values can be parsed with `errorCode.getErrorString()`. When no data is available, return code = RETURN_CODE_NO_DATA_AVAILABLE

21.18.1.35 RETURN_CODE IDTechSDK.IDT_Vendi.device_retrieveAIDList (ref byte *response*[[]], string *ident* = " ")

Retrieve AID list

Returns all the AID names and their assigned groups installed on the terminal for CTLS/CONTACT.

Parameters

<i>response</i>	array of TLV data objects: FFE4 (group name) followed by 9F06 (AID), and DFEE4F (Interface 01 = CTLS, 02 = CONTACT, string <i>ident</i> = "")
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with `errorCode.getErrorString(string ident = "")`

21.18.1.36 `RETURN_CODE IDTechSDK.IDT_Vendi.device_sendConfiguration (string filename, VC_OPERATION_TYPE type, bool matchFW, string ident = " ", bool isForeground = false)`

Send Configuration Executes a ViVOconfig update or verify to a device

Once a ViVOconfig write or verify starts, all messages will be returned to the callback as DeviceState.ViVOconfig. The ViVOconfig operation is complete when the return code from a DeviceState.ViVOconfig message is one of the following values:

- `RETURN_CODE_SDK_VIVOCONFIG_VERIFY_SUCCESS` = Verification process completed successfully. No differences found.
- `RETURN_CODE_SDK_VIVOCONFIG_VERIFY_SUCCESS_WITH_WARNING` = Verification process completed with warnings.
- `RETURN_CODE_SDK_VIVOCONFIG_VERIFY_FAILED` = Verification process FAILED
- `RETURN_CODE_SDK_VIVOCONFIG_WRITE_SUCCESS` = Write process completed successfully.
- `RETURN_CODE_SDK_VIVOCONFIG_WRITE_SUCCESS_WITH_WARNING` = Write process completed with warnings
- `RETURN_CODE_SDK_VIVOCONFIG_WRITE_FAILED` = Write process FAILED

Parameters

<i>filename</i>	The .json configuration file to update the device with NOTE: You can also use this parameter to pass the path of where the .json file is located at on local storage instead
<i>filename</i>	The name of the .json configuration file
<i>type</i>	<code>VC_OPERATION_TYPE</code>

- `VC_OPERATION_TYPE_WRITE` = Write File To Device, Hash must validate
- `VC_OPERATION_TYPE_VERIFY` = Verify Device With File
- `VC_OPERATION_TYPE_WRITE_IGNORE_HASH` = Write File To Device, Ignore Hash
- `VC_OPERATION_TYPE_WRITE_FIX_HASH` = Write File To Device, Fix Hash If Necessary

Parameters

<i>matchFW</i>	TRUE = Device FW must match file FW, FALSE = Don't validate device FW against file
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.
<i>isForeground</i>	If TRUE, will block program until update complete. Otherwise, FALSE performs update on background.

Returns

`RETURN_CODE`: Values can be parsed with `errorCode.getErrorString()`. When no data is available, return code = `RETURN_CODE_NO_DATA_AVAILABLE`

21.18.1.37 `RETURN_CODE IDTechSDK.IDT_Vendi.device_sendConfigurationFromZip (byte[] zip, string filename, VC_OPERATION_TYPE type, bool matchFW, string ident = " ", bool isForeground = false)`

Send Configuration From Zip Executes a ViVOconfig update or verify to a device using files from a .zip archive

Once a ViVOconfig write or update starts, all messages will be returned to the callback as DeviceState.ViVOconfig. The ViVOconfig operation is complete when the return code from a DeviceState.ViVOconfig message is one of the following values:

- `RETURN_CODE_SDK_VIVOCONFIG_VERIFY_SUCCESS` = Verification process completed successfully. No differences found.
- `RETURN_CODE_SDK_VIVOCONFIG_VERIFY_SUCCESS_WITH_WARNING` = Verification process completed with warnings.
- `RETURN_CODE_SDK_VIVOCONFIG_VERIFY_FAILED` = Verification process FAILED
- `RETURN_CODE_SDK_VIVOCONFIG_WRITE_SUCCESS` = Write process completed successfully.
- `RETURN_CODE_SDK_VIVOCONFIG_WRITE_SUCCESS_WITH_WARNING` = Write process completed with warnings
- `RETURN_CODE_SDK_VIVOCONFIG_WRITE_FAILED` = Write process FAILED

Parameters

<i>zip</i>	.Zip archive containing the .json configuration file and also any associated image assets NOTE: You can also use this parameter to pass the path of where the .zip file is located at on local storage instead
<i>filename</i>	The name of the .json configuration file
<i>type</i>	<code>VC_OPERATION_TYPE</code>

- `VC_OPERATION_TYPE_WRITE` = Write File To Device, Hash must validate
- `VC_OPERATION_TYPE_VERIFY` = Verify Device With File
- `VC_OPERATION_TYPE_WRITE_IGNORE_HASH` = Write File To Device, Ignore Hash
- `VC_OPERATION_TYPE_WRITE_FIX_HASH` = Write File To Device, Fix Hash If Necessary

Parameters

<i>matchFW</i>	<code>TRUE</code> = Device FW must match file FW, <code>FALSE</code> = Don't validate device FW against file
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.
<i>isForeground</i>	If <code>TRUE</code> , will block program until update complete. Otherwise, <code>FALSE</code> performs update on background.

Returns

`RETURN_CODE`: Values can be parsed with `errorCode.getErrorString()`. When no data is available, return code = `RETURN_CODE_NO_DATA_AVAILABLE`

21.18.1.38 `RETURN_CODE IDTechSDK.IDT_Vendi.device_sendDataCommand (string cmd, bool calcLRC, ref byte[] response, string ident = " ")`

Send a data command to the device

Sends a command to the device.

Parameters

<i>cmd</i>	String representation of command to execute
<i>calcLRC</i>	If <code>TRUE</code> , this will wrap command with start/length/lrc/sum/end: '{STX}{Len_Low}{Len_High} data {CheckLRC} {CheckSUM} {ETX}'
<i>response</i>	Response data
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString

21.18.1.39 RETURN_CODE IDTechSDK.IDT_Vendi.device_sendDataCommand_ext (string *cmd*, bool *calcLRC*, ref byte[] *response*, int *timeout*, bool *noResponse*, string *ident* = " ")

Send a data command to the device - extended

Sends a command to the device.

Parameters

<i>cmd</i>	String representation of command to execute
<i>calcLRC</i>	If TRUE, this will wrap command as NGA with start/length/lrc/sum/end: '{STX}{Len_Low}{Len_High} data {CheckLRC} {CheckSUM} {ETX}'
<i>response</i>	Response data
<i>timeout</i>	Timeout value waiting for response, in milliseconds (1000 = 1 second, string ident = "")
<i>noResponse</i>	if TRUE, this will not wait for a response and immediately return SUCCESS
<i>calcITP</i>	If TRUE, this will wrap command as ITP with start/end/lrc: '{STX} data {ETX}{CheckLRC}'
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString

21.18.1.40 RETURN_CODE IDTechSDK.IDT_Vendi.device_sendPAE (string *command*, ref string *response*, int *timeout*, string *ident* = " ")

Send Payment Application Engine Command

Executes a PAE command

Parameters

<i>command</i>	ASCII command string, should start with "*PAE"
<i>response</i>	command response
<i>timeout</i>	timeout waiting for PAE response
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with errorCode.getErrorString(string ident = "")

21.18.1.41 RETURN_CODE IDTechSDK.IDT_Vendi.device_sendVivoCommandP2 (byte *command*, byte *subCommand*, byte[] *data*, ref byte[] *response*, string *ident* = " ")

Send Vivo Command Protocol 2

Sends a protocol 2 command to Vivo readers (IDG/NEO, string ident = "")

Parameters

<i>command</i>	Command
<i>subCommand</i>	Sub-Command
<i>data</i>	Data. May be null
<i>response</i>	Response
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with `errorCode.getErrorString(string ident = "")`

21.18.1.42 RETURN_CODE IDTechSDK.IDT_Vendi.device_sendVivoCommandP2_ext (byte *command*, byte *subCommand*, byte[] *data*, ref byte[] *response*, int *timeout*, bool *noResponse*, string *ident* = " ")

Send Vivo Command Protocol 2 Extended

Sends a protocol 2 command to Vivo readers (IDG/NEO, string *ident* = "")

Parameters

<i>command</i>	Command
<i>subCommand</i>	Sub-Command
<i>data</i>	Data. May be null
<i>response</i>	Response
<i>timeout</i>	Timeout, in milliseconds (3000 = 3 seconds, string <i>ident</i> = "")
<i>noResponse</i>	TRUE = don't wait for response, FALSE = wait for response defined by timeout
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with `errorCode.getErrorString(string ident = "")`

21.18.1.43 RETURN_CODE IDTechSDK.IDT_Vendi.device_setBurstMode (byte *mode*, string *ident* = " ")

Send Burst Mode

Sets the burst mode for the device.

Parameters

<i>mode</i>	0 = OFF, 1 = Always On, 2 = Auto Exit
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with `errorCode.getErrorString(string ident = "")`

21.18.1.44 RETURN_CODE IDTechSDK.IDT_Vendi.device_setMerchantRecord (int *index*, bool *enabled*, string *merchantID*, string *merchantURL*, string *ident* = " ")

Set Merchant Record Sets the merchant record for ApplePay VAS

Parameters

<i>index</i>	Merchant Record index, valid values 1-6
<i>enabled</i>	Merchant Enabled/Valid flag
<i>merchantID</i>	Merchant unique identifier registered with Apple. Example com.idtechproducts.applePay
<i>merchantURL</i>	Merchant URL, when applicable
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with `errorCode.getErrorString(string ident = "")`

21.18.1.45 RETURN_CODE IDTechSDK.IDT_Vendi.device_setPollMode (byte *mode*, string *ident* = " ")

Send Poll Mode

Sets the poll mode for the device. Auto Poll keeps reader active, Poll On Demand only polls when requested by terminal

Parameters

<i>mode</i>	0 = Auto Poll, 1 = Poll On Demand
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with `errorCode.getErrorString(string ident = "")`

21.18.1.46 RETURN_CODE IDTechSDK.IDT_Vendi.device_startRKI (string *ident* = " ")

Start Remote Key Injection

Starts a remote key injection request with IDTech RKI servers.

Parameters

<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.
--------------	--

Returns

RETURN_CODE: Values can be parsed with `device_getResponseCodeString`

21.18.1.47 **RETURN_CODE** IDTechSDK.IDT_Vendi.device_startTransaction (double *amount*, double *amtOther*, int *exponent*, int *type*, int *timeout*, byte[] *tags*, bool *isFastEMV* = false, string *ident* = " ")

Start CTLS Transaction Request

The tags will be returned in the callback routine.

Parameters

<i>amount</i>	Transaction amount value (tag value 9F02, string ident = "")
<i>amtOther</i>	Other amount value, if any (tag value 9F03, string ident = "")
<i>type</i>	Transaction type (tag value 9C).
<i>timeout</i>	Timeout value in seconds.
<i>tags</i>	Any other tags to be included in the request. Passed as TLV Data. Example, tag 9F02 with amount 0x000000000100 would be 0x9F0206000000000100 If tags 9F02 (amount), 9F03 (other amount), or 9C (transaction type) are included, they will take priority over these values supplied as individual parameters to this method.
<i>isFastEMV</i>	If TRUE, it will populate the IDTTransactionData.fastEMV with ASCII data similar to IDTech FastEMV KB output, after performing an auto-authenticate and auto-complete with ResultCode = Could Not Contact Host
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with errorCode.getErrorString()

NOTE ON APPLEPAY VAS: To enable ApplePay VAS, first a merchant record must be defined in one of the six available index positions (1-6) using device_setMerchantRecord, then container tag FFEE06 must be sent as part of the additional tags parameter of ctls_startTransaction. Tag FFEE06 must contain tag 9F26 and 9F22, and can optionally contain tags 9F2B and DFO1. Example FFEE06189F220201009F2604000000009F2B050100000000↵DF010101 9F22 = two bytes = ApplePay Terminal Application Version Number. Hard defined as 0100 for now. (required, string ident = "") 9F26 = four bytes = ApplePay Terminal Capabilities Information (required, string ident = "")

- Byte 1 = RFU
- Byte 2 = Terminal Type
- - Bit 8 = VAS Support (1=on, 0 = off, string ident = "")
- - Bit 7 = Touch ID Required (1=on, 0 = off, string ident = "")
- - Bit 6 = RFU
- - Bit 5 = RFU
- - Bit 1,2,3,4
- - - 0 = Payment Terminal
- - - 1 = Transit Terminal
- - - 2 = Access Terminal
- - - 3 = Wireless Handoff Terminal
- - - 4 = App Handoff Terminal
- - - 15 = Other Terminal
- Byte 3 = RFU

- Byte 4 = Terminal Mode
- - 0 = ApplePay VAS OR ApplePay
- - 1 = ApplePay VAS AND ApplePay
- - 2 = ApplePay VAS ONLY
- - 3 = ApplePay ONLY 9F2B = 5 bytes = ApplePay VAS Filter. Each byte filters for that specific merchant index (optional, string ident = "") DF01 = 1 byte = ApplePay VAS Protocol. (optional, string ident = "")
- - Bit 1 : 1 = URL VAS, 0 = Full VAS
- - Bit 2 : 1 = VAS Beeps, 0 = No VAS Beeps
- - Bit 3 : 1 = Silent Comm Error, 2 = EMEA Comm Error
- - Bit 4-8 : RFU

21.18.1.48 `static RETURN_CODE IDTechSDK.IDT_Vendi.device_updateDeviceFirmware (byte[] firmwareData, string ident = " ") [static]`

Update Firmware

Updates the firmware .

Parameters

<i>firmwareData</i>	Signed binary data of a firmware file provided by IDTech
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with `errorCode.getErrorString(string ident = "")`

After you pass the `firmwareData` file, a new thread will start to execute the firmware download. You will receive status of the progress through callbacks to the `IDTechSDK.Callback()` delegate. The following parameters will be passed back:

- `sender = IDT_DEVICE_Types.IDT_DEVICE_AUGUSTA`
- `state = DeviceState.FirmwareUpdate`
- `transactionResultCode = status of the firmware update (starting, entering bootloader, applying update, block success, firmware success, string ident = "")`
- `data = File Progress. Four bytes, with bytes [0][1] = current block, and bytes [2][3] = total blocks. 0x00030010 = block 3 of 16`

Example code starting a firmware update

```
OpenFileDialog diag = new OpenFileDialog( ident);
diag.Filter = "NGA FW Files|*.fm";

if (diag.ShowDialog() == DialogResult.OK, string ident = "")
{
    byte[] file = File.ReadAllBytes(diag.FileName, ident);
    RETURN_CODE rt = IDT_Device.SharedController.device_updateDeviceFirmware(file, ident);
    if (rt == RETURN_CODE.RETURN_CODE_DO_SUCCESS, string ident = "")
    {
        //Was a success
    }
}
```

```

    else
    {
        //Error starting firmware download
    }
}

```

Example monitoring firmware update status / success

```

private void MessageCallBack(IDTechSDK.IDT_DEVICE_Types type, DeviceState state, byte[] data,
    IDTTransactionData cardData, EMV_Callback emvCallback, RETURN_CODE transactionResultCode, string ident = "")
{
    switch (state, string ident = "")
    {
        case DeviceState.FirmwareUpdate:
            switch (transactionResultCode, string ident = "")
            {
                case RETURN_CODE.RETURN_CODE_FW_STARTING_UPDATE:
                    SetOutputText("Starting Firmware Update\n", ident);
                    break;
                case RETURN_CODE.RETURN_CODE_DO_SUCCESS:
                    SetOutputText("Firmware Update Successful\n", ident);
                    break;
                case RETURN_CODE.RETURN_CODE_APPLYING_FIRMWARE_UPDATE:
                    SetOutputText("Applying Firmware Update...\n", ident);
                    break;
                case RETURN_CODE.RETURN_CODE_ENTERING_BOOTLOADER_MODE:
                    SetOutputText("Entering Bootloader Mode...\n", ident);
                    break;
                case RETURN_CODE.RETURN_CODE_BLOCK_TRANSFER_SUCCESS:

                    int start = data[0] * 0x100 + data[1];
                    int end = data[2] * 0x100 + data[3];

                    SetOutputText("Sent block " + start.ToString() + " of " + end.ToString() + "\n", ident)
;
                    break;
                default:
                    SetOutputText("Firmware Update Error Code: " + "0x" + String.Format("{0:X}", (ushort)
transactionResultCode) + ": " + IDTechSDK.errorCode.getErrorString(transactionResultCode) + "\r\n",
ident);
                    break;
            }
            break;
    }
}

```

21.18.1.49 `RETURN_CODE IDTechSDK.IDT_Vendi.device_updateFirmwareFromZip (byte[] zipfile, string ident = "", bool isForeground = false)`

Update Firmware From Zip

Performs one or more device firmware updates from firmware files passed as a compressed archive with a valid configuration file (.json format)

Parameters

<i>zipfile</i>	Zip file containing on or more firmware update files and a .json configuration file
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.
<i>isForeground</i>	If TRUE, will block program until firmware update complete. Otherwise, FALSE performs FW update on background.

Returns

`RETURN_CODE`: Values can be parsed with `errorCode.getErrorString(string ident = "")`

21.18.1.50 `static int IDTechSDK.IDT_Vendi.getCommandTimeout (string ident = "") [static]`

Get Command Timeout

Gets the default timeout (in milliseconds) waiting for a blocking command response

Return values

<i>time</i>	Time
-------------	------

21.18.1.51 static void IDTechSDK.IDT_Vendi.lcd_retrieveMessage (DisplayMessages.DISPLAY_MESSAGE_LANGUAGE *lang*, DisplayMessages.DISPLAY_MESSAGE_IDENTIFIER *id*, ref string *line1*, ref string *line2*) [static]

Retrieve LCD Message

Returns the string value for a message ID returned for LCD messaging

Parameters

<i>lang</i>	Language.
<i>id</i>	Message ID
<i>line1</i>	Line 1 string value
<i>line2</i>	Line 2 string value

21.18.1.52 RETURN_CODE IDTechSDK.IDT_Vendi.msr_cancelMSRSwipe (string *ident* = " ")

Disable MSR Swipe Cancels MSR swipe request.

Parameters

<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.
--------------	--

Returns

RETURN_CODE: Values can be parsed with `errorCode.getErrorString(string ident = "")`

21.18.1.53 RETURN_CODE IDTechSDK.IDT_Vendi.msr_startMSRSwipe (int *timeout*, string *ident* = " ")

Enable MSR Swipe

Enables MSR, waiting for swipe to occur. Allows track selection. Returns IDTMSRData instance to `deviceDelegate::swipeMSRData:(string ident = "")`

Parameters

<i>timeout</i>	Swipe Timeout Value
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with `errorCode.getErrorString(string ident = "")`

21.18.1.54 static String IDTechSDK.IDT_Vendi.SDK_Version () [static]

SDK Version

- All Devices

Returns the current version of SDK

Returns

Framework version

21.18.1.55 `static void IDTechSDK.IDT_Vendi.setCallback (Callback my_Callback) [static]`

Set Callback

Sets the class callback

21.18.1.56 `static void IDTechSDK.IDT_Vendi.setCallback (IntPtr my_Callback, SynchronizationContext context) [static]`

Set Callback

Sets the class callback

Parameters

<i>my_Callback</i>	The callback function to receive the response message from device. defined as follows. public unsafe delegate void MFCCallBack(Parameters parameters, ident);
<i>context</i>	The context of the UI thread

21.18.1.57 `static void IDTechSDK.IDT_Vendi.setCommandTimeout (int milliseconds, string ident = " ") [static]`

Set Command Timeout

Sets the default timeout (in milliseconds) waiting for a blocking command response

Parameters

<i>milliseconds</i>	Time
---------------------	------

21.18.1.58 `static bool IDTechSDK.IDT_Vendi.useSerialPort (int port) [static]`

Use Serial Port Interface

Instructs SDK to attempt to use the Serial Port for communication with [IDT_Vendi](#) using default baud rate

Parameters

<i>port</i>	Serial Port to use. Example COM1 = 1.
-------------	---------------------------------------

Returns

bool TRUE=successful, FALSE=failure

21.18.1.59 `static bool IDTechSDK.IDT_Vendi.useSerialPort (int port, int baud) [static]`

Use Serial Port Interface with baud rate

Instructs SDK to attempt to use the Serial Port for communication with [IDT_Vendi](#)

Parameters

<i>port</i>	Serial Port to use. Example COM1 = 1.
<i>baud</i>	Baud rate to override default. Example 115200;

Returns

bool TRUE=successful, FALSE=failure

21.18.1.60 `static bool IDTechSDK.IDT_Vendi.useSerialPortLinux (string path) [static]`

Use Serial Port Interface on Linux

Instructs SDK to attempt to use the Serial Port for communication with BTMag using default baud rate on Linux implementations

Parameters

<i>path</i>	Path to use. Example /dev/ttyUSB*
-------------	-----------------------------------

Returns

bool TRUE=successful, FALSE=failure

21.18.1.61 `static bool IDTechSDK.IDT_Vendi.useSerialPortLinux (string path, int baud) [static]`

Use Serial Port Interface on Linux with baud rate

Instructs SDK to attempt to use the Serial Port for communication with BTPay on Linux implementations

Parameters

<i>path</i>	Path to use. Example /dev/ttyUSB*
<i>baud</i>	Baud rate to override default. Example 115200;

Returns

bool TRUE=successful, FALSE=failure

21.18.2 Property Documentation

21.18.2.1 `IDT_Vendi IDTechSDK.IDT_Vendi.SharedController [static],[get]`

Singleton Instance

Establishes an singleton instance of [IDT_Vendi](#) class.

Returns

Instance of [IDT_Vendi](#)

The documentation for this class was generated from the following file:

- /Users/andy/Repo/universal-sdk/IDTechSDK_STD/DeviceHeaders/IDT_Vendi.cs

21.19 IDTechSDK.IDT_VendIII Class Reference

Public Member Functions

- RETURN_CODE [device_controlLED](#) (byte indexLED, byte control, string ident="")
- RETURN_CODE [device_retrieveAIDList](#) (ref byte[][] response, string ident="")
- RETURN_CODE [device_sendDataCommand](#) (string cmd, bool calcLRC, ref byte[] response, string ident="")
- RETURN_CODE [config_setBaudRate](#) (int baud, string ident="")
- RETURN_CODE [device_sendDataCommand_ext](#) (string cmd, bool calcLRC, ref byte[] response, int timeout, bool noResponse, string ident="")
- RETURN_CODE [device_getFirmwareVersion](#) (ref string response, string ident="")
- RETURN_CODE [device_getMerchantRecord](#) (int index, ref byte[] record, string ident="")
- RETURN_CODE [device_getTransactionResults](#) (ref IDTTransactionData results, string ident="")
- RETURN_CODE [config_getSerialNumber](#) (ref string response, string ident="")
- RETURN_CODE [device_pingDevice](#) (string ident="")
- RETURN_CODE [device_controlUserInterface](#) (byte[] values, string ident="")
- RETURN_CODE [device_sendVivoCommandP2](#) (byte command, byte subCommand, byte[] data, ref byte[] response, string ident="")
- RETURN_CODE [device_sendVivoCommandP2_ext](#) (byte command, byte subCommand, byte[] data, ref byte[] response, int timeout, bool noResponse, string ident="")
- RETURN_CODE [ctls_retrieveAIDList](#) (ref byte[][] response, string ident="")
- RETURN_CODE [ctls_getAllConfigurationGroups](#) (ref byte[][] response, string ident="")
- RETURN_CODE [ctls_retrieveApplicationData](#) (byte[] AID, ref byte[] tlv, string ident="")
- RETURN_CODE [ctls_removeApplicationData](#) (byte[] AID, string ident="")
- RETURN_CODE [ctls_setApplicationData](#) (byte[] tlv, string ident="")
- RETURN_CODE [ctls_setDefaultConfiguration](#) (string ident="")
- RETURN_CODE [ctls_setConfigurationGroup](#) (byte[] tlv, string ident="")
- RETURN_CODE [device_sendPAE](#) (string command, ref string response, int timeout, string ident="")
- RETURN_CODE [ctls_retrieveTerminalData](#) (ref byte[] tlv, string ident="")
- RETURN_CODE [ctls_setTerminalData](#) (byte[] tlv, string ident="")
- RETURN_CODE [ctls_trySetTerminalData](#) (byte[] tlv, ref byte[] rejectedTLV, ref byte[] convertedTLV, string ident="")
- RETURN_CODE [ctls_getConfigurationGroup](#) (int group, ref byte[] tlv, string ident="")
- RETURN_CODE [ctls_removeConfigurationGroup](#) (int group, string ident="")
- RETURN_CODE [ctls_setCAPK](#) (byte[] key, string ident="")
- RETURN_CODE [ctls_retrieveCAPK](#) (byte[] capk, ref byte[] key, string ident="")
- RETURN_CODE [ctls_removeCAPK](#) (byte[] capk, string ident="")
- RETURN_CODE [ctls_removeAllCAPK](#) (string ident="")
- RETURN_CODE [ctls_retrieveCAPKList](#) (ref byte[] keys, string ident="")
- RETURN_CODE [ctls_startTransaction](#) (double amount, double amtOther, int exponent, int type, int timeout, byte[] tags, bool isFastEMV=false, string ident="")
- RETURN_CODE [ctls_activateTransaction](#) (int timeout, byte[] tags, bool isFastEMV=false, string ident="")
- RETURN_CODE [emv_startTransaction](#) (double amount, double amtOther, int exponent, int type, int timeout, byte[] tags, bool forceOnline, byte[] returnTags, bool isFastEMV=false, string ident="")
- RETURN_CODE [emv_activateTransaction](#) (int timeout, byte[] tags, bool forceOnline, byte[] returnTags, bool isFastEMV=false, string ident="")

- RETURN_CODE [device_startTransaction](#) (double amount, double amtOther, int exponent, int type, int timeout, byte[] tags, bool isFastEMV=false, string ident="")
- RETURN_CODE [device_activateTransaction](#) (int timeout, byte[] tags, bool isFastEMV=false, string ident="")
- RETURN_CODE [emv_completeTransaction](#) (bool commError, byte[] authCode, byte[] iad, byte[] tlvScripts, byte[] tlv, string ident="")
- RETURN_CODE [ctls_cancelTransaction](#) (string ident="")
- RETURN_CODE [device_setBurstMode](#) (byte mode, string ident="")
- RETURN_CODE [device_setMerchantRecord](#) (int index, bool enabled, string merchantID, string merchantURL, string ident="")
- RETURN_CODE [device_setPollMode](#) (byte mode, string ident="")
- RETURN_CODE [device_startRKI](#) (string ident="")
- RETURN_CODE [msr_startMSRSwipe](#) (int timeout, string ident="")
- RETURN_CODE [msr_cancelMSRSwipe](#) (string ident="")
- RETURN_CODE [device_getRKIStatus](#) (bool isProd, bool isMultiKey, ref string status, ref Dictionary< string, RKI_KEY_TYPE > keys, string ident="")
- RETURN_CODE [device_getAnyRKIStatus](#) (bool isMultiKey, ref string status, ref Dictionary< string, RKI_KEY_TYPE > keys, string ident="")
- RETURN_CODE [device_updateFirmwareFromZip](#) (byte[] zipfile, string ident="", bool isForeground=false)
- RETURN_CODE [device_sendConfigurationFromZip](#) (byte[] zip, string filename, VC_OPERATION_TYPE type, bool matchFW, string ident="", bool isForeground=false)
- RETURN_CODE [device_sendConfiguration](#) (string filename, VC_OPERATION_TYPE type, bool matchFW, string ident="", bool isForeground=false)
- RETURN_CODE [device_readConfigurationToMemory](#) (string memo, Configs.InstallRules rules, List< Configs.DeviceCommand > cmds, string ident="", bool isForeground=false)
- RETURN_CODE [device_getConfigurationFromMemory](#) (ref string json, string ident="")
- RETURN_CODE [device_RemoteKeyInjection](#) (RKI_KEY_TYPE type, string keyName, string ident="", bool performOnForeground=false)

Static Public Member Functions

- static bool [useSerialPort](#) (int port)
- static bool [useSerialPort](#) (int port, int baud)
- static bool [useSerialPortLinux](#) (string path)
- static bool [useSerialPortLinux](#) (string path, int baud)
- static void [initSC](#) ()
- static void [setCallback](#) (CallBack my_Callback)
- static void [setCallback](#) (IntPtr my_Callback, SynchronizationContext context)
- static String [SDK_Version](#) ()
- static RETURN_CODE [device_updateDeviceFirmware](#) (byte[] firmwareData, string ident="")
- static void [lcd_retrieveMessage](#) (DisplayMessages.DISPLAY_MESSAGE_LANGUAGE lang, DisplayMessages.DISPLAY_MESSAGE_IDENTIFIER id, ref string line1, ref string line2)

Properties

- static [IDT_VendIII SharedController](#) [get]

21.19.1 Member Function Documentation

21.19.1.1 RETURN_CODE IDTechSDK.IDT_VendIII.config_getSerialNumber (ref string response, string ident = " ")

Polls device for Serial Number

Parameters

<i>response</i>	Returns Serial Number
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString

21.19.1.2 RETURN_CODE IDTechSDK.IDT_VendIII.config_setBaudRate (int *baud*, string *ident* = " ")

Set Baud Rate

Sets the buad rate for RS-232 communication.

Parameters

<i>baud</i>	<ul style="list-style-type: none"> • 4 = 9600 • 6 = 19200 • 7 = 38400 • 8 = 57600 • 9 = 115200
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString

21.19.1.3 RETURN_CODE IDTechSDK.IDT_VendIII.ctls_activateTransaction (int *timeout*, byte[] *tags*, bool *isFastEMV* = false, string *ident* = " ")

Start CTLS Transaction Request

The tags will be returned in the callback routine.

Parameters

<i>amount</i>	Transaction amount value (tag value 9F02, string ident = "")
<i>amtOther</i>	Other amount value, if any (tag value 9F03, string ident = "")
<i>type</i>	Transaction type (tag value 9C).
<i>timeout</i>	Timeout value in seconds.
<i>tags</i>	The tags to be included in the request. Passed as TLV Data. Example, tag 9F02 with amount 0x000000000100 would be 0x9F0206000000000100
<i>isFastEMV</i>	If TRUE, it will populate the IDTTransactionData.fastEMV with ASCII data similar to IDTech FastEMV KB output, after performing an auto-authenticate and auto-complete with ResultCode = Could Not Contact Host
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with `errorCode.getErrorString()`

NOTE ON APPLEPAY VAS: To enable ApplePay VAS, first a merchant record must be defined in one of the six available index positions (1-6) using `device_setMerchantRecord`, then container tag FFEE06 must be sent as part of the additional tags parameter of `ctls_startTransaction`. Tag FFEE06 must contain tag 9F26 and 9F22, and can optionally contain tags 9F2B and DFO1. Example FFEE06189F220201009F2604000000009F2B050100000000↵DF010101 9F22 = two bytes = ApplePay Terminal Application Version Number. Hard defined as 0100 for now. (required, string ident = "") 9F26 = four bytes = ApplePay Terminal Capabilities Information (required, string ident = "")

- Byte 1 = RFU
- Byte 2 = Terminal Type
 - - Bit 8 = VAS Support (1=on, 0 = off, string ident = "")
 - - Bit 7 = Touch ID Required (1=on, 0 = off, string ident = "")
 - - Bit 6 = RFU
 - - Bit 5 = RFU
 - - Bit 1,2,3,4
 - - - 0 = Payment Terminal
 - - - 1 = Transit Terminal
 - - - 2 = Access Terminal
 - - - 3 = Wireless Handoff Terminal
 - - - 4 = App Handoff Terminal
 - - - 15 = Other Terminal
- Byte 3 = RFU
- Byte 4 = Terminal Mode
 - - 0 = ApplePay VAS OR ApplePay
 - - 1 = ApplePay VAS AND ApplePay
 - - 2 = ApplePay VAS ONLY
 - - 3 = ApplePay ONLY 9F2B = 5 bytes = ApplePay VAS Filter. Each byte filters for that specific merchant index (optional, string ident = "") DFO1 = 1 byte = ApplePay VAS Protocol. (optional, string ident = "")
 - - Bit 1 : 1 = URL VAS, 0 = Full VAS
 - - Bit 2 : 1 = VAS Beeps, 0 = No VAS Beeps
 - - Bit 3 : 1 = Silent Comm Error, 2 = EMEA Comm Error
 - - Bit 4-8 : RFU

21.19.1.4 RETURN_CODE IDTechSDK.IDT_VendIII.ctls_cancelTransaction (string ident = " ")

Cancel EMV Transaction

Cancels the currently executing EMV transaction.

Parameters

<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.
--------------	--

Returns

RETURN_CODE: Values can be parsed with `errorCode.getErrorString()`

21.19.1.5 RETURN_CODE IDTechSDK.IDT_VendIII.ctls_getAllConfigurationGroups (ref byte *response*[[[]], string *ident* = " ")

Retrieve All Configuration Groups

Returns all the Configuration Groups installed on the terminal for CTLS

Parameters

<i>response</i>	array of CTLS groups as TLV bytes
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with `errorCode.getErrorString(string ident = "")`

21.19.1.6 RETURN_CODE IDTechSDK.IDT_VendIII.ctls_getConfigurationGroup (int *group*, ref byte[] *tlv*, string *ident* = " ")

Get Configuration Group

Retrieves the Configuration for the specified Group.

Parameters

<i>group</i>	Configuration Group
<i>tlv</i>	return data
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with `errorCode.getErrorString()`

21.19.1.7 RETURN_CODE IDTechSDK.IDT_VendIII.ctls_removeAllCAPK (string *ident* = " ")

Remove All Certificate Authority Public Key

Removes all the CAPK for CTLS

Parameters

<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.
--------------	--

Returns

RETURN_CODE: Values can be parsed with `errorCode.getErrorString(string ident = "")`

21.19.1.8 RETURN_CODE IDTechSDK.IDT_VendIII.ctls_removeApplicationData (byte[] AID, string ident = " ")

Remove Application Data by AID

Removes the Application Data for CTLS as specified by the AID name passed as a parameter

Parameters

<i>AID</i>	Name of ApplicationID Must be between 5 and 16 bytes
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with `errorCode.getErrorString(string ident = "")`

21.19.1.9 RETURN_CODE IDTechSDK.IDT_VendIII.ctls_removeCAPK (byte[] capk, string ident = " ")

Remove Certificate Authority Public Key

Removes the CAPK as specified by the RID/Index

Parameters

<i>capk</i>	6 byte CAPK = 5 bytes RID + 1 byte INDEX
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with `errorCode.getErrorString(string ident = "")`

21.19.1.10 RETURN_CODE IDTechSDK.IDT_VendIII.ctls_removeConfigurationGroup (int group, string ident = " ")

Remove Configuration Group

Removes the Configuration as specified by the Group. Must not by group 0

Parameters

<i>group</i>	Configuration Group
--------------	---------------------

Return values

<i>RETURN_CODE</i>	Return codes listed as typedef enum in IDTCommon:RETURN_CODE. Values can be parsed with <code>IDT_BTPay::device_getResponseCodeString:(string ident = "")</code>
--------------------	--

21.19.1.11 **RETURN_CODE** IDTechSDK.IDT_VendIII.ctls_retrieveAIDList (ref byte *response*[[], string *ident* = " ")

Retrieve AID list

Returns all the AID names and their assigned groups installed on the terminal for CTLS.

Parameters

<i>response</i>	array of 2-tag TLV data objects: FFE4 (group name) followed by 9F06 (AID, string <i>ident</i> = "")
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with `errorCode.getErrorString(string ident = "")`

21.19.1.12 **RETURN_CODE** IDTechSDK.IDT_VendIII.ctls_retrieveApplicationData (byte[] *AID*, ref byte[] *tlv*, string *ident* = " ")

Retrieve Application Data by AID

Retrieves the CTLS Application Data as specified by the AID name passed as a parameter.

Parameters

<i>AID</i>	Name of ApplicationID. Must be between 5 and 16 bytes
<i>tlv</i>	The TLV elements of the requested AID
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with `errorCode.getErrorString()`

21.19.1.13 **RETURN_CODE** IDTechSDK.IDT_VendIII.ctls_retrieveCAPK (byte[] *capk*, ref byte[] *key*, string *ident* = " ")

Retrieve Certificate Authority Public Key

Retrieves the CAPK for CTLS as specified by the RID/Index passed as a parameter.

Parameters

<i>capk</i>	6 bytes CAPK = 5 bytes RID + 1 byte Index
-------------	---

Parameters

<i>key</i>	<p>Response returned as a CAKey format: [5 bytes RID][1 byte Index][1 byte Hash Algorithm][1 byte Encryption Algorithm][20 bytes HashValue][4 bytes Public Key Exponent][2 bytes Modulus Length][Variable bytes Modulus] Where:</p> <ul style="list-style-type: none"> • Hash Algorithm: The only algorithm supported is SHA-1. The value is set to 0x01 • Encryption Algorithm: The encryption algorithm in which this key is used. Currently support only one type: RSA. The value is set to 0x01. • HashValue: Which is calculated using SHA-1 over the following fields: RID & Index & Modulus & Exponent • Public Key Exponent: Actually, the real length of the exponent is either one byte or 3 bytes. It can have two values: 3 (Format is 0x00 00 00 03), or 65537 (Format is 0x00 01 00 01, string ident = "") • Modulus Length: LenL LenH Indicated the length of the next field. • Modulus: This is the modulus field of the public key. Its length is specified in the field above.
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with `errorCode.getErrorString()`

21.19.1.14 RETURN_CODE IDTechSDK.IDT_VendIII.ctls_retrieveCAPKList (ref byte[] keys, string ident = " ")

Retrieve the Certificate Authority Public Key list

Returns all the CAPK RID and Index installed on the terminal for CTLS.

Parameters

<i>keys</i>	[key1][key2]...[keyn], each key 6 bytes where key = 5 bytes RID + 1 byte index
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with `errorCode.getErrorString()`

21.19.1.15 RETURN_CODE IDTechSDK.IDT_VendIII.ctls_retrieveTerminalData (ref byte[] tlv, string ident = " ")

Retrieve Terminal Data

Retrieves the Terminal Data for CTLS. This is configuration group 0 (Tag FFEE - > FFEE0100). The terminal data can also be retrieved by `ctls_getConfiguraitonGroup(0)`.

Parameters

<i>tlv</i>	Response returned as a TLV
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with `errorCode.getErrorString()`

21.19.1.16 RETURN_CODE IDTechSDK.IDT_VendIII.ctls_setApplicationData (byte[] tlv, string ident = " ")

Set Application Data by AID

Sets the Application Data for CTLS as specified by TLV data

Parameters

<i>tlv</i>	Application data in TLV format The first tag of the TLV data must be the group number (FFE4). The second tag of the TLV data must be the AID (9F06)
------------	---

Example valid TLV, for Group #2, AID a0000000035010: "ffe401029f0607a0000000051010ffe10101ffe50110ffe30114ffe20106"

Parameters

<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.
--------------	--

Returns

RETURN_CODE: Values can be parsed with `errorCode.getErrorString()`

21.19.1.17 RETURN_CODE IDTechSDK.IDT_VendIII.ctls_setCAPK (byte[] key, string ident = " ")

Set Certificate Authority Public Key

Sets the CAPK for CTLS as specified by the CAKey structure

Parameters

<i>key</i>	<p>CAKey format: [5 bytes RID][1 byte Index][1 byte Hash Algorithm][1 byte Encryption Algorithm][20 bytes HashValue][4 bytes Public Key Exponent][2 bytes Modulus Length][Variable bytes Modulus] Where:</p> <ul style="list-style-type: none"> • Hash Algorithm: The only algorithm supported is SHA-1. The value is set to 0x01 • Encryption Algorithm: The encryption algorithm in which this key is used. Currently support only one type: RSA. The value is set to 0x01. • HashValue: Which is calculated using SHA-1 over the following fields: RID & Index & Modulus & Exponent • Public Key Exponent: Actually, the real length of the exponent is either one byte or 3 bytes. It can have two values: 3 (Format is 0x00 00 00 03), or 65537 (Format is 0x00 01 00 01, string ident = "") • Modulus Length: LenL LenH Indicated the length of the next field. • Modulus: This is the modulus field of the public key. Its length is specified in the field above.
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with `errorCode.getErrorString()`

21.19.1.18 RETURN_CODE IDTechSDK.IDT_VendIII.ctls_setConfigurationGroup (byte[] *tlv*, string *ident* = " ")

Set Configuration Group

Sets the Configuration Group for CTLS as specified by the TLV data

Parameters

<i>tlv</i>	Configuration Group Data in TLV format The first tag of the TLV data must be the group number (FFE4). A second tag must exist
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with `errorCode.getErrorString()`

21.19.1.19 RETURN_CODE IDTechSDK.IDT_VendIII.ctls_setDefaultConfiguration (string *ident* = " ")

Set Default Configuration Group

Resets the device to default CTLS configuration group settings

Parameters

<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.
--------------	--

Returns

RETURN_CODE: Values can be parsed with `errorCode.getErrorString()`

21.19.1.20 RETURN_CODE IDTechSDK.IDT_VendIII.ctls_setTerminalData (byte[] *tlv*, string *ident* = " ")

Set Terminal Data

Sets the Terminal Data for CTLS as specified by the TLV. The first TLV must be Configuration Group Number (Tag FFE4). The terminal global data is group 0, so the first TLV would be FFE40100. Other groups can be defined using this method (1 or greater), and those can be retrieved with `emv_getConfigurationGroup(int group)`, and deleted with `emv_removeConfigurationGroup(int group)`. You cannot delete group 0.

Parameters

<i>tlv</i>	TerminalData configuration data
------------	---------------------------------

Return values

<i>RETURN_CODE</i>	Return codes listed as typedef enum in IDTCommon:RETURN_CODE. Values can be parsed with <code>IDT_BTPay::device_getResponseCodeString:(string ident = "")</code>
--------------------	--

21.19.1.21 **RETURN_CODE** IDTechSDK.IDT_VendIII.ctls_startTransaction (double *amount*, double *amtOther*, int *exponent*, int *type*, int *timeout*, byte[] *tags*, bool *isFastEMV* = false, string *ident* = " ")

Start CTLS Transaction Request

The tags will be returned in the callback routine.

Parameters

<i>amount</i>	Transaction amount value (tag value 9F02, string ident = "")
<i>amtOther</i>	Other amount value, if any (tag value 9F03, string ident = "")
<i>type</i>	Transaction type (tag value 9C).
<i>timeout</i>	Timeout value in seconds.
<i>tags</i>	Any other tags to be included in the request. Passed as TLV Data. Example, tag 9F02 with amount 0x000000000100 would be 0x9F0206000000000100 If tags 9F02 (amount), 9F03 (other amount), or 9C (transaction type) are included, they will take priority over these values supplied as individual parameters to this method.
<i>isFastEMV</i>	If TRUE, it will populate the IDTTransactionData.fastEMV with ASCII data similar to IDTech FastEMV KB output, after performing an auto-authenticate and auto-complete with ResultCode = Could Not Contact Host
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with errorCode.getErrorString()

NOTE ON APPLEPAY VAS: To enable ApplePay VAS, first a merchant record must be defined in one of the six available index positions (1-6) using device_setMerchantRecord, then container tag FFEE06 must be sent as part of the additional tags parameter of ctls_startTransaction. Tag FFEE06 must contain tag 9F26 and 9F22, and can optionally contain tags 9F2B and DFO1. Example FFEE06189F220201009F2604000000009F2B050100000000↵DF010101 9F22 = two bytes = ApplePay Terminal Application Version Number. Hard defined as 0100 for now. (required, string ident = "") 9F26 = four bytes = ApplePay Terminal Capabilities Information (required, string ident = "")

- Byte 1 = RFU
- Byte 2 = Terminal Type
- - Bit 8 = VAS Support (1=on, 0 = off, string ident = "")
- - Bit 7 = Touch ID Required (1=on, 0 = off, string ident = "")
- - Bit 6 = RFU
- - Bit 5 = RFU
- - Bit 1,2,3,4
- - - 0 = Payment Terminal
- - - 1 = Transit Terminal
- - - 2 = Access Terminal
- - - 3 = Wireless Handoff Terminal
- - - 4 = App Handoff Terminal
- - - 15 = Other Terminal
- Byte 3 = RFU

- Byte 4 = Terminal Mode
- - 0 = ApplePay VAS OR ApplePay
- - 1 = ApplePay VAS AND ApplePay
- - 2 = ApplePay VAS ONLY
- - 3 = ApplePay ONLY 9F2B = 5 bytes = ApplePay VAS Filter. Each byte filters for that specific merchant index (optional, string ident = "") DF01 = 1 byte = ApplePay VAS Protocol. (optional, string ident = "")
- - Bit 1 : 1 = URL VAS, 0 = Full VAS
- - Bit 2 : 1 = VAS Beeps, 0 = No VAS Beeps
- - Bit 3 : 1 = Silent Comm Error, 2 = EMEA Comm Error
- - Bit 4-8 : RFU

21.19.1.22 `RETURN_CODE IDTechSDK.IDT_VendIII.ctls_trySetTerminalData (byte[] tlv, ref byte[] rejectedTLV, ref byte[] convertedTLV, string ident = " ")`

Try to Set CTLS Terminal Data

Attempts to set the CTLS Terminal Data. Use this functions to attempt to set the terminal data to the device. This function will allow the cross-device tag compatible writing of IDTech Legacy/New tag definitions

Parameters

<i>tlv</i>	TerminalData TLV Data
<i>rejectedTLV</i>	Contains the tags (if any) that were rejected by the firmware
<i>convertedTLV</i>	Contains the tags (if any) that were converted and accepted by the firmware
<i>ident</i>	Optional identifier

Return values

<i>RETURN_CODE</i>	Return codes listed as typedef enum in IDTCommon:RETURN_CODE. Values can be parsed with IDT_BTPay::device_getResponseCodeString:()
--------------------	--

21.19.1.23 `RETURN_CODE IDTechSDK.IDT_VendIII.device_activateTransaction (int timeout, byte[] tags, bool isFastEMV = false, string ident = " ")`

Start a Transaction Request

The tags will be returned in the callback routine.

Parameters

<i>timeout</i>	Timeout value in seconds.
<i>tags</i>	The tags to be included in the request. Passed as TLV Data. Example, tag 9F02 with amount 0x000000000100 would be 0x9F0206000000000100 Note: To request tags to be included in default response, use tag DFEE1A, and specify tag list. Example four tags 9F02, 9F36, 95, 9F37 to be included in response = DFEE1A079F029F369F9F37
<i>isFastEMV</i>	If TRUE, it will populate the IDTTransactionData.fastEMV with ASCII data similar to IDTech FastEMV KB output, after performing an auto-authenticate and auto-complete with ResultCode = Could Not Contact Host
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString

21.19.1.24 RETURN_CODE IDTechSDK.IDT_VendIII.device_controlLED (byte *indexLED*, byte *control*, string *ident* = " ")

Control LED

Controls the LED for the reader. This command will only operate in pass-through mode

Parameters

<i>indexLED</i>	For LED <ul style="list-style-type: none"> • 00: LED 0 (Power LED, leftmost LED) • 01: LED 1 • 02: LED 2 • 03: LED 3 • FF: All LED's
<i>control</i>	LED Status: <ul style="list-style-type: none"> • 00: LED Off • 01: LED On
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString

21.19.1.25 RETURN_CODE IDTechSDK.IDT_VendIII.device_controlUserInterface (byte[] *values*, string *ident* = " ")

Control User Interface

Controls the User Interface: Display, Beep, LED

```
@param values Four bytes to control the user interface
Byte[0] = LCD Message
Messages 00-07 are normally controlled by the reader.
- 00h: Idle Message (Welcome, string ident = "")
- 01h: Present card (Please Present Card, string ident = "")
- 02h: Time Out or Transaction cancel (No Card, string ident = "")
- 03h: Transaction between reader and card is in the middle (Processing...)
- 04h: Transaction Pass (Thank You, string ident = "")
- 05h: Transaction Fail (Fail, string ident = "")
- 06h: Amount (Amount $ 0.00 Tap Card, string ident = "")
- 07h: Balance or Offline Available funds (Balance $ 0.00) Messages 08-0B are controlled by the terminal
- 08h: Insert or Swipe card (Use Chip & PIN, string ident = "")
- 09h: Try Again(Tap Again, string ident = "")
- 0Ah: Tells the customer to present only one card (Present 1 card only)
- 0Bh: Tells the customer to wait for authentication/authorization (Wait)
- FFh: indicates the command is setting the LED/Buzzer only.
Byte[1] = Beep Indicator
- 00h: No beep
- 01h: Single beep
- 02h: Double beep
- 03h: Three short beeps
- 04h: Four short beeps
```

- 05h: One long beep of 200 ms
- 06h: One long beep of 400 ms
- 07h: One long beep of 600 ms
- 08h: One long beep of 800 ms

Byte[2] = LED Number

- 00h: LED 0 (Power LED) 01h: LED 1
- 02h: LED 2
- 03h: LED 3
- FFh: All LEDs

Byte[3] = LED Status

- 00h: LED Off
- 01h: LED On

Parameters

<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.
--------------	--

Returns

RETURN_CODE: Values can be parsed with `errorCode.getErrorString(string ident = "")`

21.19.1.26 RETURN_CODE IDTechSDK.IDT_VendIII.device_getAnyRKIStatus (bool *isMultiKey*, ref string *status*, ref Dictionary< string, RKI_KEY_TYPE > *keys*, string *ident* = " ")

Get RKI Status Polls the RKI servers to validate device RKI eligibility. Note: if device type is known in advance (production or demo device), it is more efficient to use `device_getRKIStatus` instead

Parameters

<i>isMultiKey</i>	True = MultiKey, False = SingleKey
<i>status</i>	RKI status returned as a string
<i>keys</i>	Returns all available keys
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with `errorCode.getErrorString()`. When no data is available, return code = RETURN_CODE_NO_DATA_AVAILABLE

21.19.1.27 RETURN_CODE IDTechSDK.IDT_VendIII.device_getConfigurationFromMemory (ref string *json*, string *ident* = " ")

Read Device Configuration From Memory Retrieves the ViVOconfig configuration data from memory after a successful `device_readConfigurationToMemory`

Parameters

<i>json</i>	Returns the .json file data that can be saved to disk
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with `errorCode.getErrorString()`. When no data is available, return code = RETURN_CODE_NO_DATA_AVAILABLE

21.19.1.28 **RETURN_CODE** IDTechSDK.IDT_VendIII.device_getFirmwareVersion (ref string *response*, string *ident* = " ")

Polls device for Firmware Version

Parameters

<i>response</i>	Response returned of Firmware Version
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with `errorCode.getErrorString(string ident = "")`

21.19.1.29 **RETURN_CODE** IDTechSDK.IDT_VendIII.device_getMerchantRecord (int *index*, ref byte[] *record*, string *ident* = " ")

Get Merchant Record Gets the merchant record for ApplePay VAS

Parameters

<i>index</i>	Merchant Record index, valid values 1-6
<i>record</i>	Data returned containing 99 bytes: Byte 0 = Merchand Index Byte 1 = Merchant Enabled (1 = enabled, string ident = "") Byte 2 - 33 = Merchant Protocol Hash-256 value Byte 34 = Length of Merchant URL Bytes 35 - 99 = URL
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with `errorCode.getErrorString(string ident = "")`

21.19.1.30 **RETURN_CODE** IDTechSDK.IDT_VendIII.device_getRKIStatus (bool *isProd*, bool *isMultiKey*, ref string *status*, ref Dictionary< string, RKI_KEY_TYPE > *keys*, string *ident* = " ")

Get RKI Status From Specified Server Polls the RKI server to validate device RKI eligibility

Parameters

<i>isProd</i>	TRUE = poll production RKI server, FALSE = poll demo RKI server
<i>isMultiKey</i>	True = MultiKey, False = SingleKey
<i>status</i>	RKI status returned as a string
<i>keys</i>	Returns dictionary of available keys
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with `errorCode.getErrorString()`. When no data is available, return code = **RETURN_CODE_NO_DATA_AVAILABLE**

21.19.1.31 **RETURN_CODE** IDTechSDK.IDT_VendIII.device_getTransactionResults (ref IDTTransactionData *results*, string *ident* = " ")

Get Transaction Results Gets the transaction results when the reader is functioning in "Auto Poll" mode

Parameters

<i>results</i>	The transaction results
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with `errorCode.getErrorString()`. When no data is available, return code = RETURN_CODE_NO_DATA_AVAILABLE

21.19.1.32 RETURN_CODE IDTechSDK.IDT_VendIII.device_pingDevice (string *ident* = " ")

Ping Device

Pings the reader. If connected, returns success. Otherwise, returns timeout.

Parameters

<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.
--------------	--

Returns

RETURN_CODE: Values can be parsed with `errorCode.getErrorString(string ident = "")`

21.19.1.33 RETURN_CODE IDTechSDK.IDT_VendIII.device_readConfigurationToMemory (string *memo*, Configs.InstallRules *rules*, List< Configs.DeviceCommand > *cmds*, string *ident* = " ", bool *isForeground* = false)

Read Device Configuration To Memory Executes a ViVOconfig read of device settings and stores the results in memory

Once a ViVOconfig read starts, all messages will be returned to the callback as `DeviceState.ViVOconfig`. The ViVOconfig operation is complete when the return code from a `DeviceState.ViVOconfig` message is one of the following values:

- RETURN_CODE_SDK_VIVOCONFIG_READ_CONFIG_SUCCESS = Read process completed successfully. Execute `device_getConfigurationFromMemory` to retrieve the configuration data
- RETURN_CODE_SDK_VIVOCONFIG_READ_CONFIG_FAILED = Read process FAILED.

Parameters

<i>memo</i>	Optional: populates the memo field of the captured .json file
<i>rules</i>	Optional: populates the .json file with available install rules
<i>cmds</i>	Optional: populates the .json file with commands to execute
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.
<i>isForeground</i>	If TRUE, will block program until update complete. Otherwise, FALSE performs update on background.

Returns

RETURN_CODE: Values can be parsed with `errorCode.getErrorString()`. When no data is available, return code = RETURN_CODE_NO_DATA_AVAILABLE

21.19.1.34 `RETURN_CODE IDTechSDK.IDT_VendIII.device_RemoteKeyInjection (RKL_KEY_TYPE type, string keyName, string ident = " ", bool performOnForeground = false)`

Remote Key Injection Performs a remote key injection for the device

Parameters

<i>type</i>	Remote Key Injection Type
<i>keyName</i>	Name of key (optional)
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.
<i>performOnForeground</i>	TRUE = block during process (default is false)

Returns

RETURN_CODE: Values can be parsed with `errorCode.getErrorString()`. When no data is available, return code = RETURN_CODE_NO_DATA_AVAILABLE

21.19.1.35 `RETURN_CODE IDTechSDK.IDT_VendIII.device_retrieveAIDList (ref byte response[[]], string ident = " ")`

Retrieve AID list

Returns all the AID names and their assigned groups installed on the terminal for CTLS/CONTACT.

Parameters

<i>response</i>	array of TLV data objects: FFE4 (group name) followed by 9F06 (AID), and DFEE4F (Interface 01 = CTLS, 02 = CONTACT, string ident = "")
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with `errorCode.getErrorString(string ident = "")`

21.19.1.36 `RETURN_CODE IDTechSDK.IDT_VendIII.device_sendConfiguration (string filename, VC_OPERATION_TYPE type, bool matchFW, string ident = " ", bool isForeground = false)`

Send Configuration Executes a ViVOconfig update or verify to a device

Once a ViVOconfig write or verify starts, all messages will be returned to the callback as `DeviceState.ViVOconfig`. The ViVOconfig operation is complete when the return code from a `DeviceState.ViVOconfig` message is one of the following values:

- RETURN_CODE_SDK_VIVOCONFIG_VERIFY_SUCCESS = Verification process completed successfully. No differences found.
- RETURN_CODE_SDK_VIVOCONFIG_VERIFY_SUCCESS_WITH_WARNING = Verification process completed with warnings.
- RETURN_CODE_SDK_VIVOCONFIG_VERIFY_FAILED = Verification process FAILED
- RETURN_CODE_SDK_VIVOCONFIG_WRITE_SUCCESS = Write process completed successfully.
- RETURN_CODE_SDK_VIVOCONFIG_WRITE_SUCCESS_WITH_WARNING = Write process completed with warnings
- RETURN_CODE_SDK_VIVOCONFIG_WRITE_FAILED = Write process FAILED

Parameters

<i>filename</i>	The .json configuration file to update the device with NOTE: You can also use this parameter to pass the path of where the .json file is located at on local storage instead
<i>filename</i>	The name of the .json configuration file
<i>type</i>	VC_OPERATION_TYPE

- VC_OPERATION_TYPE_WRITE = Write File To Device, Hash must validate
- VC_OPERATION_TYPE_VERIFY = Verify Device With File
- VC_OPERATION_TYPE_WRITE_IGNORE_HASH = Write File To Device, Ignore Hash
- VC_OPERATION_TYPE_WRITE_FIX_HASH = Write File To Device, Fix Hash If Necessary

Parameters

<i>matchFW</i>	TRUE = Device FW must match file FW, FALSE = Don't validate device FW against file
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.
<i>isForeground</i>	If TRUE, will block program until update complete. Otherwise, FALSE performs update on background.

Returns

RETURN_CODE: Values can be parsed with `errorCode.getErrorString()`. When no data is available, return code = RETURN_CODE_NO_DATA_AVAILABLE

21.19.1.37 `RETURN_CODE IDTechSDK.IDT_VendIII.device_sendConfigurationFromZip (byte[] zip, string filename, VC_OPERATION_TYPE type, bool matchFW, string ident = "", bool isForeground = false)`

Send Configuration From Zip Executes a ViVOconfig update or verify to a device using files from a .zip archive

Once a ViVOconfig write or update starts, all messages will be returned to the callback as `DeviceState.ViVOconfig`. The ViVOconfig operation is complete when the return code from a `DeviceState.ViVOconfig` message is one of the following values:

- RETURN_CODE_SDK_VIVOCONFIG_VERIFY_SUCCESS = Verification process completed successfully. No differences found.
- RETURN_CODE_SDK_VIVOCONFIG_VERIFY_SUCCESS_WITH_WARNING = Verification process completed with warnings.
- RETURN_CODE_SDK_VIVOCONFIG_VERIFY_FAILED = Verification process FAILED
- RETURN_CODE_SDK_VIVOCONFIG_WRITE_SUCCESS = Write process completed successfully.
- RETURN_CODE_SDK_VIVOCONFIG_WRITE_SUCCESS_WITH_WARNING = Write process completed with warnings
- RETURN_CODE_SDK_VIVOCONFIG_WRITE_FAILED = Write process FAILED

Parameters

<i>zip</i>	.Zip archive containing the .json configuration file and also any associated image assets NOTE: You can also use this parameter to pass the path of where the .zip file is located at on local storage instead
<i>filename</i>	The name of the .json configuration file
<i>type</i>	VC_OPERATION_TYPE

- VC_OPERATION_TYPE_WRITE = Write File To Device, Hash must validate
- VC_OPERATION_TYPE_VERIFY = Verify Device With File
- VC_OPERATION_TYPE_WRITE_IGNORE_HASH = Write File To Device, Ignore Hash
- VC_OPERATION_TYPE_WRITE_FIX_HASH = Write File To Device, Fix Hash If Necessary

Parameters

<i>matchFW</i>	TRUE = Device FW must match file FW, FALSE = Don't validate device FW against file
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.
<i>isForeground</i>	If TRUE, will block program until update complete. Otherwise, FALSE performs update on background.

Returns

RETURN_CODE: Values can be parsed with `errorCode.getErrorString()`. When no data is available, return code = RETURN_CODE_NO_DATA_AVAILABLE

21.19.1.38 RETURN_CODE IDTechSDK.IDT_VendIII.device_sendDataCommand (string *cmd*, bool *calcLRC*, ref byte[] *response*, string *ident* = " ")

Send a data command to the device

Sends a command to the device.

Parameters

<i>cmd</i>	String representation of command to execute
<i>calcLRC</i>	If TRUE, this will wrap command with start/length/lrc/sum/end: '{STX}{Len_Low}{Len_High} data {CheckLRC} {CheckSUM} {ETX}'
<i>response</i>	Response data
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with `device_getResponseCodeString`

21.19.1.39 RETURN_CODE IDTechSDK.IDT_VendIII.device_sendDataCommand_ext (string *cmd*, bool *calcLRC*, ref byte[] *response*, int *timeout*, bool *noResponse*, string *ident* = " ")

Send a data command to the device - extended

Sends a command to the device.

Parameters

<i>cmd</i>	String representation of command to execute
<i>calcLRC</i>	If TRUE, this will wrap command as NGA with start/length/lrc/sum/end: '{STX}{Len_Low}{Len_High} data {CheckLRC} {CheckSUM} {ETX}'
<i>response</i>	Response data
<i>timeout</i>	Timeout value waiting for response, in milliseconds (1000 = 1 second, string <i>ident</i> = "")
<i>noResponse</i>	if TRUE, this will not wait for a response and immediately return SUCCESS
<i>calcITP</i>	If TRUE, this will wrap command as ITP with start/end/lrc: '{STX} data {ETX}{CheckLRC}'
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString

21.19.1.40 RETURN_CODE IDTechSDK.IDT_VendIII.device_sendPAE (string *command*, ref string *response*, int *timeout*, string *ident* = " ")

Send Payment Application Engine Command

Executes a PAE command

Parameters

<i>command</i>	ASCII command string, should start with "*PAE"
<i>response</i>	command response
<i>timeout</i>	timeout waiting for PAE response
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with errorCode.getErrorString(string ident = "")

21.19.1.41 RETURN_CODE IDTechSDK.IDT_VendIII.device_sendVivoCommandP2 (byte *command*, byte *subCommand*, byte[] *data*, ref byte[] *response*, string *ident* = " ")

Send Vivo Command Protocol 2

Sends a protocol 2 command to Vivo readers (IDG/NEO, string ident = "")

Parameters

<i>command</i>	Command
<i>subCommand</i>	Sub-Command
<i>data</i>	Data. May be null
<i>response</i>	Response
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with errorCode.getErrorString(string ident = "")

21.19.1.42 RETURN_CODE IDTechSDK.IDT_VendIII.device_sendVivoCommandP2_ext (byte *command*, byte *subCommand*, byte[] *data*, ref byte[] *response*, int *timeout*, bool *noResponse*, string *ident* = " ")

Send Vivo Command Protocol 2 Extended

Sends a protocol 2 command to Vivo readers (IDG/NEO, string ident = "")

Parameters

<i>command</i>	Command
<i>subCommand</i>	Sub-Command
<i>data</i>	Data. May be null

Parameters

<i>response</i>	Response
<i>timeout</i>	Timeout, in milliseconds (3000 = 3 seconds, string ident = "")
<i>noResponse</i>	TRUE = don't wait for response, FALSE = wait for response defined by timeout
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with `errorCode.getErrorString(string ident = "")`

21.19.1.43 RETURN_CODE IDTechSDK.IDT_VendIII.device_setBurstMode (byte mode, string ident = " ")

Send Burst Mode

Sets the burst mode for the device.

Parameters

<i>mode</i>	0 = OFF, 1 = Always On, 2 = Auto Exit
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with `errorCode.getErrorString(string ident = "")`

21.19.1.44 RETURN_CODE IDTechSDK.IDT_VendIII.device_setMerchantRecord (int index, bool enabled, string merchantID, string merchantURL, string ident = " ")

Set Merchant Record Sets the merchant record for ApplePay VAS

Parameters

<i>index</i>	Merchant Record index, valid values 1-6
<i>enabled</i>	Merchant Enabled/Valid flag
<i>merchantID</i>	Merchant unique identifier registered with Apple. Example com.idtechproducts.applePay
<i>merchantURL</i>	Merchant URL, when applicable
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with `errorCode.getErrorString(string ident = "")`

21.19.1.45 RETURN_CODE IDTechSDK.IDT_VendIII.device_setPollMode (byte mode, string ident = " ")

Send Poll Mode

Sets the poll mode for the device. Auto Poll keeps reader active, Poll On Demand only polls when requested by terminal

Parameters

<i>mode</i>	0 = Auto Poll, 1 = Poll On Demand
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with `errorCode.getErrorString(string ident = "")`

21.19.1.46 RETURN_CODE IDTechSDK.IDT_VendIII.device_startRKI (string *ident* = " ")

Start Remote Key Injection

Starts a remote key injection request with IDTech RKI servers.

Parameters

<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.
--------------	--

Returns

RETURN_CODE: Values can be parsed with `device_getResponseCodeString`

21.19.1.47 RETURN_CODE IDTechSDK.IDT_VendIII.device_startTransaction (double *amount*, double *amtOther*, int *exponent*, int *type*, int *timeout*, byte[] *tags*, bool *isFastEMV* = false, string *ident* = " ")

Start a Transaction Request

The tags will be returned in the callback routine.

Parameters

<i>amount</i>	Transaction amount value (tag value 9F02, string ident = "")
<i>amtOther</i>	Other amount value, if any (tag value 9F03, string ident = "")
<i>type</i>	Transaction type (tag value 9C).
<i>timeout</i>	Timeout value in seconds.
<i>tags</i>	Any other tags to be included in the request. Passed as TLV Data. Example, tag 9F02 with amount 0x000000000100 would be 0x9F0206000000000100 If tags 9F02 (amount), 9F03 (other amount), or 9C (transaction type) are included, they will take priority over these values supplied as individual parameters to this method. Note: To request tags to be included in default response, use tag DFEE1A, and specify tag list. Example four tags 9F02, 9F36, 95, 9F37 to be included in response = DFEE1A079F029F369F37
<i>isFastEMV</i>	If TRUE, it will populate the IDTTransactionData.fastEMV with ASCII data similar to IDTech FastEMV KB output, after performing an auto-authenticate and auto-complete with ResultCode = Could Not Contact Host
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with `device_getResponseCodeString`

21.19.1.48 `static RETURN_CODE IDTechSDK.IDT_VendIII.device_updateDeviceFirmware (byte[] firmwareData, string ident = "") [static]`

Update Firmware

Updates the firmware .

Parameters

<i>firmwareData</i>	Signed binary data of a firmware file provided by IDTech
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with `errorCode.getErrorString(string ident = "")`

After you pass the `firmwareData` file, a new thread will start to execute the firmware download. You will receive status of the progress through callbacks to the `IDTechSDK.Callback()` delegate. The following parameters will be passed back:

- `sender = IDT_DEVICE_Types.IDT_DEVICE_AUGUSTA`
- `state = DeviceState.FirmwareUpdate`
- `transactionResultCode = status of the firmware update (starting, entering bootloader, applying update, block success, firmware success, string ident = "")`
- `data = File Progress. Four bytes, with bytes [0][1] = current block, and bytes [2][3] = total blocks. 0x00030010 = block 3 of 16`

Example code starting a firmware update

```
OpenFileDialog diag = new OpenFileDialog( ident);
diag.Filter = "NGA FW Files (*.fm)";

if (diag.ShowDialog() == DialogResult.OK, string ident = "")
{
    byte[] file = File.ReadAllBytes(diag.FileName, ident);
    RETURN_CODE rt = IDT_Device.SharedController.device_updateDeviceFirmware(file, ident);
    if (rt == RETURN_CODE.RETURN_CODE_DO_SUCCESS, string ident = "")
    {
        //Was a success
    }
    else
    {
        //Error starting firmware download
    }
}
```

Example monitoring firmware update status / success

```
private void MessageCallBack(IDTechSDK.IDT_DEVICE_Types type, DeviceState state, byte[] data,
    IDTTransactionData cardData, EMV_Callback emvCallback, RETURN_CODE transactionResultCode, string ident = "")
{
    switch (state, string ident = "")
    {
        case DeviceState.FirmwareUpdate:
            switch (transactionResultCode, string ident = "")
            {
                case RETURN_CODE.RETURN_CODE_FW_STARTING_UPDATE:
                    SetOutputText("Starting Firmware Update\n", ident);
                    break;
                case RETURN_CODE.RETURN_CODE_DO_SUCCESS:
                    SetOutputText("Firmware Update Successful\n", ident);
                    break;
                case RETURN_CODE.RETURN_CODE_APPLYING_FIRMWARE_UPDATE:
                    break;
            }
        break;
    }
}
```

```

        SetOutputText("Applying Firmware Update...\n", ident);
        break;
    case RETURN_CODE.RETURN_CODE_ENTERING_BOOTLOADER_MODE:
        SetOutputText("Entering Bootloader Mode...\n", ident);
        break;
    case RETURN_CODE.RETURN_CODE_BLOCK_TRANSFER_SUCCESS:

        int start = data[0] * 0x100 + data[1];
        int end = data[2] * 0x100 + data[3];

        SetOutputText("Sent block " + start.ToString() + " of " + end.ToString() + "\n", ident)
;
        break;
    default:
        SetOutputText("Firmware Update Error Code: " + "0x" + String.Format("{0:X}", (ushort)
transactionResultCode) + ": " + IDTechSDK.errorCode.getErrorString(transactionResultCode) + "\r\n",
ident);
        break;
    }
    break;
}
}
}

```

21.19.1.49 RETURN_CODE IDTechSDK.IDT_VendIII.device_updateFirmwareFromZip (byte[] zipfile, string ident = "", bool isForeground = false)

Update Firmware From Zip

Performs one or more device firmware updates from firmware files passed as a compressed archive with a valid configuration file (.json format)

Parameters

<i>zipfile</i>	Zip file containing on or more firmware update files and a .json configuration file
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.
<i>isForeground</i>	If TRUE, will block program until firmware update complete. Otherwise, FALSE performs FW update on background.

Returns

RETURN_CODE: Values can be parsed with `errorCode.getErrorString(string ident = "")`

21.19.1.50 RETURN_CODE IDTechSDK.IDT_VendIII.emv_activateTransaction (int timeout, byte[] tags, bool forceOnline, byte[] returnTags, bool isFastEMV = false, string ident = "")

Start EMV Transaction Request

Authorizes the EMV transaction for an ICC card

The tags will be returned in the callback routine.

Parameters

<i>timeout</i>	Timeout value in seconds.
<i>tags</i>	The tags to be included in the request. Passed as TLV Data. Example, tag 9F02 with amount 0x000000000100 would be 0x9F0206000000000100
<i>forceOnline</i>	TRUE = do not allow offline approval, FALSE = allow ICC to approve offline if terminal capable Note: To request tags to be included in default response, use tag DFEE1A, and specify tag list. Example four tags 9F02, 9F36, 95, 9F37 to be included in response = DFEE1A079F029F369F37
<i>isFastEMV</i>	If TRUE, it will populate the <code>IDTTransactionData.fastEMV</code> with ASCII data similar to IDTech FastEMV KB output, after performing an auto-authenticate and auto-complete with <code>ResultCode = Could Not Contact Host</code>
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString

21.19.1.51 RETURN_CODE IDTechSDK.IDT_VendIII.emv_completeTransaction (bool *commError*, byte[] *authCode*, byte[] *iad*, byte[] *tlvScripts*, byte[] *tlv*, string *ident* = " ")

Complete EMV Transaction Request

Completes the EMV transaction for an ICC card when online authorization request is received from emv_← authenticateTransaction

The tags will be returned in the callback routine.

Parameters

<i>commError</i>	Communication error with host. Set to TRUE if host was unreachable, or FALSE if host response received. If Communication error, authCode, iad, tlvScripts can be null.
<i>authCode</i>	Authorization code from host. Two bytes. Example 0x3030. (Tag value 8A). Required
<i>iad</i>	Issuer Authentication Data, if any. Example 0x11223344556677883030 (tag value 91).
<i>tlvScripts</i>	71/72 scripts, if any
<i>tlv</i>	Additional TVL data to return with transaction results (if any, string ident = "")
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString

21.19.1.52 RETURN_CODE IDTechSDK.IDT_VendIII.emv_startTransaction (double *amount*, double *amtOther*, int *exponent*, int *type*, int *timeout*, byte[] *tags*, bool *forceOnline*, byte[] *returnTags*, bool *isFastEMV* = false, string *ident* = " ")

Start EMV Transaction Request

Authorizes the EMV transaction for an ICC card

The tags will be returned in the callback routine.

Parameters

<i>amount</i>	Transaction amount value (tag value 9F02, string ident = "")
<i>amtOther</i>	Other amount value, if any (tag value 9F03, string ident = "")
<i>type</i>	Transaction type (tag value 9C).
<i>timeout</i>	Timeout value in seconds.
<i>tags</i>	Any other tags to be included in the request. Passed as TLV Data. Example, tag 9F02 with amount 0x000000000100 would be 0x9F0206000000000100 If tags 9F02 (amount), 9F03 (other amount), or 9C (transaction type) are included, they will take priority over these values supplied as individual parameters to this method.
<i>forceOnline</i>	TRUE = do not allow offline approval, FALSE = allow ICC to approve offline if terminal capable Note: To request tags to be included in default response, use tag DFEE1A, and specify tag list. Example four tags 9F02, 9F36, 95, 9F37 to be included in response = DFEE1A079F029F369F37
<i>isFastEMV</i>	If TRUE, it will populate the IDTTransactionData.fastEMV with ASCII data similar to IDTech FastEMV KB output, after performing an auto-authenticate and auto-complete with ResultCode = Could Not Contact Host
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString

21.19.1.53 static void IDTechSDK.IDT_VendIII.lcd_retrieveMessage (DisplayMessages.DISPLAY_MESSAGE_LANGUAGE *lang*, DisplayMessages.DISPLAY_MESSAGE_IDENTIFIER *id*, ref string *line1*, ref string *line2*) [static]

Retrieve LCD Message

Returns the string value for a message ID returned for LCD messaging

Parameters

<i>lang</i>	Language.
<i>id</i>	Message ID
<i>line1</i>	Line 1 string value
<i>line2</i>	Line 2 string value
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString

21.19.1.54 RETURN_CODE IDTechSDK.IDT_VendIII.msr_cancelMSRSwipe (string *ident* = " ")

Disable MSR Swipe Cancels MSR swipe request.

Parameters

<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.
--------------	--

Returns

RETURN_CODE: Values can be parsed with errorCode.getErrorString(string ident = "")

21.19.1.55 RETURN_CODE IDTechSDK.IDT_VendIII.msr_startMSRSwipe (int *timeout*, string *ident* = " ")

Enable MSR Swipe

Enables MSR, waiting for swipe to occur. Allows track selection. Returns IDTMSRData instance to deviceDelegate::swipeMSRData:(string ident = "")

Parameters

<i>timeout</i>	Swipe Timeout Value
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with errorCode.getErrorString(string ident = "")

21.19.1.56 `static String IDTechSDK.IDT_VendIII.SDK_Version () [static]`

SDK Version

- All Devices

Returns the current version of SDK

Returns

Framework version

21.19.1.57 `static void IDTechSDK.IDT_VendIII.setCallback (CallBack my_Callback) [static]`

Set Callback

Sets the class callback

21.19.1.58 `static void IDTechSDK.IDT_VendIII.setCallback (IntPtr my_Callback, SynchronizationContext context) [static]`

Set Callback

Sets the class callback

Parameters

<i>my_Callback</i>	The callback function to receive the response message from device. defined as follows. public unsafe delegate void MFCCallBack(Parameters parameters, ident);
<i>context</i>	The context of the UI thread

21.19.1.59 `static bool IDTechSDK.IDT_VendIII.useSerialPort (int port) [static]`

Use Serial Port Interface

Instructs SDK to attempt to use the Serial Port for communication with [IDT_VendIII](#) using default baud rate

Parameters

<i>port</i>	Serial Port to use. Example COM1 = 1.
-------------	---------------------------------------

Returns

bool TRUE=successful, FALSE=failure

21.19.1.60 `static bool IDTechSDK.IDT_VendIII.useSerialPort (int port, int baud) [static]`

Use Serial Port Interface with baud rate

Instructs SDK to attempt to use the Serial Port for communication with [IDT_VendIII](#)

Parameters

<i>port</i>	Serial Port to use. Example COM1 = 1.
-------------	---------------------------------------

Parameters

<i>baud</i>	Baud rate to override default. Example 115200;
-------------	--

Returns

bool TRUE=successful, FALSE=failure

21.19.1.61 `static bool IDTechSDK.IDT_VendIII.useSerialPortLinux (string path) [static]`

Use Serial Port Interface on Linux

Instructs SDK to attempt to use the Serial Port for communication with BTMag using default baud rate on Linux implementations

Parameters

<i>path</i>	Path to use. Example /dev/ttyUSB*
-------------	-----------------------------------

Returns

bool TRUE=successful, FALSE=failure

21.19.1.62 `static bool IDTechSDK.IDT_VendIII.useSerialPortLinux (string path, int baud) [static]`

Use Serial Port Interface on Linux with baud rate

Instructs SDK to attempt to use the Serial Port for communication with BTPay on Linux implementations

Parameters

<i>path</i>	Path to use. Example /dev/ttyUSB*
<i>baud</i>	Baud rate to override default. Example 115200;

Returns

bool TRUE=successful, FALSE=failure

21.19.2 Property Documentation

21.19.2.1 `IDT_VendIII IDTechSDK.IDT_VendIII.SharedController [static], [get]`

Singleton Instance

Establishes an singleton instance of [IDT_VendIII](#) class.

Returns

Instance of [IDT_VendIII](#)

The documentation for this class was generated from the following file:

- /Users/randy/Repo/universal-sdk/IDTechSDK_STD/DeviceHeaders/IDT_VendIII.cs

21.20 IDTechSDK.IDT_VP3300 Class Reference

Public Member Functions

- RETURN_CODE [config_getSerialNumber](#) (ref string response, string ident="")
- RETURN_CODE [device_getFirmwareVersion](#) (ref string response, string ident="")
- RETURN_CODE [device_setQuickChipHIDMode](#) (byte mode, string ident="")
- RETURN_CODE [device_sendDataCommand](#) (string cmd, bool calcLRC, ref byte[] response, string ident="")
- RETURN_CODE [device_sendDataCommand_ext](#) (string cmd, bool calcLRC, ref byte[] response, int timeout, bool noResponse, string ident="")
- RETURN_CODE [msr_startMSRSwipe](#) (int timeout, string ident="")
- RETURN_CODE [msr_cancelMSRSwipe](#) (string ident="")
- RETURN_CODE [emv_cancelTransaction](#) (string ident="")
- RETURN_CODE [ctls_cancelTransaction](#) (string ident="")
- RETURN_CODE [emv_startTransaction](#) (double amount, double amtOther, int exponent, int type, int timeout, byte[] tags, bool forceOnline, bool isFastEMV=false, string ident="")
- RETURN_CODE [emv_activateTransaction](#) (int timeout, byte[] tags, bool forceOnline, bool isFastEMV=false, string ident="")
- RETURN_CODE [emv_authenticateTransaction](#) (byte[] updatedTLV, string ident="")
- RETURN_CODE [ctls_startTransaction](#) (double amount, double amtOther, int exponent, int type, int timeout, byte[] tags, bool forceOnline, bool isFastEMV=false, string ident="")
- RETURN_CODE [device_sendPAE](#) (string command, ref string response, int timeout, string ident="")
- RETURN_CODE [ctls_activateTransaction](#) (int timeout, byte[] tags, bool forceOnline, bool isFastEMV=false, string ident="")
- RETURN_CODE [device_startTransaction](#) (double amount, double amtOther, int exponent, int type, int timeout, byte[] tags, bool isFastEMV=false, string ident="")
- RETURN_CODE [device_activateTransaction](#) (int timeout, byte[] tags, bool isFastEMV=false, string ident="")
- RETURN_CODE [emv_completeTransaction](#) (bool commError, byte[] authCode, byte[] iad, byte[] tlvScripts, byte[] tlv, string ident="")
- RETURN_CODE [emv_callbackResponseLCD](#) (EMV_LCD_DISPLAY_MODE type, byte selection, string ident="")
- RETURN_CODE [emv_callbackResponsePIN](#) (EMV_PIN_MODE type, byte[] KSN, byte[] PIN, string ident="")
- RETURN_CODE [emv_callbackResponseMSR](#) (byte[] MSR, string ident="")
- RETURN_CODE [emv_getEMVKernelCheckValue](#) (ref string response, string ident="")
- RETURN_CODE [emv_getEMVConfigurationCheckValue](#) (ref string response, string ident="")
- RETURN_CODE [emv_getEMVKernelVersion](#) (ref string response, string ident="")
- RETURN_CODE [emv_retrieveTransactionResult](#) (byte[] tags, ref IDTTransactionData tlv, string ident="")
- RETURN_CODE [emv_removeApplicationData](#) (byte[] AID, string ident="")
- RETURN_CODE [ctls_removeApplicationData](#) (byte[] AID, string ident="")
- RETURN_CODE [emv_removeAllApplicationData](#) (string ident="")
- RETURN_CODE [emv_removeCAPK](#) (byte[] capk, string ident="")
- RETURN_CODE [ctls_removeCAPK](#) (byte[] capk, string ident="")
- RETURN_CODE [emv_removeAllCAPK](#) (string ident="")
- RETURN_CODE [ctls_removeAllCAPK](#) (string ident="")
- RETURN_CODE [emv_removeCRL](#) (byte[] crlList, string ident="")
- RETURN_CODE [emv_removeAllCRL](#) (string ident="")
- RETURN_CODE [emv_retrieveApplicationData](#) (byte[] AID, ref byte[] tlv, string ident="")
- RETURN_CODE [ctls_retrieveApplicationData](#) (byte[] AID, ref byte[] tlv, string ident="")
- RETURN_CODE [emv_retrieveAIDList](#) (ref byte[][] response, string ident="")
- RETURN_CODE [ctls_retrieveAIDList](#) (ref byte[][] response, string ident="")
- RETURN_CODE [emv_retrieveCAPK](#) (byte[] capk, ref byte[] key, string ident="")
- RETURN_CODE [ctls_retrieveCAPK](#) (byte[] capk, ref byte[] key, string ident="")
- RETURN_CODE [emv_retrieveCAPKList](#) (ref byte[] keys, string ident="")
- RETURN_CODE [ctls_retrieveCAPKList](#) (ref byte[] keys, string ident="")
- RETURN_CODE [emv_retrieveCRLList](#) (ref byte[] list, string ident="")

- RETURN_CODE [ctls_retrieveTerminalData](#) (ref byte[] tlv, string ident="")
- RETURN_CODE [emv_retrieveTerminalData](#) (ref byte[] tlv, string ident="")
- RETURN_CODE [emv_removeTerminalData](#) (string ident="")
- RETURN_CODE [emv_setApplicationData](#) (byte[] name, byte[] tlv, string ident="")
- RETURN_CODE [ctls_setApplicationData](#) (byte[] tlv, string ident="")
- RETURN_CODE [ctls_setDefaultConfiguration](#) (string ident="")
- RETURN_CODE [ctls_setConfigurationGroup](#) (byte[] tlv, string ident="")
- RETURN_CODE [emv_setCAPK](#) (byte[] key, string ident="")
- RETURN_CODE [ctls_setCAPK](#) (byte[] key, string ident="")
- RETURN_CODE [emv_setCRL](#) (byte[] list, string ident="")
- RETURN_CODE [emv_getTerminalMajorConfiguration](#) (ref int configuration, string ident="")
- RETURN_CODE [emv_setTerminalMajorConfiguration](#) (int configuration, string ident="")
- RETURN_CODE [emv_trySetTerminalData](#) (byte[] tlv, ref byte[] rejectedTLV, ref byte[] convertedTLV, bool overwrite=false, string ident="")
- RETURN_CODE [emv_setTerminalData](#) (byte[] tlv, string ident="")
- RETURN_CODE [emv_addTerminalData](#) (byte[] tlv, string ident="")
- RETURN_CODE [ctls_setTerminalData](#) (byte[] tlv, string ident="")
- RETURN_CODE [ctls_trySetTerminalData](#) (byte[] tlv, ref byte[] rejectedTLV, ref byte[] convertedTLV, string ident="")
- RETURN_CODE [device_pingDevice](#) (string ident="")
- RETURN_CODE [device_buzzer](#) (string ident="")
- RETURN_CODE [device_cancelTransaction](#) (string ident="")
- RETURN_CODE [device_setLED](#) (byte indexLED, byte control, string ident="")
- RETURN_CODE [device_controlUserInterface](#) (byte[] values, string ident="")
- RETURN_CODE [device_sendVivoCommandP2](#) (byte command, byte subCommand, byte[] data, ref byte[] response, string ident="")
- RETURN_CODE [device_sendVivoCommandP2_ext](#) (byte command, byte subCommand, byte[] data, ref byte[] response, int timeout, bool noResponse, string ident="")
- RETURN_CODE [ctls_getConfigurationGroup](#) (int group, ref byte[] tlv, string ident="")
- RETURN_CODE [ctls_removeConfigurationGroup](#) (int group, string ident="")
- RETURN_CODE [ctls_getAllConfigurationGroups](#) (ref byte[][] response, string ident="")
- RETURN_CODE [device_setBurstMode](#) (byte mode, string ident="")
- RETURN_CODE [device_setPollMode](#) (byte mode, string ident="")
- RETURN_CODE [device_getPollMode](#) (ref byte mode, string ident="")
- RETURN_CODE [device_setMerchantRecord](#) (int index, bool enabled, string merchantID, string merchantURL, string ident="")
- RETURN_CODE [device_startRKI](#) (string ident="")
- RETURN_CODE [icc_getICCReaderStatus](#) (ref byte status, string ident="")
- RETURN_CODE [icc_powerOffICC](#) (string ident="")
- RETURN_CODE [icc_powerOnICC](#) (ref byte[] ATR, byte interfaces, string ident="")
- RETURN_CODE [icc_exchangeAPDU](#) (string c_APDU, ref byte[] response, string ident="")
- RETURN_CODE [device_enablePassThrough](#) (bool enablePassThrough, string ident="")
- string [device_getResponseCodeString](#) (RETURN_CODE eCode)
- RETURN_CODE [device_getTransactionResults](#) (ref IDTTransactionData results, string ident="")
- RETURN_CODE [config_getDate](#) (ref int year, ref int month, ref int day, string ident="")
- RETURN_CODE [config_setDate](#) (int year, int month, int day, string ident="")
- RETURN_CODE [config_getTime](#) (ref int hour, ref int minute, string ident="")
- RETURN_CODE [config_setTime](#) (int hour, int minute, string ident="")
- RETURN_CODE [device_setDateTime](#) (string ident="")
- RETURN_CODE [device_getDateTime](#) (ref byte[] dateTime, string ident="")
- RETURN_CODE [device_StartRKI](#) (int type, string ident="")
- RETURN_CODE [device_getDRIReaderRiskPara](#) (byte index, ref byte[] tlv, string ident="")
- RETURN_CODE [config_setEncryptionControl](#) (bool msr, bool icc, string ident="")
- RETURN_CODE [config_getEncryptionControl](#) (ref bool msr, ref bool icc, string ident="")
- bool [createFastEMVData](#) (ref IDTTransactionData cData, string ident="")

- RETURN_CODE [device_controlLED](#) (byte indexLED, byte control, string ident="")
- RETURN_CODE [device_getMerchantRecord](#) (int index, ref byte[] record, string ident="")
- RETURN_CODE [device_rebootDevice](#) (string ident="")
- RETURN_CODE [device_getProductType](#) (ref byte[] type, string ident="")
- RETURN_CODE [device_getCashTranRiskPara](#) (ref byte[] tlv, string ident="")
- RETURN_CODE [device_getHardwareInfor](#) (ref string ascii, string ident="")
- RETURN_CODE [device_getModuleVer](#) (ref string moduleVer, string ident="")
- RETURN_CODE [device_getUIDofMCU](#) (ref string uid, string ident="")
- RETURN_CODE [device_getUsbBootLoader](#) (ref string bootLoader, string ident="")
- RETURN_CODE [device_getRKIStatus](#) (bool isProd, bool isMultiKey, ref string status, ref Dictionary< string, RKI_KEY_TYPE > keys, string ident="")
- RETURN_CODE [device_getAnyRKIStatus](#) (bool isMultiKey, ref string status, ref Dictionary< string, RKI_KEY_TYPE > keys, string ident="")
- RETURN_CODE [emv_getEMVKernelVersionExt](#) (ref string response, string ident="")
- RETURN_CODE [device_updateFirmwareFromZip](#) (byte[] zipfile, string ident="", bool isForeground=false)
- RETURN_CODE [device_sendConfigurationFromZip](#) (byte[] zip, string filename, VC_OPERATION_TYPE type, bool matchFW, string ident="", bool isForeground=false)
- RETURN_CODE [device_sendConfiguration](#) (string filename, VC_OPERATION_TYPE type, bool matchFW, string ident="", bool isForeground=false)
- RETURN_CODE [device_readConfigurationToMemory](#) (string memo, Configs.InstallRules rules, List< Configs.DeviceCommand > cmds, string ident="", bool isForeground=false)
- RETURN_CODE [device_getConfigurationFromMemory](#) (ref string json, string ident="")
- RETURN_CODE [device_RemoteKeyInjection](#) (RKI_KEY_TYPE type, string keyName, string ident="", bool performOnForeground=false)

Static Public Member Functions

- static void **initSC** ()
- static bool [useSerialPort](#) (int port)
- static bool [useSerialPort](#) (int port, int baud)
- static int [getCommandTimeout](#) (string ident="")
- static void [setCommandTimeout](#) (int milliseconds, string ident="")
- static void [setCallback](#) (CallBack my_Callback)
- static void [setCallback](#) (IntPtr my_Callback, SynchronizationContext context)
- static String [SDK_Version](#) ()
- static void [emv_autoAuthenticate](#) (bool authenticate, string ident="")
- static void [emv_autoAuthenticateTags](#) (bool authenticate, byte[] tags, string ident="")
- static void [emv_allowFallback](#) (bool allow, string ident="")
- static RETURN_CODE [device_updateDeviceFirmware](#) (byte[] firmwareData, string ident="")
- static void [lcd_retrieveMessage](#) (DisplayMessages.DISPLAY_MESSAGE_LANGUAGE lang, DisplayMessages.DISPLAY_MESSAGE_IDENTIFIER id, ref string line1, ref string line2)

Properties

- static [IDT_VP3300 SharedController](#) [get]

21.20.1 Detailed Description

Class for VP4800 reader

21.20.2 Member Function Documentation

21.20.2.1 RETURN_CODE IDTechSDK.IDT_VP3300.config_getDate (ref int *year*, ref int *month*, ref int *day*, string *ident* = " ")

Get Date Gets the date of the Real Time Clock

Parameters

<i>year</i>	Valid values 0 - 9999 (example 2019, string ident = "")
<i>month</i>	Valid values 1-12
<i>day</i>	Valid values 1-31
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with `errorCode.getErrorString()`. When no data is available, return code = RETURN_CODE_NO_DATA_AVAILABLE

21.20.2.2 RETURN_CODE IDTechSDK.IDT_VP3300.config_getEncryptionControl (ref bool *msr*, ref bool *icc*, string *ident* = " ")

Get Encryption Control

Get Encryption Control to switch status between MSR and ICC/EMV function. Following Encryption status supported:

- MSR ON, ICC/EMV ON,
- MSR ON, ICC/EMV OFF,
- MSR OFF, ICC/EMV OFF,

Parameters

<i>msr</i>	<ul style="list-style-type: none"> • true: enabled MSR with Encryption, • false: disabled MSR with Encryption,
<i>icc</i>	<ul style="list-style-type: none"> • true: enabled ICC with Encryption, • false: disabled ICC with Encryption,
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with `device_getResponseCodeString`

21.20.2.3 RETURN_CODE IDTechSDK.IDT_VP3300.config_getSerialNumber (ref string *response*, string *ident* = " ")

Polls device for Serial Number

Parameters

<i>response</i>	Returns Serial Number
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString

21.20.2.4 RETURN_CODE IDTechSDK.IDT_VP3300.config_getTime (ref int *hour*, ref int *minute*, string *ident* = " ")

Get Time Gets the time of the Real Time Clock

Parameters

<i>hour</i>	Valid values 0-23
<i>minute</i>	Valid values 0-59
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with errorCode.getErrorString(). When no data is available, return code = RETURN_CODE_NO_DATA_AVAILABLE

Set Time Sets the time of the Real Time Clock

Parameters

<i>hour</i>	Valid values 0-23
<i>minute</i>	Valid values 0-59
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with errorCode.getErrorString(). When no data is available, return code = RETURN_CODE_NO_DATA_AVAILABLE

21.20.2.5 RETURN_CODE IDTechSDK.IDT_VP3300.config_setDate (int *year*, int *month*, int *day*, string *ident* = " ")

Set Date Sets the date of the Real Time Clock

Parameters

<i>year</i>	Valid values 0 - 9999 (example 2019, string ident = "")
<i>month</i>	Valid values 1-12
<i>day</i>	Valid values 1-31
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with errorCode.getErrorString(). When no data is available, return code = RETURN_CODE_NO_DATA_AVAILABLE

21.20.2.6 RETURN_CODE IDTechSDK.IDT_VP3300.config_setEncryptionControl (bool *msr*, bool *icc*, string *ident* = " ")

Set Encryption Control

Set Encryption Control to switch status between MSR and ICC/EMV function. Following Encryption status supported:

- MSR ON, ICC/EMV ON,
- MSR ON, ICC/EMV OFF,
- MSR OFF, ICC/EMV OFF,

Parameters

<i>msr</i>	<ul style="list-style-type: none"> • true: enable MSR with Encryption, • false: disable MSR with Encryption,
<i>icc</i>	<ul style="list-style-type: none"> • true: enable ICC with Encryption, • false: disable ICC with Encryption,
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString

21.20.2.7 `bool IDTechSDK.IDT_VP3300.createFastEMVData (ref IDTTransactionData cData, string ident = " ")`

Create Fast EMV Data

At the completion of a Fast EMV Transaction, after the final card decision is returned and the IDTTransactionData object is provided, sending that cData object to this method will populate the .fastEMV element with string data that represents the Fast EMV data that would be returned from and IDTech FastEMV over KB protocol

Parameters

<i>cData</i>	The IDTTransactionData object populated with card data.
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with errorCode.getErrorString(string ident = "")

21.20.2.8 `RETURN_CODE IDTechSDK.IDT_VP3300.ctls_activateTransaction (int timeout, byte[] tags, bool forceOnline, bool isFastEMV = false, string ident = " ")`

Start a CTLS Transaction Request

Authorizes the CTLS transaction for an CTLS card

The tags will be returned in the callback routine.

Parameters

<i>amount</i>	Transaction amount value (tag value 9F02, string ident = "")
---------------	---

Parameters

<i>amtOther</i>	Other amount value, if any (tag value 9F03, string ident = "")
<i>exponent</i>	Number of characters after decimile point
<i>type</i>	Transaction type (tag value 9C).
<i>timeout</i>	Timeout value in seconds.
<i>tags</i>	The other tags to be included in the request. Passed as TLV Data. Example, tag 9F02 with amount 0x000000000100 would be 0x9F0206000000000100
<i>forceOnline</i>	TRUE = do not allow offline approval, FALSE = allow ICC to approve offline if terminal capable
<i>isFastEMV</i>	If TRUE, it will populate the IDTTransactionData.fastEMV with ASCII data similar to IDTech FastEMV KB output, after performing an auto-authenticate and auto-complete with ResultCode = Could Not Contact Host
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with `errorCode.getErrorString()`

NOTE ON APPLEPAY VAS: To enable ApplePay VAS, first a merchant record must be defined in one of the six available index positions (1-6) using `device_setMerchantRecord`, then container tag FFEE06 must be sent as part of the additional tags parameter of `ctls_startTransaction`. Tag FFEE06 must contain tag 9F26 and 9F22, and can optionally contain tags 9F2B and DFO1. Example FFEE06189F220201009F2604000000009F2B050100000000DFO10101 9F22 = two bytes = ApplePay Terminal Application Version Number. Hard defined as 0100 for now. (required, string ident = "") 9F26 = four bytes = ApplePay Terminal Capabilities Information (required, string ident = "")

- Byte 1 = RFU
- Byte 2 = Terminal Type
- - Bit 8 = VAS Support (1=on, 0 = off, string ident = "")
- - Bit 7 = Touch ID Required (1=on, 0 = off, string ident = "")
- - Bit 6 = RFU
- - Bit 5 = RFU
- - Bit 1,2,3,4
- - - 0 = Payment Terminal
- - - 1 = Transit Terminal
- - - 2 = Access Terminal
- - - 3 = Wireless Handoff Terminal
- - - 4 = App Handoff Terminal
- - - 15 = Other Terminal
- Byte 3 = RFU
- Byte 4 = Terminal Mode
- - 0 = ApplePay VAS OR ApplePay
- - 1 = ApplePay VAS AND ApplePay
- - 2 = ApplePay VAS ONLY

- - 3 = ApplePay ONLY 9F2B = 5 bytes = ApplePay VAS Filter. Each byte filters for that specific merchant index (optional, string ident = "") DF01 = 1 byte = ApplePay VAS Protocol. (optional, string ident = "")
- - Bit 1 : 1 = URL VAS, 0 = Full VAS
- - Bit 2 : 1 = VAS Beeps, 0 = No VAS Beeps
- - Bit 3 : 1 = Silent Comm Error, 2 = EMEA Comm Error
- - Bit 4-8 : RFU

21.20.2.9 RETURN_CODE IDTechSDK.IDT_VP3300.ctls_cancelTransaction (string ident = " ")

Cancel Transaction

Cancels the currently executing EMV or CTLS transaction.

Parameters

<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.
--------------	--

Returns

RETURN_CODE: Values can be parsed with `errorCode.getErrorString()`

21.20.2.10 RETURN_CODE IDTechSDK.IDT_VP3300.ctls_getAllConfigurationGroups (ref byte response[[]], string ident = " ")

Retrieve All Configuration Groups

Returns all the Configuration Groups installed on the terminal for CTLS

Parameters

<i>response</i>	array of CTLS groups as TLV bytes
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with `errorCode.getErrorString(string ident = "")`

21.20.2.11 RETURN_CODE IDTechSDK.IDT_VP3300.ctls_getConfigurationGroup (int group, ref byte[] tlv, string ident = " ")

Get Configuration Group

Retrieves the Configuration for the specified Group. Group 0 = terminal settings.

Parameters

<i>group</i>	Configuration Group
<i>tlv</i>	return data
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with `errorCode.getErrorString()`

21.20.2.12 RETURN_CODE IDTechSDK.IDT_VP3300.ctls_removeAllCAPK (string *ident* = " ")

Remove All Certificate Authority Public Key

Removes all the CAPK for CTLS

Parameters

<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.
--------------	--

Returns

RETURN_CODE: Values can be parsed with `errorCode.getErrorString(string ident = "")`

21.20.2.13 RETURN_CODE IDTechSDK.IDT_VP3300.ctls_removeApplicationData (byte[] *AID*, string *ident* = " ")

Remove Application Data by AID

Removes the Application Data for CTLS as specified by the AID name passed as a parameter

Parameters

<i>AID</i>	Name of ApplicationID Must be between 5 and 16 bytes
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with `errorCode.getErrorString(string ident = "")`

21.20.2.14 RETURN_CODE IDTechSDK.IDT_VP3300.ctls_removeCAPK (byte[] *capk*, string *ident* = " ")

Remove Certificate Authority Public Key

Removes the CAPK for CTLS as specified by the RID/Index

Parameters

<i>capk</i>	6 byte CAPK = 5 bytes RID + 1 byte INDEX
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with `errorCode.getErrorString(string ident = "")`

21.20.2.15 RETURN_CODE IDTechSDK.IDT_VP3300.ctls_removeConfigurationGroup (int *group*, string *ident* = " ")

Remove Configuration Group

Removes the Configuration as specified by the Group. Must not by group 0

Parameters

<i>group</i>	Configuration Group
--------------	---------------------

Return values

<i>RETURN_CODE</i>	Return codes listed as typedef enum in IDTechCommon:RETURN_CODE. Values can be parsed with IDT_BTPay::device_getResponseCodeString :(string ident = "")
--------------------	---

21.20.2.16 **RETURN_CODE** IDTechSDK.IDT_VP3300.ctls_retrieveAIDList (ref byte *response*[[], string *ident* = " ")

Retrieve AID list

Returns all the AID names and their assigned groups installed on the terminal for CTLS.

Parameters

<i>response</i>	array of TLV data objects: FFE4 (group name) followed by 9F06 (AID), and DFEE4F (Interface 01 = CTLS, 02 = CONTACT, string ident = "")
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with `errorCode.getErrorString(string ident = "")`

21.20.2.17 **RETURN_CODE** IDTechSDK.IDT_VP3300.ctls_retrieveApplicationData (byte[] *AID*, ref byte[] *tlv*, string *ident* = " ")

Retrieve Application Data by AID

Retrieves the CTLS Application Data as specified by the AID name passed as a parameter.

Parameters

<i>AID</i>	Name of ApplicationID. Must be between 5 and 16 bytes
<i>tlv</i>	The TLV elements of the requested AID
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with `errorCode.getErrorString()`

21.20.2.18 **RETURN_CODE** IDTechSDK.IDT_VP3300.ctls_retrieveCAPK (byte[] *capk*, ref byte[] *key*, string *ident* = " ")

Retrieve Certificate Authority Public Key

Retrieves the CAPKfor CTLS as specified by the RID/Index passed as a parameter.

Parameters

<i>capk</i>	6 bytes CAPK = 5 bytes RID + 1 byte Index
-------------	---

Parameters

<i>key</i>	<p>Response returned as a CAKey format: [5 bytes RID][1 byte Index][1 byte Hash Algorithm][1 byte Encryption Algorithm][20 bytes HashValue][4 bytes Public Key Exponent][2 bytes Modulus Length][Variable bytes Modulus] Where:</p> <ul style="list-style-type: none"> • Hash Algorithm: The only algorithm supported is SHA-1. The value is set to 0x01 • Encryption Algorithm: The encryption algorithm in which this key is used. Currently support only one type: RSA. The value is set to 0x01. • HashValue: Which is calculated using SHA-1 over the following fields: RID & Index & Modulus & Exponent • Public Key Exponent: Actually, the real length of the exponent is either one byte or 3 bytes. It can have two values: 3 (Format is 0x00 00 00 03), or 65537 (Format is 0x00 01 00 01, string ident = "") • Modulus Length: LenL LenH Indicated the length of the next field. • Modulus: This is the modulus field of the public key. Its length is specified in the field above.
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with `errorCode.getErrorString()`

21.20.2.19 RETURN_CODE IDTechSDK.IDT_VP3300.ctls_retrieveCAPKList (ref byte[] keys, string ident = " ")

Retrieve the Certificate Authority Public Key list

Returns all the CAPK RID and Index installed on the terminal for CTLS.

Parameters

<i>keys</i>	[key1][key2]...[keyn], each key 6 bytes where key = 5 bytes RID + 1 byte index
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with `errorCode.getErrorString()`

21.20.2.20 RETURN_CODE IDTechSDK.IDT_VP3300.ctls_retrieveTerminalData (ref byte[] tlv, string ident = " ")

Retrieve Terminal Data

Retrieves the Terminal Data for CTLS. This is configuration group 0 (Tag FFEE - > FFEE0100). The terminal data can also be retrieved by `ctls_getConfiguraitonGroup(0)`.

Parameters

<i>tlv</i>	Response returned as a TLV
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with `errorCode.getErrorString()`

21.20.2.21 RETURN_CODE IDTechSDK.IDT_VP3300.ctls_setApplicationData (byte[] tlv, string ident = " ")

Set Application Data by AID

Sets the Application Data for CTLS as specified by TLV data

Parameters

<i>tlv</i>	Application data in TLV format The first tag of the TLV data must be the group number (FFE4). The second tag of the TLV data must be the AID (9F06)
------------	---

Example valid TLV, for Group #2, AID a0000000035010: "ffe401029f0607a0000000051010ffe10101ffe50110ffe30114ffe20106"

Parameters

<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.
--------------	--

Returns

RETURN_CODE: Values can be parsed with `errorCode.getErrorString()`

21.20.2.22 RETURN_CODE IDTechSDK.IDT_VP3300.ctls_setCAPK (byte[] key, string ident = " ")

Set Certificate Authority Public Key

Sets the CAPK for CTLS as specified by the CAKey structure

Parameters

<i>key</i>	CAKey format: [5 bytes RID][1 byte Index][1 byte Hash Algorithm][1 byte Encryption Algorithm][20 bytes HashValue][4 bytes Public Key Exponent][2 bytes Modulus Length][Variable bytes Modulus] Where: <ul style="list-style-type: none"> • Hash Algorithm: The only algorithm supported is SHA-1. The value is set to 0x01 • Encryption Algorithm: The encryption algorithm in which this key is used. Currently support only one type: RSA. The value is set to 0x01. • HashValue: Which is calculated using SHA-1 over the following fields: RID & Index & Modulus & Exponent • Public Key Exponent: Actually, the real length of the exponent is either one byte or 3 bytes. It can have two values: 3 (Format is 0x00 00 00 03), or 65537 (Format is 0x00 01 00 01, string ident = "") • Modulus Length: LenL LenH Indicated the length of the next field. • Modulus: This is the modulus field of the public key. Its length is specified in the field above.
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with `errorCode.getErrorString()`

21.20.2.23 RETURN_CODE IDTechSDK.IDT_VP3300.ctls_setConfigurationGroup (byte[] tlv, string ident = " ")

Set Configuration Group

Sets the Configuration Group for CTLS as specified by the TLV data

Parameters

<i>tlv</i>	Configuration Group Data in TLV format The first tag of the TLV data must be the group number (FFE4). A second tag must exist
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with `errorCode.getErrorString()`

21.20.2.24 RETURN_CODE IDTechSDK.IDT_VP3300.ctls_setDefaultConfiguration (string ident = " ")

Set Default Configuration Group

Resets the device to default CTLS configuration group settings

Parameters

<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.
--------------	--

Returns

RETURN_CODE: Values can be parsed with `errorCode.getErrorString()`

21.20.2.25 RETURN_CODE IDTechSDK.IDT_VP3300.ctls_setTerminalData (byte[] tlv, string ident = " ")

Set Terminal Data

Sets the Terminal Data for CTLS as specified by the TLV. The first TLV must be Configuration Group Number (Tag FFE4). The terminal global data is group 0, so the first TLV would be FFE40100. Other groups can be defined using this method (1 or greater), and those can be retrieved with `emv_getConfigurationGroup(int group)`, and deleted with `emv_removeConfigurationGroup(int group)`. You cannot delete group 0.

Parameters

<i>tlv</i>	TerminalData configuration data
------------	---------------------------------

Return values

<i>RETURN_CODE</i>	Return codes listed as typedef enum in IDTCommon:RETURN_CODE. Values can be parsed with <code>IDT_BTPay::device_getResponseCodeString:(string ident = "")</code>
--------------------	--

21.20.2.26 **RETURN_CODE** IDTechSDK.IDT_VP3300.ctls_startTransaction (double *amount*, double *amtOther*, int *exponent*, int *type*, int *timeout*, byte[] *tags*, bool *forceOnline*, bool *isFastEMV* = false, string *ident* = " ")

Start a CTLS Transaction Request

Authorizes the CTLS transaction for an CTLS card

The tags will be returned in the callback routine.

Parameters

<i>amount</i>	Transaction amount value (tag value 9F02, string ident = "")
<i>amtOther</i>	Other amount value, if any (tag value 9F03, string ident = "")
<i>exponent</i>	Number of characters after decimile point
<i>type</i>	Transaction type (tag value 9C).
<i>timeout</i>	Timeout value in seconds.
<i>tags</i>	Any other tags to be included in the request. Passed as TLV Data. Example, tag 9F02 with amount 0x000000000100 would be 0x9F0206000000000100 If tags 9F02 (amount),9F03 (other amount), or 9C (transaction type) are included, they will take priority over these values supplied as individual parameters to this method.
<i>forceOnline</i>	TRUE = do not allow offline approval, FALSE = allow ICC to approve offline if terminal capable
<i>isFastEMV</i>	If TRUE, it will populate the IDTTransactionData.fastEMV with ASCII data similar to IDTech FastEMV KB output, after performing an auto-authenticate and auto-complete with ResultCode = Could Not Contact Host
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with errorCode.getErrorString()

NOTE ON APPLEPAY VAS: To enable ApplePay VAS, first a merchant record must be defined in one of the six available index positions (1-6) using device_setMerchantRecord, then container tag FFEE06 must be sent as part of the additional tags parameter of ctls_startTransaction. Tag FFEE06 must contain tag 9F26 and 9F22, and can optionally contain tags 9F2B and DFO1. Example FFEE06189F220201009F2604000000009F2B050100000000DF010101 9F22 = two bytes = ApplePay Terminal Application Version Number. Hard defined as 0100 for now. (required, string ident = "") 9F26 = four bytes = ApplePay Terminal Capabilities Information (required, string ident = "")

- Byte 1 = RFU
- Byte 2 = Terminal Type
- - Bit 8 = VAS Support (1=on, 0 = off, string ident = "")
- - Bit 7 = Touch ID Required (1=on, 0 = off, string ident = "")
- - Bit 6 = RFU
- - Bit 5 = RFU
- - Bit 1,2,3,4
- - - 0 = Payment Terminal
- - - 1 = Transit Terminal
- - - 2 = Access Terminal
- - - 3 = Wireless Handoff Terminal
- - - 4 = App Handoff Terminal

- - 15 = Other Terminal
- Byte 3 = RFU
- Byte 4 = Terminal Mode
- - 0 = ApplePay VAS OR ApplePay
- - 1 = ApplePay VAS AND ApplePay
- - 2 = ApplePay VAS ONLY
- - 3 = ApplePay ONLY 9F2B = 5 bytes = ApplePay VAS Filter. Each byte filters for that specific merchant index (optional, string ident = "") DF01 = 1 byte = ApplePay VAS Protocol. (optional, string ident = "")
- - Bit 1 : 1 = URL VAS, 0 = Full VAS
- - Bit 2 : 1 = VAS Beeps, 0 = No VAS Beeps
- - Bit 3 : 1 = Silent Comm Error, 2 = EMEA Comm Error
- - Bit 4-8 : RFU

21.20.2.27 `RETURN_CODE IDTechSDK.IDT_VP3300.ctls_trySetTerminalData (byte[] tlv, ref byte[] rejectedTLV, ref byte[] convertedTLV, string ident = " ")`

Try to Set CTLS Terminal Data

Attempts to set the CTLS Terminal Data. Use this functions to attempt to set the terminal data to the device. This function will allow the cross-device tag compatible writing of IDTech Legacy/New tag definitions

Parameters

<i>tlv</i>	TerminalData TLV Data
<i>rejectedTLV</i>	Contains the tags (if any) that were rejected by the firmware
<i>convertedTLV</i>	Contains the tags (if any) that were converted and accepted by the firmware
<i>ident</i>	Optional identifier

Return values

<i>RETURN_CODE</i>	Return codes listed as typedef enum in IDTCommon:RETURN_CODE. Values can be parsed with IDT_BTPay::device_getResponseCodeString:()
--------------------	--

21.20.2.28 `RETURN_CODE IDTechSDK.IDT_VP3300.device_activateTransaction (int timeout, byte[] tags, bool isFastEMV = false, string ident = " ")`

Start a Transaction Request

Authorizes the transaction CTLS, MSR or Contact EMV

The tags will be returned in the callback routine.

Parameters

<i>timeout</i>	Timeout value in seconds.
<i>tags</i>	The tags to be included in the request. Passed as TLV Data. Example, tag 9F02 with amount 0x0000000000100 would be 0x9F0206000000000100

Parameters

<i>isFastEMV</i>	If TRUE, it will populate the IDTTransactionData.fastEMV with ASCII data similar to IDTech FastEMV KB output, after performing an auto-authenticate and auto-complete with ResultCode = Could Not Contact Host
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with `errorCode.getErrorString()`

NOTE ON APPLEPAY VAS: To enable ApplePay VAS, first a merchant record must be defined in one of the six available index positions (1-6) using `device_setMerchantRecord`, then container tag FFEE06 must be sent as part of the additional tags parameter of `ctls_startTransaction`. Tag FFEE06 must contain tag 9F26 and 9F22, and can optionally contain tags 9F2B and DFO1. Example FFEE06189F220201009F2604000000009F2B050100000000DF010101 9F22 = two bytes = ApplePay Terminal Application Version Number. Hard defined as 0100 for now. (required, string ident = "") 9F26 = four bytes = ApplePay Terminal Capabilities Information (required, string ident = "")

- Byte 1 = RFU
- Byte 2 = Terminal Type
- - Bit 8 = VAS Support (1=on, 0 = off, string ident = "")
- - Bit 7 = Touch ID Required (1=on, 0 = off, string ident = "")
- - Bit 6 = RFU
- - Bit 5 = RFU
- - Bit 1,2,3,4
- - - 0 = Payment Terminal
- - - 1 = Transit Terminal
- - - 2 = Access Terminal
- - - 3 = Wireless Handoff Terminal
- - - 4 = App Handoff Terminal
- - - 15 = Other Terminal
- Byte 3 = RFU
- Byte 4 = Terminal Mode
- - 0 = ApplePay VAS OR ApplePay
- - 1 = ApplePay VAS AND ApplePay
- - 2 = ApplePay VAS ONLY
- - 3 = ApplePay ONLY 9F2B = 5 bytes = ApplePay VAS Filter. Each byte filters for that specific merchant index (optional, string ident = "") DFO1 = 1 byte = ApplePay VAS Protocol. (optional, string ident = "")
- - Bit 1 : 1 = URL VAS, 0 = Full VAS
- - Bit 2 : 1 = VAS Beeps, 0 = No VAS Beeps
- - Bit 3 : 1 = Silent Comm Error, 2 = EMEA Comm Error
- - Bit 4-8 : RFU

21.20.2.29 RETURN_CODE IDTechSDK.IDT_VP3300.device_buzzer (string *ident* = " ")**Buzzer Device**

Buzzer the reader. If success can hear one beep. Otherwise, returns timeout.

Parameters

<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.
--------------	--

Returns

RETURN_CODE: Values can be parsed with `errorCode.getErrorString(string ident = "")`

21.20.2.30 RETURN_CODE IDTechSDK.IDT_VP3300.device_cancelTransaction (string *ident* = " ")**Cancel Transaction**

Cancels the currently executing device transaction.

Parameters

<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.
--------------	--

Returns

RETURN_CODE: Values can be parsed with `errorCode.getErrorString()`

21.20.2.31 RETURN_CODE IDTechSDK.IDT_VP3300.device_controlLED (byte *indexLED*, byte *control*, string *ident* = " ")**Control LED**

Controls the LED for the reader. This command will only operate in pass-through mode

Parameters

<i>indexLED</i>	For LED <ul style="list-style-type: none"> • 00: LED 0 (Power LED, leftmost LED) • 01: LED 1 • 02: LED 2 • 03: LED 3 • FF: All LED's
<i>control</i>	LED Status: <ul style="list-style-type: none"> • 00: LED Off • 01: LED On
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with `device_getResponseCodeString`

21.20.2.32 RETURN_CODE IDTechSDK.IDT_VP3300.device_controlUserInterface (byte[] *values*, string *ident* = " ")

Control User Interface

Controls the User Interface: Display, Beep, LED

Parameters

<i>values</i>	<p>Four bytes to control the user interface Byte[0] = LCD Message Messages 00-07 are normally controlled by the reader.</p> <ul style="list-style-type: none"> • 00h: Idle Message (Welcome, string ident = "") • 01h: Present card (Please Present Card, string ident = "") • 02h: Time Out or Transaction cancel (No Card, string ident = "") • 03h: Transaction between reader and card is in the middle (Processing...) • 04h: Transaction Pass (Thank You, string ident = "") • 05h: Transaction Fail (Fail, string ident = "") • 06h: Amount (Amount \$ 0.00 Tap Card, string ident = "") • 07h: Balance or Offline Available funds (Balance \$ 0.00) Messages 08-0B are controlled by the terminal • 08h: Insert or Swipe card (Use Chip & PIN, string ident = "") • 09h: Try Again(Tap Again, string ident = "") • 0Ah: Tells the customer to present only one card (Present 1 card only) • 0Bh: Tells the customer to wait for authentication/authorization (Wait) • FFh: indicates the command is setting the LED/Buzzer only. Byte[1] = Beep Indicator • 00h: No beep • 01h: Single beep • 02h: Double beep • 03h: Three short beeps • 04h: Four short beeps • 05h: One long beep of 200 ms • 06h: One long beep of 400 ms • 07h: One long beep of 600 ms • 08h: One long beep of 800 ms Byte[2] = LED Number • 00h: LED 0 (Power LED) 01h: LED 1 • 02h: LED 2 • 03h: LED 3 • FFh: All LEDs Byte[3] = LED Status • 00h: LED Off • 01h: LED On
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with `errorCode.getErrorString(string ident = "")`

21.20.2.33 **RETURN_CODE** IDTechSDK.IDT_VP3300.device_enablePassThrough (bool *enablePassThrough*, string *ident* = " ")

Enable Pass Through

Enables Pass Through Mode for direct communication with L1 interface (power on icc, send apdu, etc).

Parameters

<i>enablePassThrough</i>	true = pass through ON, false = pass through OFF
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with `errorCode.getErrorString(string ident = "")`

21.20.2.34 **RETURN_CODE** IDTechSDK.IDT_VP3300.device_getAnyRKIStatus (bool *isMultiKey*, ref string *status*, ref Dictionary< string, RKI_KEY_TYPE > *keys*, string *ident* = " ")

Get RKI Status Polls the RKI servers to validate device RKI eligibility. Note: if device type is known in advance (production or demo device), it is more efficient to use `device_getRKIStatus` instead

Parameters

<i>isMultiKey</i>	True = MultiKey, False = SingleKey
<i>status</i>	RKI status returned as a string
<i>keys</i>	Returns all available keys
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with `errorCode.getErrorString()`. When no data is available, return code = **RETURN_CODE_NO_DATA_AVAILABLE**

21.20.2.35 **RETURN_CODE** IDTechSDK.IDT_VP3300.device_getCashTranRiskPara (ref byte[] *tlv*, string *ident* = " ")

Get Cash Transaction Reader Risk Parameters Returns the TTQ and reader risk parameters that will be used for cash transactions, if enabled.

Parameters

<i>tlv</i>	TLV Data Objects
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with `errorCode.getErrorString(string ident = "")`

21.20.2.36 **RETURN_CODE** IDTechSDK.IDT_VP3300.device_getConfigurationFromMemory (ref string *json*, string *ident* = " ")

Read Device Configuration From Memory Retrieves the ViVOconfig configuration data from memory after a successful `device_readConfigurationToMemory`

Parameters

<i>json</i>	Returns the .json file data that can be saved to disk
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with `errorCode.getErrorString()`. When no data is available, return code = RETURN_CODE_NO_DATA_AVAILABLE

21.20.2.37 RETURN_CODE IDTechSDK.IDT_VP3300.device_getDateTime (ref byte[] *dateTime*, string *ident* = " ")

Get Date Time

Gets current system date and time of the device.

Parameters

<i>dateTime</i>	The date time returned as follows: <ul style="list-style-type: none"> • byte 0: Year 00-99 • byte 1: Month 01-12 • byte 2: Date 01-31 • byte 3: Hour 00-23 • byte 4: Minute 00-59 • byte 5: Second 00-59
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with `device_getResponseCodeString`

21.20.2.38 RETURN_CODE IDTechSDK.IDT_VP3300.device_getDrlReaderRiskPara (byte *index*, ref byte[] *tlv*, string *ident* = " ")

Get DRL Reader Risk Parameters Get the Index, Application Program ID, and reader risk parameters for the DRL settings.

Parameters

<i>index</i>	DRL index (01-04, string <i>ident</i> = "")
<i>tlv</i>	TLV Data Objects
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with `errorCode.getErrorString(string ident = "")`

21.20.2.39 **RETURN_CODE** IDTechSDK.IDT_VP3300.device_getFirmwareVersion (ref string *response*, string *ident* = " ")

Polls device for Firmware Version

Parameters

<i>response</i>	Response returned of Firmware Version
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with `errorCode.getErrorString(string ident = "")`

21.20.2.40 **RETURN_CODE** IDTechSDK.IDT_VP3300.device_getHardwareInfor (ref string *ascii*, string *ident* = " ")

Get Hardware Information

Parameters

<i>ascii</i>	the ascii charactor
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

RETURN_CODE: Values can be parsed with `errorCode.getErrorString(string ident = "")`

ASCII | Description

HW,VPVendi<CR><LF>K21F Rev9 | Vendi HW,VP3300 Audio Jack<CR><LF>K21F Rev9 | Unipay III HW,V←
PUnipay1.5<CR><LF>K21F Rev9 | Unipay 1.5 HW,VPUniPay1.5TTK<CR><LF>K21F Rev9 | UniPay 1.5 TTK
HW,VP3300 USB<CR><LF>K21F Rev9 | VP3300 USB, VP3300 USB OEM (iBase/Cake same code, string ident
= "") HW,VP3300 USB-E<CR><LF>K21F Rev9 | VP3300 USB-E HW,VP3300 USB-C<CR><LF>K21F Rev9
| VP3300 USB-C HW,VPVP3300 Bluetooth<CR><LF>K21F Rev9 | VP3300 Bluetooth HW,.VP6300<CR><L←
F>K81F.Rev4 | VP6300

21.20.2.41 **RETURN_CODE** IDTechSDK.IDT_VP3300.device_getMerchantRecord (int *index*, ref byte[] *record*, string *ident* = " ")

Get Merchant Record Gets the merchant record for ApplePay VAS

Parameters

<i>index</i>	Merchant Record index, valid values 1-6
<i>record</i>	Data returned containing 99 bytes: Byte 0 = Merchand Index Byte 1 = Merchant Enabled (1 = enabled, string ident = "") Byte 2 - 33 = Merchant Protocol Hash-256 value Byte 34 = Length of Merchant URL Bytes 35 - 99 = URL
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with `errorCode.getErrorString(string ident = "")`

21.20.2.42 RETURN_CODE IDTechSDK.IDT_VP3300.device_getModuleVer (ref string *moduleVer*, string *ident* = " ")

Get Module Version Information Get the 16 byte UID of MCU

Parameters

<i>uid</i>	string UID
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with `errorCode.getErrorString(string ident = "")`

21.20.2.43 RETURN_CODE IDTechSDK.IDT_VP3300.device_getPollMode (ref byte *mode*, string *ident* = " ")

Get Poll Mode

Gets the poll mode for the device.

Parameters

<i>mode</i>	: 00h = Auto Poll (USB I/F auto set to USB/HID, string ident = "") 01h = Poll on Demand (USB I/F auto set to USB/HID, string ident = "") 02h = QuickChip (USB I/F auto set to USB KB, string ident = "") 03h = QuickChip (enable QuickChip for CT + MSR, string ident = "") 04h = QuickChip (enable QuickChip for CL + MSR, string ident = "")
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with `errorCode.getErrorString(string ident = "")`

21.20.2.44 RETURN_CODE IDTechSDK.IDT_VP3300.device_getProductType (ref byte[] *type*, string *ident* = " ")

Get Product Type Returns a "product type" value in a proprietary TLV

Parameters

<i>type</i>	product type
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

RETURN_CODE: Values can be parsed with `errorCode.getErrorString(string ident = "")`

Product Type | Description

42 37 00 | ViVOpay 5000 43 33 00 | ViVOpay 4500 43 35 00 | ViVOpay Vend 43 36 00 | Vendi (NEO, string ident = "") 43 37 00 | ViVOpay Kiosk1 (ATM1, string ident = "") 43 38 00 | Kiosk2 43 39 00 | Kiosk3 (NEO, string ident = "") 55 31 00 | UniPay 1.5 (NEO, string ident = "") 55 33 00 | UniPay III (NEO) 55 33 31 | VP3300, VP3300 OEM (NEO) (iBase/Cake same code, string ident = "") 55 33 32 | VP3300E(NEO, string ident = "") 55 33 33 | VP3300C(NEO, string ident = "") 55 33 34 | BTPay Mini (NEO) (UniPayIII + BLE, string ident = "") 56 31 00 | VP3600 56 32 00 | VP5200 56 33 00 | VP5300 56 34 00 | VP6300 56 35 00 | VP6800 56 36 00 | VP8300 56 37 00 | VP8310 56 38 00 | VP8800 56 39 00 | VP8810 56 40 00 | VP9000 44 30 00 | QX120 44 31 00 | Mx8Series 44 32 00 | NETs 44 33 00

| Magtek 44 35 00 | ICP

21.20.2.45 string IDTechSDK.IDT_VP3300.device_getResponseCodeString (RETURN_CODE *eCode*)

Get the description of response result.

Parameters

<i>eCode</i>	the response result.
--------------	----------------------

Return values

<i>the</i>	string for description of response result
------------	---

- case 0: "no error, beginning task";
- case 1: "no response from reader";
- case 2: "invalid response data";
- case 3: "time out for task or CMD";
- case 4: "wrong parameter";
- case 5: "SDK is doing MSR or ICC task";
- case 6: "SDK is doing PINPad task";
- case 7: "SDK is doing CTLS task";
- case 8: "SDK is doing EMV task";
- case 9: "SDK is doing Other task";
- case 10: "err response or data";
- case 11: "no reader attached";
- case 12: "mono audio is enabled";
- case 13: "did connection";
- case 14: "audio volume is too low";
- case 15: "task or CMD be canceled";
- case 16: "UF wrong string format";
- case 17: "UF file not found";
- case 18: "UF wrong file format";
- case 19: "Attempt to contact online host failed";
- case 20: "Attempt to perform RKI failed";
- case 0x300: "Key Type(TDES) of Session Key is not same as the related Master Key.";
- case 0x400: "Related Key was not loaded.";
- case 0x500: "Key Same.";
- case 0x501: "Key is all zero";

- case 0x502: "TR-31 format error";
- case 0x702: "PAN is Error Key.";
- case 0x705: "No Internal MSR PAN (or Internal MSR PAN is erased timeout)";
- case 0X0C01: "Incorrect Frame Tag";
- case 0X0C02: "Incorrect Frame Type";
- case 0X0C03: "Unknown Frame Type";
- case 0X0C04: "Unknown Command";
- case 0X0C05: "Unknown Sub-Command";
- case 0X0C06: "CRC Error";
- case 0X0C07: "Failed";
- case 0X0C08: "Timeout";
- case 0X0C0A: "Incorrect Parameter";
- case 0X0C0B: "Command Not Supported";
- case 0X0C0C: "Sub-Command Not Supported";
- case 0X0C0D: "Parameter Not Supported / Status Abort Command";
- case 0X0C0F: "Sub-Command Not Allowed";
- case 0X0D01: "Incorrect Header Tag";
- case 0X0D02: "Unknown Command";
- case 0X0D03: "Unknown Sub-Command";
- case 0X0D04: "CRC Error in Frame";
- case 0X0D05: "Incorrect Parameter";
- case 0X0D06: "Parameter Not Supported";
- case 0X0D07: "Mal-formatted Data";
- case 0X0D08: "Timeout";
- case 0X0D0A: "Failed / NACK";
- case 0X0D0B: "Command not Allowed";
- case 0X0D0C: "Sub-Command not Allowed";
- case 0X0D0D: "Buffer Overflow (Data Length too large for reader buffer)";
- case 0X0D0E: "User Interface Event";
- case 0X0D11: "Communication type not supported, VT-1, burst, etc.";
- case 0X0D12: "Secure interface is not functional or is in an intermediate state.";
- case 0X0D13: "Data field is not mod 8";
- case 0X0D14: "Pad 0x80 not found where expected";
- case 0X0D15: "Specified key type is invalid";
- case 0X0D1: "Could not retrieve key from the SAM(InitSecureComm)";
- case 0X0D17: "Hash code problem";

- case 0X0D18: "Could not store the key into the SAM(InstallKey)";
- case 0X0D19: "Frame is too large";
- case 0X0D1A: "Unit powered up in authentication state but POS must resend the InitSecureComm command";
- case 0X0D1B: "The EEPROM may not be initialized because SecCommInterface does not make sense";
- case 0X0D1C: "Problem encoding APDU";
- case 0X0D20: "Unsupported Index(ILM) SAM Transceiver error – problem communicating with the SAM(Key Mgr)";
- case 0X0D2: "Unexpected Sequence Counter in multiple frames for single bitmap(ILM) Length error in data returned from the SAM(Key Mgr)";
- case 0X0D22: "Improper bit map(ILM)";
- case 0X0D23: "Request Online Authorization";
- case 0X0D24: "ViVOCard3 raw data read successful";
- case 0X0D25: "Message index not available(ILM) ViVOcomm activate transaction card type(ViVOcomm)";
- case 0X0D26: "Version Information Mismatch(ILM)";
- case 0X0D27: "Not sending commands in correct index message index(ILM)";
- case 0X0D28: "Time out or next expected message not received(ILM)";
- case 0X0D29: "ILM languages not available for viewing(ILM)";
- case 0X0D2A: "Other language not supported(ILM)";
- case 0X0D41: "Unknown Error from SAM";
- case 0X0D42: "Invalid data detected by SAM";
- case 0X0D43: "Incomplete data detected by SAM";
- case 0X0D44: "Reserved";
- case 0X0D45: "Invalid key hash algorithm";
- case 0X0D46: "Invalid key encryption algorithm";
- case 0X0D47: "Invalid modulus length";
- case 0X0D48: "Invalid exponent";
- case 0X0D49: "Key already exists";
- case 0X0D4A: "No space for new RID";
- case 0X0D4B: "Key not found";
- case 0X0D4C: "Crypto not responding";
- case 0X0D4D: "Crypto communication error";
- case 0X0D4E: "Module-specific error for Key Manager";
- case 0X0D4F: "All key slots are full (maximum number of keys has been installed)";
- case 0X0D50: "Auto-Switch OK";
- case 0X0D51: "Auto-Switch failed";
- case 0X0D90: "Account DUKPT Key not exist";

- case 0X0D91: "Account DUKPT Key KSN exhausted";
- case 0x0D00: "This Key had been loaded.";
- case 0x0E00: "Base Time was loaded.";
- case 0x0F00: "Encryption Or Decryption Failed.";
- case 0x1000: "Battery Low Warning (It is High Priority Response while Battery is Low.)";
- case 0x1800: "Send "Cancel Command" after send "Get Encrypted PIN" &"Get Numeric "& "Get Amount"";
- case 0x1900: "Press "Cancel" key after send "Get Encrypted PIN" &"Get Numeric "& "Get Amount"";
- case 0x30FF: "Security Chip is not connect";
- case 0x3000: "Security Chip is deactivation & Device is In Removal Legally State.";
- case 0x3101: "Security Chip is activation & Device is In Removal Legally State.";
- case 0x5500: "No Admin DUKPT Key.";
- case 0x5501: "Admin DUKPT Key STOP.";
- case 0x5502: "Admin DUKPT Key KSN is Error.";
- case 0x5503: "Get Authentication Code1 Failed.";
- case 0x5504: "Validate Authentication Code Error.";
- case 0x5505: "Encrypt or Decrypt data failed.";
- case 0x5506: "Not Support the New Key Type.";
- case 0x5507: "New Key Index is Error.";
- case 0x5508: "Step Error.";
- case 0x5509: "KSN Error";
- case 0x550A: "MAC Error.";
- case 0x550B: "Key Usage Error.";
- case 0x550C: "Mode Of Use Error.";
- case 0x550F: "Other Error.";
- case 0x6000: "Save or Config Failed / Or Read Config Error.";
- case 0x6200: "No Serial Number.";
- case 0x6900: "Invalid Command - Protocol is right, but task ID is invalid.";
- case 0x6A01: "Unsupported Command – Protocol and task ID are right, but command is invalid – In this State";
- case 0x6A00: "Unsupported Command - Protocol and task ID are right, but command is invalid.";
- case 0x6B00: "Unknown parameter in command - Protocol task ID and command are right, but parameter is invalid.";
- case 0x6C00: "Unknown parameter in command – Protocol task ID and command are right, but length is out of the requirement.";
- case 0x7200: "Device is suspend (MKSK suspend or press password suspend).";
- case 0x7300: "PIN DUKPT is STOP (21 bit 1).";
- case 0x7400: "Device is Busy.";

- case 0xE100: "Can not enter sleep mode";
- case 0xE200: "File has existed";
- case 0xE300: "File has not existed";
- case 0xE313: "IO line low -- Card error after session start";
- case 0xE400: "Open File Error";
- case 0xE500: "SmartCard Error";
- case 0xE600: "Get MSR Card data is error";
- case 0xE700: "Command time out";
- case 0xE800: "File read or write is error";
- case 0xE900: "Active 1850 error!";
- case 0xEA00: "Load bootloader error";
- case 0xEF00: "Protocol Error- STX or ETX or check error.";
- case 0xEB00: "Picture is not exist";
- case 0x2C02: "No Microprocessor ICC seated";
- case 0x2C06: "no card seated to request ATR";
- case 0x2D01: "Card Not Supported,";
- case 0x2D03: "Card Not Supported, wants CRC";
- case 0x690D: "Command not supported on reader without ICC support";
- case 0x8100: "ICC error time out on power-up";
- case 0x8200: "invalid TS character received - Wrong operation step";
- case 0x8300: "Decode MSR Error";
- case 0x8400: "TriMagII no Response";
- case 0x8500: "No Swipe MSR Card";
- case 0x8510: "No Financial Card";
- case 0x8600: "Unsupported F, D, or combination of F and D";
- case 0x8700: "protocol not supported EMV TD1 out of range";
- case 0x8800: "power not at proper level";
- case 0x8900: "ATR length too long";
- case 0x8B01: "EMV invalid TA1 byte value";
- case 0x8B02: "EMV TB1 required";
- case 0x8B03: "EMV Unsupported TB1 only 00 allowed";
- case 0x8B04: "EMV Card Error, invalid BWI or CWI";
- case 0x8B06: "EMV TB2 not allowed in ATR";
- case 0x8B07: "EMV TC2 out of range";
- case 0x8B08: "EMV TC2 out of range";
- case 0x8B09: "per EMV96 TA3 must be > 0xF";

- case 0x8B10: "ICC error on power-up";
- case 0x8B11: "EMV T=1 then TB3 required";
- case 0x8B12: "Card Error, invalid BWI or CWI";
- case 0x8B13: "Card Error, invalid BWI or CWI";
- case 0x8B17: "EMV TC1/TB3 conflict-";
- case 0x8B20: "EMV TD2 out of range must be T=1";
- case 0x8C00: "TCK error";
- case 0xA304: "connector has no voltage setting";
- case 0xA305: "ICC error on power-up invalid (SBLK(IFSD) exchange";
- case 0xE301: "ICC error after session start";
- case 0xFF00: "Request to go online";
- case 0xFF01: "EMV: Accept the offline transaction";
- case 0xFF02: "EMV: Decline the offline transaction";
- case 0xFF03: "EMV: Accept the online transaction";
- case 0xFF04: "EMV: Decline the online transaction";
- case 0xFF05: "EMV: Application may fallback to magstripe technology";
- case 0xFF06: "EMV: ICC detected tah the conditions of use are not satisfied";
- case 0xFF07: "EMV: ICC didn't accept transaction";
- case 0xFF08: "EMV: Transaction was cancelled";
- case 0xFF09: "EMV: Application was not selected by kernel or ICC format error or ICC missing data error";
- case 0xFF0A: "EMV: Transaction is terminated";
- case 0xFF0B: "EMV: Other EMV Error";
- case 0xFFFF: "NO RESPONSE";
- case 0xF002: "ICC communication timeout";
- case 0xF003: "ICC communication Error";
- case 0xF00F: "ICC Card Seated and Highest Priority, disable MSR work request";
- case 0xF200: "AID List / Application Data is not exist";
- case 0xF201: "Terminal Data is not exist";
- case 0xF202: "TLV format is error";
- case 0xF203: "AID List is full";
- case 0xF204: "Any CA Key is not exist";
- case 0xF205: "CA Key RID is not exist";
- case 0xF206: "CA Key Index it not exist";
- case 0xF207: "CA Key is full";
- case 0xF208: "CA Key Hash Value is Error";
- case 0xF209: "Transaction format error";

- case 0xF20A: "The command will not be processing";
- case 0xF20B: "CRL is not exist";
- case 0xF20C: "CRL number exceed max number";
- case 0xF20D: "Amount,Other Amount,Trasaction Type are missing";
- case 0xF20E: "The Identification of algorithm is mistake";
- case 0xF20F: "No Financial Card";
- case 0xF210: "In Encrypt Result state, TLV total Length is greater than Max Length";
- case 0x1001: "INVALID ARG";
- case 0x1002: "FILE_OPEN_FAILED";
- case 0x1003: "FILE OPERATION_FAILED";
- case 0x2001: "MEMORY_NOT_ENOUGH";
- case 0x3002: "SMARTCARD_FAIL";
- case 0x3003: "SMARTCARD_INIT_FAILED";
- case 0x3004: "FALLBACK_SITUATION";
- case 0x3005: "SMARTCARD_ABSENT";
- case 0x3006: "SMARTCARD_TIMEOUT";
- case 0x5001: "EMV_PARSING_TAGS_FAILED";
- case 0x5002: "EMV_DUPLICATE_CARD_DATA_ELEMENT";
- case 0x5003: "EMV_DATA_FORMAT_INCORRECT";
- case 0x5004: "EMV_NO_TERM_APP";
- case 0x5005: "EMV_NO_MATCHING_APP";
- case 0x5006: "EMV_MISSING_MANDATORY_OBJECT";
- case 0x5007: "EMV_APP_SELECTION_RETRY";
- case 0x5008: "EMV_GET_AMOUNT_ERROR";
- case 0x5009: "EMV_CARD_REJECTED";
- case 0x5010: "EMV_AIP_NOT_RECEIVED";
- case 0x5011: "EMV_AFL_NOT_RECEIVED";
- case 0x5012: "EMV_AFL_LEN_OUT_OF_RANGE";
- case 0x5013: "EMV_SFI_OUT_OF_RANGE";
- case 0x5014: "EMV_AFL_INCORRECT";
- case 0x5015: "EMV_EXP_DATE_INCORRECT";
- case 0x5016: "EMV_EFF_DATE_INCORRECT";
- case 0x5017: "EMV_ISS_COD_TBL_OUT_OF_RANGE";
- case 0x5018: "EMV_CRYPTOGAM_TYPE_INCORRECT";
- case 0x5019: "EMV_PSE_NOT_SUPPORTED_BY_CARD";
- case 0x5020: "EMV_USER_SELECTED_LANGUAGE";

- case 0x5021: "EMV_SERVICE_NOT_ALLOWED";
- case 0x5022: "EMV_NO_TAG_FOUND";
- case 0x5023: "EMV_CARD_BLOCKED";
- case 0x5024: "EMV_LEN_INCORRECT";
- case 0x5025: "CARD_COM_ERROR";
- case 0x5026: "EMV_TSC_NOT_INCREASED";
- case 0x5027: "EMV_HASH_INCORRECT";
- case 0x5028: "EMV_NO_ARC";
- case 0x5029: "EMV_INVALID_ARC";
- case 0x5030: "EMV_NO_ONLINE_COMM";
- case 0x5031: "TRAN_TYPE_INCORRECT";
- case 0x5032: "EMV_APP_NO_SUPPORT";
- case 0x5033: "EMV_APP_NOT_SELECT";
- case 0x5034: "EMV_LANG_NOT_SELECT";
- case 0x5035: "EMV_NO_TERM_DATA";
- case 0x6001: "CVM_TYPE_UNKNOWN";
- case 0x6002: "CVM_AIP_NOT_SUPPORTED";
- case 0x6003: "CVM_TAG_8E_MISSING";
- case 0x6004: "CVM_TAG_8E_FORMAT_ERROR";
- case 0x6005: "CVM_CODE_IS_NOT_SUPPORTED";
- case 0x6006: "CVM_COND_CODE_IS_NOT_SUPPORTED";
- case 0x6007: "NO_MORE_CVM";
- case 0x6008: "PIN_BYPASSED_BEFORE";
- case 0x7001: "PK_BUFFER_SIZE_TOO_BIG";
- case 0x7002: "PK_FILE_WRITE_ERROR";
- case 0x7003: "PK_HASH_ERROR";
- case 0x8001: "NO_CARD_HOLDER_CONFIRMATION";
- case 0x8002: "GET_ONLINE_PIN";
- case 0xD000: "Data not exist";
- case 0xD001: "Data access error";
- case 0xD100: "RID not exist";
- case 0xD101: "RID existed";
- case 0xD102: "Index not exist";
- case 0xD200: "Maximum exceeded";
- case 0xD201: "Hash error";
- case 0xD205: "System Busy";

- case 0x0E01: "Unable to go online";
- case 0x0E02: "Technical Issue";
- case 0x0E03: "Declined";
- case 0x0E04: "Issuer Referral transaction";
- case 0x0F01: "Decline the online transaction";
- case 0x0F02: "Request to go online";
- case 0x0F03: "Transaction is terminated";
- case 0x0F05: "Application was not selected by kernel or ICC format error or ICC missing data error";
- case 0x0F07: "ICC didn't accept transaction";
- case 0x0F0A: "Application may fallback to magstripe technology";
- case 0x0F0C: "Transaction was cancelled";
- case 0x0F0D: "Timeout";
- case 0x0F0F: "Other EMV Error";
- case 0x0F10: "Accept the offline transaction";
- case 0x0F11: "Decline the offline transaction";
- case 0x0F21: "ICC detected tah the conditions of use are not satisfied";
- case 0x0F22: "No app were found on card matching terminal configuration";
- case 0x0F23: "Terminal file does not exist";
- case 0x0F24: "CAPK file does not exist";
- case 0x0F25: "CRL Entry does not exist";
- case 0x0FFE: "code when blocking is disabled";
- case 0x0FFF: "code when command is not applicable on the selected device";
- case 0xF005: "ICC Encrypted C-APDU Data Structure Length Error Or Format Error.";
- case 0xBBE0: "CM100 Success";
- case 0xBBE1: "CM100 Parameter Error";
- case 0xBBE2: "CM100 Low Output Buffer";
- case 0xBBE3: "CM100 Card Not Found";
- case 0xBBE4: "CM100 Collision Card Exists";
- case 0xBBE5: "CM100 Too Many Cards Exist";
- case 0xBBE6: "CM100 Saved Data Does Not Exist";
- case 0xBBE8: "CM100 No Data Available";
- case 0xBBE9: "CM100 Invalid CID Returned";
- case 0xBBEA: "CM100 Invalid Card Exists";
- case 0xBBEC: "CM100 Command Unsupported";
- case 0xBBED: "CM100 Error In Command Process";
- case 0xBBEE: "CM100 Invalid Command";

- case 0X9031: "Unknown command";
- case 0X9032: "Wrong parameter (such as the length of the command is incorrect)";
- case 0X9038: "Wait (the command couldn't be finished in BWT)";
- case 0X9039: "Busy (a previously command has not been finished)";
- case 0X903A: "Number of retries over limit";
- case 0X9040: "Invalid Manufacturing system data";
- case 0X9041: "Not authenticated";
- case 0X9042: "Invalid Master DUKPT Key";
- case 0X9043: "Invalid MAC Key";
- case 0X9044: "Reserved for future use";
- case 0X9045: "Reserved for future use";
- case 0X9046: "Invalid DATA DUKPT Key";
- case 0X9047: "Invalid PIN Pairing DUKPT Key";
- case 0X9048: "Invalid DATA Pairing DUKPT Key";
- case 0X9049: "No nonce generated";
- case 0X9949: "No GUID available. Perform getVersion first.";
- case 0X9950: "MAC Calculation unsuccessful. Check BDK value.";
- case 0X904A: "Not ready";
- case 0X904B: "Not MAC data";
- case 0X9050: "Invalid Certificate";
- case 0X9051: "Duplicate key detected";
- case 0X9052: "AT checks failed";
- case 0X9053: "TR34 checks failed";
- case 0X9054: "TR31 checks failed";
- case 0X9055: "MAC checks failed";
- case 0X9056: "Firmware download failed";
- case 0X9060: "Log is full";
- case 0X9061: "Removal sensor unengaged";
- case 0X9062: "Any hardware problems";
- case 0X9070: "ICC communication timeout";
- case 0X9071: "ICC data error (such check sum error)";
- case 0X9072: "Smart Card not powered up";

21.20.2.46 `RETURN_CODE IDTechSDK.IDT_VP3300.device_getRKIStatus (bool isProd, bool isMultiKey, ref string status, ref Dictionary< string, RKI_KEY_TYPE > keys, string ident = " ")`

Get RKI Status From Specified Server Polls the RKI server to validate device RKI eligibility

Parameters

<i>isProd</i>	TRUE = poll production RKI server, FALSE = poll demo RKI server
<i>isMultiKey</i>	True = MultiKey, False = SingleKey
<i>status</i>	RKI status returned as a string
<i>keys</i>	Returns dictionary of available keys
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with `errorCode.getErrorString()`. When no data is available, return code = RETURN_CODE_NO_DATA_AVAILABLE

21.20.2.47 RETURN_CODE IDTechSDK.IDT_VP3300.device_getTransactionResults (ref IDTTransactionData *results*, string *ident* = " ")

Get Transaction Results Gets the transaction results when the reader is functioning in "Auto Poll" mode

Parameters

<i>results</i>	The transaction results
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with `errorCode.getErrorString()`. When no data is available, return code = RETURN_CODE_NO_DATA_AVAILABLE

21.20.2.48 RETURN_CODE IDTechSDK.IDT_VP3300.device_getUIDofMCU (ref string *uid*, string *ident* = " ")

Get UID of MCU

Parameters

<i>moduleVer</i>	module version information
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with `errorCode.getErrorString(string ident = "")`

21.20.2.49 RETURN_CODE IDTechSDK.IDT_VP3300.device_getUsbBootLoader (ref string *bootLoader*, string *ident* = " ")

Get USB Boot Loader Version Get the version of the USB Boot Loader

Parameters

<i>bootLoader</i>	USB boot loader information
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with `errorCode.getErrorString(string ident = "")`

21.20.2.50 RETURN_CODE IDTechSDK.IDT_VP3300.device_pingDevice (string ident = " ")**Ping Device**

Pings the reader. If connected, returns success. Otherwise, returns timeout.

Parameters

<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.
--------------	--

Returns

RETURN_CODE: Values can be parsed with `errorCode.getErrorString(string ident = "")`

21.20.2.51 RETURN_CODE IDTechSDK.IDT_VP3300.device_readConfigurationToMemory (string memo, Configs.InstallRules rules, List< Configs.DeviceCommand > cmds, string ident = " ", bool isForeground = false)

Read Device Configuration To Memory Executes a ViVOconfig read of device settings and stores the results in memory

Once a ViVOconfig read starts, all messages will be returned to the callback as `DeviceState.ViVOconfig`. The ViVOconfig operation is complete when the return code from a `DeviceState.ViVOconfig` message is one of the following values:

- RETURN_CODE_SDK_VIVOCONFIG_READ_CONFIG_SUCCESS = Read process completed successfully. Execute `device_getConfigurationFromMemory` to retrieve the configuration data
- RETURN_CODE_SDK_VIVOCONFIG_READ_CONFIG_FAILED = Read process FAILED.

Parameters

<i>memo</i>	Optional: populates the memo field of the captured .json file
<i>rules</i>	Optional: populates the .json file with available install rules
<i>cmds</i>	Optional: populates the .json file with commands to execute
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.
<i>isForeground</i>	If TRUE, will block program until update complete. Otherwise, FALSE performs update on background.

Returns

RETURN_CODE: Values can be parsed with `errorCode.getErrorString()`. When no data is available, return code = RETURN_CODE_NO_DATA_AVAILABLE

21.20.2.52 RETURN_CODE IDTechSDK.IDT_VP3300.device_rebootDevice (string ident = " ")**Reboot Device**

Performs a reboot of the device

Parameters

<i>ip</i>	Optional: The callback function will only be applicable to the provided IP.
-----------	---

21.20.2.53 `RETURN_CODE IDTechSDK.IDT_VP3300.device_RemoteKeyInjection (RKI_KEY_TYPE type, string keyName, string ident = " ", bool performOnForeground = false)`

Remote Key Injection Performs a remote key injection for the device

Parameters

<i>type</i>	Remote Key Injection Type
<i>keyName</i>	Name of key (optional)
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.
<i>performOnForeground</i>	TRUE = block during process (default is false)

Returns

RETURN_CODE: Values can be parsed with `errorCode.getErrorString()`. When no data is available, return code = `RETURN_CODE_NO_DATA_AVAILABLE`

21.20.2.54 `RETURN_CODE IDTechSDK.IDT_VP3300.device_sendConfiguration (string filename, VC_OPERATION_TYPE type, bool matchFW, string ident = " ", bool isForeground = false)`

Send Configuration Executes a ViVOconfig update or verify to a device

Once a ViVOconfig write or verify starts, all messages will be returned to the callback as `DeviceState.ViVOconfig`. The ViVOconfig operation is complete when the return code from a `DeviceState.ViVOconfig` message is one of the following values:

- `RETURN_CODE_SDK_VIVOCONFIG_VERIFY_SUCCESS` = Verification process completed successfully. No differences found.
- `RETURN_CODE_SDK_VIVOCONFIG_VERIFY_SUCCESS_WITH_WARNING` = Verification process completed with warnings.
- `RETURN_CODE_SDK_VIVOCONFIG_VERIFY_FAILED` = Verification process FAILED
- `RETURN_CODE_SDK_VIVOCONFIG_WRITE_SUCCESS` = Write process completed successfully.
- `RETURN_CODE_SDK_VIVOCONFIG_WRITE_SUCCESS_WITH_WARNING` = Write process completed with warnings
- `RETURN_CODE_SDK_VIVOCONFIG_WRITE_FAILED` = Write process FAILED

Parameters

<i>filename</i>	The .json configuration file to update the device with NOTE: You can also use this parameter to pass the path of where the .json file is located at on local storage instead
<i>filename</i>	The name of the .json configuration file
<i>type</i>	<code>VC_OPERATION_TYPE</code>

- `VC_OPERATION_TYPE_WRITE` = Write File To Device, Hash must validate
- `VC_OPERATION_TYPE_VERIFY` = Verify Device With File

- VC_OPERATION_TYPE_WRITE_IGNORE_HASH = Write File To Device, Ignore Hash
- VC_OPERATION_TYPE_WRITE_FIX_HASH = Write File To Device, Fix Hash If Necessary

Parameters

<i>matchFW</i>	TRUE = Device FW must match file FW, FALSE = Don't validate device FW against file
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.
<i>isForeground</i>	If TRUE, will block program until update complete. Otherwise, FALSE performs update on background.

Returns

RETURN_CODE: Values can be parsed with `errorCode.getErrorString()`. When no data is available, return code = RETURN_CODE_NO_DATA_AVAILABLE

21.20.2.55 RETURN_CODE IDTechSDK.IDT_VP3300.device_sendConfigurationFromZip (byte[] zip, string filename, VC_OPERATION_TYPE type, bool matchFW, string ident = " ", bool isForeground = false)

Send Configuration From Zip Executes a ViVOconfig update or verify to a device using files from a .zip archive

Once a ViVOconfig write or update starts, all messages will be returned to the callback as DeviceState.ViVOconfig. The ViVOconfig operation is complete when the return code from a DeviceState.ViVOconfig message is one of the following values:

- RETURN_CODE_SDK_VIVOCONFIG_VERIFY_SUCCESS = Verification process completed successfully. No differences found.
- RETURN_CODE_SDK_VIVOCONFIG_VERIFY_SUCCESS_WITH_WARNING = Verification process completed with warnings.
- RETURN_CODE_SDK_VIVOCONFIG_VERIFY_FAILED = Verification process FAILED
- RETURN_CODE_SDK_VIVOCONFIG_WRITE_SUCCESS = Write process completed successfully.
- RETURN_CODE_SDK_VIVOCONFIG_WRITE_SUCCESS_WITH_WARNING = Write process completed with warnings
- RETURN_CODE_SDK_VIVOCONFIG_WRITE_FAILED = Write process FAILED

Parameters

<i>zip</i>	.Zip archive containing the .json configuration file and also any associated image assets NOTE: You can also use this parameter to pass the path of where the .zip file is located at on local storage instead
<i>filename</i>	The name of the .json configuration file
<i>type</i>	VC_OPERATION_TYPE

- VC_OPERATION_TYPE_WRITE = Write File To Device, Hash must validate
- VC_OPERATION_TYPE_VERIFY = Verify Device With File
- VC_OPERATION_TYPE_WRITE_IGNORE_HASH = Write File To Device, Ignore Hash
- VC_OPERATION_TYPE_WRITE_FIX_HASH = Write File To Device, Fix Hash If Necessary

Parameters

<i>matchFW</i>	TRUE = Device FW must match file FW, FALSE = Don't validate device FW against file
----------------	--

Parameters

<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.
<i>isForeground</i>	If TRUE, will block program until update complete. Otherwise, FALSE performs update on background.

Returns

RETURN_CODE: Values can be parsed with `errorCode.getErrorString()`. When no data is available, return code = RETURN_CODE_NO_DATA_AVAILABLE

21.20.2.56 RETURN_CODE IDTechSDK.IDT_VP3300.device_sendDataCommand (string *cmd*, bool *calcLRC*, ref byte[] *response*, string *ident* = " ")

Send a data command to the device

Sends a command to the device.

Parameters

<i>cmd</i>	String representation of command to execute
<i>calcLRC</i>	If TRUE, this will wrap command with start/length/lrc/sum/end: '{STX}{Len_Low}{Len_High} data {CheckLRC} {CheckSUM} {ETX}'
<i>response</i>	Response data
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with `device_getResponseCodeString`

21.20.2.57 RETURN_CODE IDTechSDK.IDT_VP3300.device_sendDataCommand_ext (string *cmd*, bool *calcLRC*, ref byte[] *response*, int *timeout*, bool *noResponse*, string *ident* = " ")

Send a data command to the device - extended

Sends a command to the device.

Parameters

<i>cmd</i>	String representation of command to execute
<i>calcLRC</i>	If TRUE, this will wrap command as NGA with start/length/lrc/sum/end: '{STX}{Len_Low}{Len_High} data {CheckLRC} {CheckSUM} {ETX}'
<i>response</i>	Response data
<i>timeout</i>	Timeout value waiting for response, in milliseconds (1000 = 1 second, string <i>ident</i> = "")
<i>noResponse</i>	if TRUE, this will not wait for a response and immediately return SUCCESS
<i>calcITP</i>	If TRUE, this will wrap command as ITP with start/end/lrc: '{STX} data {ETX}{CheckLRC}'
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with `device_getResponseCodeString`

21.20.2.58 `RETURN_CODE IDTechSDK.IDT_VP3300.device_sendPAE (string command, ref string response, int timeout, string ident = " ")`

Send Payment Application Engine Command

Executes a PAE command

Parameters

<i>command</i>	ASCII command string, should start with "*PAE"
<i>response</i>	command response
<i>timeout</i>	timeout waiting for PAE response
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with `errorCode.getErrorString(string ident = "")`

21.20.2.59 `RETURN_CODE IDTechSDK.IDT_VP3300.device_sendVivoCommandP2 (byte command, byte subCommand, byte[] data, ref byte[] response, string ident = " ")`

Send Vivo Command Protocol 2

Sends a protocol 2 command to Vivo readers (IDG/NEO, string ident = "")

Parameters

<i>command</i>	Command
<i>subCommand</i>	Sub-Command
<i>data</i>	Data. May be null
<i>response</i>	Response
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with `errorCode.getErrorString(string ident = "")`

21.20.2.60 `RETURN_CODE IDTechSDK.IDT_VP3300.device_sendVivoCommandP2_ext (byte command, byte subCommand, byte[] data, ref byte[] response, int timeout, bool noResponse, string ident = " ")`

Send Vivo Command Protocol 2 Extended

Sends a protocol 2 command to Vivo readers (IDG/NEO, string ident = "")

Parameters

<i>command</i>	Command
<i>subCommand</i>	Sub-Command
<i>data</i>	Data. May be null
<i>response</i>	Response
<i>timeout</i>	Timeout, in milliseconds (3000 = 3 seconds, string ident = "")
<i>noResponse</i>	TRUE = don't wait for response, FALSE = wait for response defined by timeout
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with `errorCode.getErrorString(string ident = "")`

21.20.2.61 RETURN_CODE IDTechSDK.IDT_VP3300.device_setBurstMode (byte *mode*, string *ident* = " ")

Send Burst Mode

Sets the burst mode for the device.

Parameters

<i>mode</i>	0 = OFF, 1 = Always On, 2 = Auto Exit
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with `errorCode.getErrorString(string ident = "")`

21.20.2.62 RETURN_CODE IDTechSDK.IDT_VP3300.device_setDateTime (string *ident* = " ")

Set Date Time

Set current system date and time to the device.

Parameters

<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.
--------------	--

Returns

RETURN_CODE: Values can be parsed with `device_getResponseCodeString`

21.20.2.63 RETURN_CODE IDTechSDK.IDT_VP3300.device_setLED (byte *indexLED*, byte *control*, string *ident* = " ")

Set LED

Control the reader. If connected, returns success. Otherwise, returns timeout. *indexLED* description as follows: 00h: LED 0 (Power LED, string *ident* = "") 01h: LED 1 02h: LED 2 03h: LED 3 10h: Single Tri-Color LED (Unipay III used, string *ident* = "") FFh: All 4 LEDs + Single Tri-Color LED Where the LEDs are numbered 0, 1, 2, 3 counting from the left. Note: If you are using pass-through mode to control the Power LED (LED 0), it is your responsibility to make sure that it behaves correctly. *control* description as follows: 00h: LED Off (LED 0~4 + Tri-Color LED, string *ident* = "") 01h: LED On (LED 0~4, string *ident* = "") 02h: Green Color (Tri-Color LED, string *ident* = "") 03h: Red Color (Tri-Color LED, string *ident* = "") 04h: Amber Color (Tri-Color LED, string *ident* = "")

Parameters

<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.
--------------	--

Returns

RETURN_CODE: Values can be parsed with `errorCode.getErrorString(string ident = "")`

21.20.2.64 **RETURN_CODE** IDTechSDK.IDT_VP3300.device_setMerchantRecord (int *index*, bool *enabled*, string *merchantID*, string *merchantURL*, string *ident* = " ")

Set Merchant Record Sets the merchant record for ApplePay VAS

Parameters

<i>index</i>	Merchant Record index, valid values 1-6
<i>enabled</i>	Merchant Enabled/Valid flag
<i>merchantID</i>	Merchant unique identifier registered with Apple. Example com.idtechproducts.applePay
<i>merchantURL</i>	Merchant URL, when applicable
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with `errorCode.getErrorString(string ident = " ")`

21.20.2.65 **RETURN_CODE** IDTechSDK.IDT_VP3300.device_setPollMode (byte *mode*, string *ident* = " ")

Set Poll Mode

Sets the poll mode for the device. Auto Poll keeps reader active, Poll On Demand only polls when requested by terminal

Parameters

<i>mode</i>	: 00h = Auto Poll (USB I/F auto set to USB/HID, string ident = "") 01h = Poll on Demand (USB I/F auto set to USB/HID, string ident = "") 02h = QuickChip (USB I/F auto set to USB KB, string ident = "") 03h = QuickChip (enable QuickChip for CT + MSR, string ident = "") 04h = QuickChip (enable QuickChip for CL + MSR, string ident = "")
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with `errorCode.getErrorString(string ident = " ")`

21.20.2.66 **RETURN_CODE** IDTechSDK.IDT_VP3300.device_setQuickChipHIDMode (byte *mode*, string *ident* = " ")

Set QuickChip HID Mode

Puts the device into QuickChip KB Output Mode while in HID

Parameters

<i>mode</i>	<ul style="list-style-type: none"> • 0 = Disable • 2 = CT + MSR + CL • 3 = CT + MSR • 4 = CL + MSR
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString

21.20.2.67 RETURN_CODE IDTechSDK.IDT_VP3300.device_startRKI (string ident = " ")**Start Remote Key Injection**

Starts a remote key injection request with IDTech RKI servers.

Parameters

<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.
--------------	--

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString

21.20.2.68 RETURN_CODE IDTechSDK.IDT_VP3300.device_StartRKI (int type, string ident = " ")**Start Remote Key Injection**

Starts a remote key injection request with IDTech RKI servers. Set/Get RKI url with IDT_Device.RKI_URL.

Parameters

<i>type</i>	0 = Symmetric RKI Demo Unit 1 = Symmetric RKI Production Unit 2 = PKI-RKI Demo Unit 3 = PKI-RKI Production Unit
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString

21.20.2.69 RETURN_CODE IDTechSDK.IDT_VP3300.device_startTransaction (double amount, double amtOther, int exponent, int type, int timeout, byte[] tags, bool isFastEMV = false, string ident = " ")**Start a Transaction Request**

Authorizes the transaction CTLS, MSR or Contact EMV

The tags will be returned in the callback routine.

Parameters

<i>amount</i>	Transaction amount value (tag value 9F02, string ident = "")
<i>amtOther</i>	Other amount value, if any (tag value 9F03, string ident = "")
<i>exponent</i>	Number of characters after decimile point
<i>type</i>	Transaction type (tag value 9C).
<i>timeout</i>	Timeout value in seconds.
<i>tags</i>	Any other tags to be included in the request. Passed as TLV Data. Example, tag 9F02 with amount 0x000000000100 would be 0x9F0206000000000100 If tags 9F02 (amount), 9F03 (other amount), or 9C (transaction type) are included, they will take priority over these values supplied as individual parameters to this method.

Parameters

<i>isFastEMV</i>	If TRUE, it will populate the IDTTransactionData.fastEMV with ASCII data similar to IDTech FastEMV KB output, after performing an auto-authenticate and auto-complete with ResultCode = Could Not Contact Host
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with `errorCode.getErrorString()`

NOTE ON APPLEPAY VAS: To enable ApplePay VAS, first a merchant record must be defined in one of the six available index positions (1-6) using `device_setMerchantRecord`, then container tag FFEE06 must be sent as part of the additional tags parameter of `ctls_startTransaction`. Tag FFEE06 must contain tag 9F26 and 9F22, and can optionally contain tags 9F2B and DFO1. Example FFEE06189F220201009F2604000000009F2B050100000000↵DF010101 9F22 = two bytes = ApplePay Terminal Application Version Number. Hard defined as 0100 for now. (required, string ident = "") 9F26 = four bytes = ApplePay Terminal Capabilities Information (required, string ident = "")

- Byte 1 = RFU
- Byte 2 = Terminal Type
 - - Bit 8 = VAS Support (1=on, 0 = off, string ident = "")
 - - Bit 7 = Touch ID Required (1=on, 0 = off, string ident = "")
 - - Bit 6 = RFU
 - - Bit 5 = RFU
 - - Bit 1,2,3,4
 - - - 0 = Payment Terminal
 - - - 1 = Transit Terminal
 - - - 2 = Access Terminal
 - - - 3 = Wireless Handoff Terminal
 - - - 4 = App Handoff Terminal
 - - - 15 = Other Terminal
- Byte 3 = RFU
- Byte 4 = Terminal Mode
 - - 0 = ApplePay VAS OR ApplePay
 - - 1 = ApplePay VAS AND ApplePay
 - - 2 = ApplePay VAS ONLY
 - - 3 = ApplePay ONLY 9F2B = 5 bytes = ApplePay VAS Filter. Each byte filters for that specific merchant index (optional, string ident = "") DFO1 = 1 byte = ApplePay VAS Protocol. (optional, string ident = "")
 - - Bit 1 : 1 = URL VAS, 0 = Full VAS
 - - Bit 2 : 1 = VAS Beeps, 0 = No VAS Beeps
 - - Bit 3 : 1 = Silent Comm Error, 2 = EMEA Comm Error
 - - Bit 4-8 : RFU

21.20.2.70 `static RETURN_CODE IDTechSDK.IDT_VP3300.device_updateDeviceFirmware (byte[] firmwareData, string ident = "") [static]`

Update Firmware

Updates the firmware .

Parameters

<i>firmwareData</i>	Signed binary data of a firmware file provided by IDTech
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with `errorCode.getErrorString(string ident = "")`

After you pass the `firmwareData` file, a new thread will start to execute the firmware download. You will receive status of the progress through callbacks to the `IDTechSDK.Callback()` delegate. The following parameters will be passed back:

- `sender = IDT_DEVICE_Types.IDT_DEVICE_AUGUSTA`
- `state = DeviceState.FirmwareUpdate`
- `transactionResultCode =` status of the firmware update (starting, entering bootloader, applying update, block success, firmware success, string `ident = ""`)
- `data =` File Progress. Four bytes, with bytes `[0][1]` = current block, and bytes `[2][3]` = total blocks. `0x00030010` = block 3 of 16

Example code starting a firmware update

```
OpenFileDialog diag = new OpenFileDialog( ident);
diag.Filter = "NGA FW Files|*.fm";

if (diag.ShowDialog() == DialogResult.OK, string ident = "")
{
    byte[] file = File.ReadAllBytes(diag.FileName, ident);
    RETURN_CODE rt = IDT_Device.SharedController.device_updateDeviceFirmware(file, ident);
    if (rt == RETURN_CODE.RETURN_CODE_DO_SUCCESS, string ident = "")
    {
        //Was a success
    }
    else
    {
        //Error starting firmware download
    }
}
```

Example monitoring firmware update status / success

```
private void MessageCallBack(IDTechSDK.IDT_DEVICE_Types type, DeviceState state, byte[] data,
    IDTTransactionData cardData, EMV_Callback emvCallback, RETURN_CODE transactionResultCode, string ident = "")
{
    switch (state, string ident = "")
    {
        case DeviceState.FirmwareUpdate:
            switch (transactionResultCode, string ident = "")
            {
                case RETURN_CODE.RETURN_CODE_FW_STARTING_UPDATE:
                    SetOutputText("Starting Firmware Update\n", ident);
                    break;
                case RETURN_CODE.RETURN_CODE_DO_SUCCESS:
                    SetOutputText("Firmware Update Successful\n", ident);
                    break;
                case RETURN_CODE.RETURN_CODE_APPLYING_FIRMWARE_UPDATE:
                    break;
            }
        break;
    }
}
```

```

        SetOutputText("Applying Firmware Update...\n", ident);
        break;
    case RETURN_CODE.RETURN_CODE_ENTERING_BOOTLOADER_MODE:
        SetOutputText("Entering Bootloader Mode...\n", ident);
        break;
    case RETURN_CODE.RETURN_CODE_BLOCK_TRANSFER_SUCCESS:

        int start = data[0] * 0x100 + data[1];
        int end = data[2] * 0x100 + data[3];

        SetOutputText("Sent block " + start.ToString() + " of " + end.ToString() + "\n", ident)
;
        break;
    default:
        SetOutputText("Firmware Update Error Code: " + "0x" + String.Format("{0:X}", (ushort)
transactionResultCode) + ": " + IDTechSDK.errorCode.getErrorString(transactionResultCode) + "\r\n",
ident);
        break;
    }
    break;
}
}
}

```

21.20.2.71 RETURN_CODE IDTechSDK.IDT_VP3300.device_updateFirmwareFromZip (byte[] zipfile, string ident = "", bool isForeground = false)

Update Firmware From Zip

Performs one or more device firmware updates from firmware files passed as a compressed archive with a valid configuration file (.json format)

Parameters

<i>zipfile</i>	Zip file containing on or more firmware update files and a .json configuration file
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.
<i>isForeground</i>	If TRUE, will block program until firmware update complete. Otherwise, FALSE performs FW update on background.

Returns

RETURN_CODE: Values can be parsed with `errorCode.getErrorString(string ident = "")`

21.20.2.72 RETURN_CODE IDTechSDK.IDT_VP3300.emv_activateTransaction (int timeout, byte[] tags, bool forceOnline, bool isFastEMV = false, string ident = "")

Start EMV Transaction Request

Authorizes the EMV transaction for an ICC card

The tags will be returned in the callback routine.

Parameters

<i>timeout</i>	Timeout value in seconds.
<i>tags</i>	The tags to be included in the request. Passed as TLV Data. Example, tag 9F02 with amount 0x000000000100 would be 0x9F0206000000000100
<i>forceOnline</i>	TRUE = do not allow offline approval, FALSE = allow ICC to approve offline if terminal capable Note: To request tags to be included in default response, use tag DFEE1A, and specify tag list. Example four tags 9F02, 9F36, 95, 9F37 to be included in response = DFEE1A079F029F369F37
<i>isFastEMV</i>	If TRUE, it will populate the <code>IDTTransactionData.fastEMV</code> with ASCII data similar to IDTech FastEMV KB output, after performing an auto-authenticate and auto-complete with <code>ResultCode = Could Not Contact Host</code>
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString

21.20.2.73 RETURN_CODE IDTechSDK.IDT_VP3300.emv_addTerminalData (byte[] *tlv*, string *ident* = " ")

Add Terminal Data

Adds to the exosting Terminal Data

Parameters

<i>tlv</i>	TerminalData TLV Data
<i>ident</i>	Optional identifier

Return values

<i>RETURN_CODE</i>	Return codes listed as typedef enum in IDTCommon:RETURN_CODE. Values can be parsed with IDT_BTPay::device_getResponseCodeString:()
--------------------	--

21.20.2.74 static void IDTechSDK.IDT_VP3300.emv_allowFallback (bool *allow*, string *ident* = " ") [static]

Allow fallback for EMV transactions. Default is TRUE

Parameters

<i>allow</i>	TRUE = allow fallback, FALSE = don't allow fallback
--------------	---

21.20.2.75 RETURN_CODE IDTechSDK.IDT_VP3300.emv_authenticateTransaction (byte[] *updatedTLV*, string *ident* = " ")

Authenticate EMV Transaction Request

Authenticates the EMV transaction for an ICC card. Execute this after receiving response with result code 0x10 to emv_startTransaction

The tags will be returned in the callback routine.

@param updatedTLV TLV stream that can be used to update the following values:

- 9F02: Amount
- 9F03: Other amount
- 9C: Transaction type
- 5F57: Account type

In addition tag DFEE1A can be sent to specify tag list to include in results. Example four tags 9F02, 9F36, 95

@param ident Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString

21.20.2.76 static void IDTechSDK.IDT_VP3300.emv_autoAuthenticate (bool *authenticate*, string *ident* = " ") [static]

Enables authenticate for EMV transactions. If a startEMVTranaction results in code 0x0010 (start transaction success), then emv_authenticateEMVTransaction can automatically execute if parameter is set to TRUE

Parameters

<i>authenticate</i>	TRUE = auto authenticate, FALSE = manually authenticate
---------------------	---

21.20.2.77 `static void IDTechSDK.IDT_VP3300.emv_autoAuthenticateTags (bool authenticate, byte[] tags, string ident = " ")`
`[static]`

Enables authenticate for EMV transactions. If a startEMVTransaction results in code 0x0010 (start transaction success), then emv_authenticateEMVTransaction can automatically execute if parameter is set to TRUE

Parameters

<i>authenticate</i>	TRUE = auto authenticate, FALSE = manually authenticate
<i>tags</i>	Tags to pass during authentication stage;

21.20.2.78 `RETURN_CODE IDTechSDK.IDT_VP3300.emv_callbackResponseLCD (EMV_LCD_DISPLAY_MODE type, byte selection, string ident = " ")`

Callback Response LCD Display

Provides menu selection responses to the kernel after a callback was received with DeviceState.EMVCallback, and callbackType = EMV_CALLBACK_TYPE.EMV_CALLBACK_TYPE_LCD, and lcd_displayMode = EMV_LCD_DISPLAY_MODE_MENU, EMV_LCD_DISPLAY_MODE_PROMPT, or EMV_LCD_DISPLAY_MODE_LANGUAGE_SELECT

Parameters

<i>type</i>	If Cancel key pressed during menu selection, then value is EMV_LCD_DISPLAY_MODE_CANCEL. Otherwise, value can be EMV_LCD_DISPLAY_MODE_MENU, EMV_LCD_DISPLAY_MODE_PROMPT, or EMV_LCD_DISPLAY_MODE_LANGUAGE_SELECT
<i>selection</i>	If type = EMV_LCD_DISPLAY_MODE_MENU or EMV_LCD_DISPLAY_MODE_LANGUAGE_SELECT, provide the selection ID line number. Otherwise, if type = EMV_LCD_DISPLAY_MODE_PROMPT supply either 0x43 ('C') for Cancel, or 0x45 ('E') for Enter/accept
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with errorCode.getErrorString()

21.20.2.79 `RETURN_CODE IDTechSDK.IDT_VP3300.emv_callbackResponseMSR (byte[] MSR, string ident = " ")`

Callback Response MSR Entry

Provides MSR information to kernel after a callback was received with DeviceState.EMVCallback, and callbackType = EMV_CALLBACK_MSR

Parameters

<i>MSR</i>	Swiped track data
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with `errorCode.getErrorString()`

21.20.2.80 RETURN_CODE IDTechSDK.IDT_VP3300.emv_callbackResponsePIN (EMV_PIN_MODE *type*, byte[] *KSN*, byte[] *PIN*, string *ident* = " ")

Callback Response PIN Entry

Provides (or cancels) PIN entry information to kernel after a callback was received with `DeviceState.EMVCallback`, and `callbackType = EMV_CALLBACK_TYPE_PINPAD`

Parameters

<i>type</i>	If Cancel key pressed during PIN entry, then value is EMV_PIN_MODE_CANCEL. Otherwise, value can be EMV_PIN_MODE_ONLINE_DUKPT, EMV_PIN_MODE_ONLINE_MKSK, or EMV_PIN_MODE_OFFLINE
<i>KSN</i>	If enciphered PIN, this is either the PIN DUKPT Key (EMV_PIN_MODE_ONLINE_DUKPT) or PIN Session Key (EMV_PIN_MODE_ONLINE_MKSK), or PIN Pairing DUKPT key (EMV_PIN_MODE_OFFLINE, string <i>ident</i> = "")
<i>PIN</i>	If enciphered PIN, this is encrypted PIN block. If device does not implement pairing function, this is plaintext PIN
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with `device_getResponseCodeString`

21.20.2.81 RETURN_CODE IDTechSDK.IDT_VP3300.emv_cancelTransaction (string *ident* = " ")

Cancel Transaction

Cancels the currently executing EMV or CTLS transaction.

Parameters

<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.
--------------	--

Returns

RETURN_CODE: Values can be parsed with `errorCode.getErrorString()`

21.20.2.82 RETURN_CODE IDTechSDK.IDT_VP3300.emv_completeTransaction (bool *commError*, byte[] *authCode*, byte[] *iad*, byte[] *tlvScripts*, byte[] *tlv*, string *ident* = " ")

Complete EMV Transaction Request

Completes the EMV transaction for an ICC card when online authorization request is received from `emv_authenticateTransaction`

The tags will be returned in the callback routine.

Parameters

<i>commError</i>	Communication error with host. Set to TRUE if host was unreachable, or FALSE if host response received. If Communication error, authCode, iad, tlvScripts can be null.
<i>authCode</i>	Authorization code from host. Two bytes. Example 0x3030. (Tag value 8A). Required
<i>iad</i>	Issuer Authentication Data, if any. Example 0x11223344556677883030 (tag value 91).
<i>tlvScripts</i>	71/72 scripts, if any
<i>tlv</i>	Additional TVL data to return with transaction results (if any, string ident = "")
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString

NOTE: There are three possible outcomes for Authorization Code: Approval, Refer To Bank, Decline. The kernel maps these three outcomes to valid authorization response codes using tag DFEE1B through 8 bytes: {2 bytes Approval Code}{2 bytes Referral Code}{2 bytes Decline Code}{2 bytes RFU} If your gateway uses "00" for Approval, "01" for Referral, and "05" for Decline, then DFEE1B 08 3030 3031 3035 0000 If you use an authorization code value that is not defined in DFEE1B, the kernel will use the DECLINE value of DFEE1B by default.

21.20.2.83 RETURN_CODE IDTechSDK.IDT_VP3300.emv_getEMVConfigurationCheckValue (ref string *response*, string *ident* = "")

Polls device for EMV Configuration Check Value

Parameters

<i>response</i>	Response returned of Check Value of Configuration
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with errorCode.getErrorString(string ident = "")

21.20.2.84 RETURN_CODE IDTechSDK.IDT_VP3300.emv_getEMVKernelCheckValue (ref string *response*, string *ident* = "")

Polls device for EMV Kernel Check Value

Parameters

<i>response</i>	Response returned of Check Value of Kernel
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with errorCode.getErrorString(string ident = "")

21.20.2.85 RETURN_CODE IDTechSDK.IDT_VP3300.emv_getEMVKernelVersion (ref string *response*, string *ident* = "")

Polls device for EMV Kernel Version

Parameters

<i>response</i>	Response returned of Kernel Version
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with `errorCode.getErrorString(string ident = "")`

21.20.2.86 RETURN_CODE IDTechSDK.IDT_VP3300.emv_getEMVKernelVersionExt (ref string *response*, string *ident* = " ")

Polls device for Extended EMV Kernel Version

Parameters

<i>response</i>	Response returned of Kernel Version
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with `errorCode.getErrorString(string ident = "")`

21.20.2.87 RETURN_CODE IDTechSDK.IDT_VP3300.emv_getTerminalMajorConfiguration (ref int *configuration*, string *ident* = " ")

Get Terminal Major Configuration

Gets the Terminal Data Major Configuration setting

Parameters

<i>configuration</i>	The configuration that is set (1-5, string <i>ident</i> = "")
----------------------	---

Return values

<i>RETURN_CODE</i>	Return codes listed as typedef enum in IDTCommon:RETURN_CODE. Values can be parsed with IDT_NEO2::device_getResponseCodeString :(string <i>ident</i> = "")
--------------------	---

21.20.2.88 RETURN_CODE IDTechSDK.IDT_VP3300.emv_removeAllApplicationData (string *ident* = " ")

Remove All Application Data

Removes all the Application Data for EMV Kernel

Parameters

<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.
--------------	--

Returns

RETURN_CODE: Values can be parsed with `errorCode.getErrorString(string ident = "")`

21.20.2.89 RETURN_CODE IDTechSDK.IDT_VP3300.emv_removeAllCAPK (string *ident* = " ")

Remove All Certificate Authority Public Key

Removes all the CAPK for EMV Kernel

Parameters

<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.
--------------	--

Returns

RETURN_CODE: Values can be parsed with `errorCode.getErrorString(string ident = "")`

21.20.2.90 RETURN_CODE IDTechSDK.IDT_VP3300.emv_removeAllCRL (string *ident* = " ")

Remove All Certificate Revocation List Entries

Removes all CRLEntry entries

Parameters

<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.
--------------	--

Returns

RETURN_CODE: Values can be parsed with `errorCode.getErrorString(string ident = "")`

21.20.2.91 RETURN_CODE IDTechSDK.IDT_VP3300.emv_removeApplicationData (byte[] *AID*, string *ident* = " ")

Remove Application Data by AID

Removes the Application Data for EMV Kernel as specified by the AID name passed as a parameter

Parameters

<i>AID</i>	Name of ApplicationID Must be between 5 and 16 bytes
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with `errorCode.getErrorString(string ident = "")`

21.20.2.92 RETURN_CODE IDTechSDK.IDT_VP3300.emv_removeCAPK (byte[] *capk*, string *ident* = " ")

Remove Certificate Authority Public Key

Removes the CAPK for EMV Kernel as specified by the RID/Index

Parameters

<i>capk</i>	6 byte CAPK = 5 bytes RID + 1 byte INDEX
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with `errorCode.getErrorString(string ident = "")`

21.20.2.93 RETURN_CODE IDTechSDK.IDT_VP3300.emv_removeCRL (*byte[] crlList*, *string ident* = " ")

Remove Certificate Revocation List Entries

Removes CRL entries as specified by the RID and Index and serial number passed as 9 bytes

Parameters

<i>crlList</i>	containing the list of CRL to remove: [CRL1][CRL2]...[CRLn] where each [CRL] is 9 bytes: [5 bytes RID][1 byte CAPK Index][3 bytes serial number]
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with `device_getResponseCodeString`

21.20.2.94 RETURN_CODE IDTechSDK.IDT_VP3300.emv_removeTerminalData (*string ident* = " ")

Remove Terminal Data

Removes the Terminal Data

Parameters

<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.
--------------	--

Returns

RETURN_CODE: Values can be parsed with `device_getResponseCodeString`

21.20.2.95 RETURN_CODE IDTechSDK.IDT_VP3300.emv_retrieveAIDList (*ref byte response[[]]*, *string ident* = " ")

Retrieve AID list

Returns all the AID names installed on the terminal for EMV Kernel.

Parameters

<i>response</i>	array of AID name byte arrays
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with `errorCode.getErrorString(string ident = "")`

21.20.2.96 RETURN_CODE IDTechSDK.IDT_VP3300.emv_retrieveApplicationData (byte[] AID, ref byte[] tlv, string ident = " ")

Retrieve Application Data by AID

Retrieves the Application Data for EMV Kernel as specified by the AID name passed as a parameter.

Parameters

<i>AID</i>	Name of ApplicationID. Must be between 5 and 16 bytes
<i>tlv</i>	The TLV elements of the requested AID
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with `errorCode.getErrorString()`

21.20.2.97 RETURN_CODE IDTechSDK.IDT_VP3300.emv_retrieveCAPK (byte[] capk, ref byte[] key, string ident = " ")

Retrieve Certificate Authority Public Key

Retrieves the CAPK for EMV Kernel as specified by the RID/Index passed as a parameter.

Parameters

<i>capk</i>	6 bytes CAPK = 5 bytes RID + 1 byte Index
<i>key</i>	<p>Response returned as a CAKey format: [5 bytes RID][1 byte Index][1 byte Hash Algorithm][1 byte Encryption Algorithm][20 bytes HashValue][4 bytes Public Key Exponent][2 bytes Modulus Length][Variable bytes Modulus] Where:</p> <ul style="list-style-type: none"> • Hash Algorithm: The only algorithm supported is SHA-1. The value is set to 0x01 • Encryption Algorithm: The encryption algorithm in which this key is used. Currently support only one type: RSA. The value is set to 0x01. • HashValue: Which is calculated using SHA-1 over the following fields: RID & Index & Modulus & Exponent • Public Key Exponent: Actually, the real length of the exponent is either one byte or 3 bytes. It can have two values: 3 (Format is 0x00 00 00 03), or 65537 (Format is 0x00 01 00 01, string ident = "") • Modulus Length: LenL LenH Indicated the length of the next field. • Modulus: This is the modulus field of the public key. Its length is specified in the field above.
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with `errorCode.getErrorString()`

21.20.2.98 **RETURN_CODE** IDTechSDK.IDT_VP3300.emv_retrieveCAPKList (ref byte[] *keys*, string *ident* = " ")

Retrieve the Certificate Authority Public Key list

Returns all the CAPK RID and Index installed on the terminal for EMV Kernel.

Parameters

<i>keys</i>	[key1][key2]...[keyn], each key 6 bytes where key = 5 bytes RID + 1 byte index
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with errorCode.getErrorString()

21.20.2.99 **RETURN_CODE** IDTechSDK.IDT_VP3300.emv_retrieveCRLList (ref byte[] *list*, string *ident* = " ")

Retrieve the Certificate Revocation List

Returns the CRL entries on the terminal.

Parameters

<i>list</i>	[CRL1][CRL2]...[CRLn], each CRL 9 bytes where CRL = 5 bytes RID + 1 byte index + 3 bytes serial number
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with errorCode.getErrorString()

21.20.2.100 **RETURN_CODE** IDTechSDK.IDT_VP3300.emv_retrieveTerminalData (ref byte[] *tlv*, string *ident* = " ")

Retrieve Terminal Data

Retrieves the Terminal Data for EMV Kernel.

Parameters

<i>tlv</i>	Response returned as a TLV
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with errorCode.getErrorString()

21.20.2.101 **RETURN_CODE** IDTechSDK.IDT_VP3300.emv_retrieveTransactionResult (byte[] *tags*, ref IDTTransactionData *tlv*, string *ident* = " ")

Retrieve Transaction Results

Retrieves specified EMV tags from the currently executing transaction.

Parameters

<i>tags</i>	Tags to be retrieved. Example 0x9F028A will retrieve tags 9F02 and 8A
<i>tlv</i>	All requested tags returned as unencrypted, encrypted and masked TLV data in IDTransactionData object
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString

21.20.2.102 RETURN_CODE IDTechSDK.IDT_VP3300.emv_setApplicationData (byte[] *name*, byte[] *tlv*, string *ident* = " ")

Set Application Data by AID

Sets the Application Data for EMV Kernel as specified by the application name and TLV data

Parameters

<i>name</i>	Application name, 10-32 ASCII hex characters representing 5-16 bytes Example "a0000000031010"
<i>tlv</i>	Application data in TLV format
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with errorCode.getErrorString()

21.20.2.103 RETURN_CODE IDTechSDK.IDT_VP3300.emv_setCAPK (byte[] *key*, string *ident* = " ")

Set Certificate Authority Public Key

Sets the CAPK for EMV Kernel as specified by the CAKey structure

Parameters

<i>key</i>	<p>CAKey format: [5 bytes RID][1 byte Index][1 byte Hash Algorithm][1 byte Encryption Algorithm][20 bytes HashValue][4 bytes Public Key Exponent][2 bytes Modulus Length][Variable bytes Modulus] Where:</p> <ul style="list-style-type: none"> • Hash Algorithm: The only algorithm supported is SHA-1. The value is set to 0x01 • Encryption Algorithm: The encryption algorithm in which this key is used. Currently support only one type: RSA. The value is set to 0x01. • HashValue: Which is calculated using SHA-1 over the following fields: RID & Index & Modulus & Exponent • Public Key Exponent: Actually, the real length of the exponent is either one byte or 3 bytes. It can have two values: 3 (Format is 0x00 00 00 03), or 65537 (Format is 0x00 01 00 01, string ident = "") • Modulus Length: LenL LenH Indicated the length of the next field. • Modulus: This is the modulus field of the public key. Its length is specified in the field above.
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with `errorCode.getErrorString()`

21.20.2.104 RETURN_CODE IDTechSDK.IDT_VP3300.emv_setCRL (*byte[] list*, *string ident* = " ")

Set Certificate Revocation List

Sets the CRL

Parameters

<i>list</i>	CRL Entries containing the RID, Index, and serial numbers to set [CRL1][CRL2]...[CRLn] where each [CRL] is 9 bytes: [5 bytes RID][1 byte CAPK Index][3 bytes serial number]
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with `errorCode.getErrorString()`

21.20.2.105 RETURN_CODE IDTechSDK.IDT_VP3300.emv_setTerminalData (*byte[] tlv*, *string ident* = " ")

Set Terminal Data

Sets the Terminal Data

Parameters

<i>tlv</i>	TerminalData TLV Data
<i>ident</i>	Optional identifier

Return values

<i>RETURN_CODE</i>	Return codes listed as typedef enum in IDTCommon:RETURN_CODE. Values can be parsed with <code>IDT_BTPay::device_getResponseCodeString:()</code>
--------------------	---

21.20.2.106 RETURN_CODE IDTechSDK.IDT_VP3300.emv_setTerminalMajorConfiguration (*int configuration*, *string ident* = " ")

Set Terminal Major Configuration

Sets the Terminal Data Major Configuration setting

Parameters

<i>configuration</i>	The configuration to set (1-5, string ident = "")
----------------------	---

Return values

<i>RETURN_CODE</i>	Return codes listed as typedef enum in IDTCommon:RETURN_CODE. Values can be parsed with IDT_BTPay::device_getResponseCodeString:(string ident = "")
--------------------	---

21.20.2.107 **RETURN_CODE** IDTechSDK.IDT_VP3300.emv_startTransaction (double *amount*, double *amtOther*, int *exponent*, int *type*, int *timeout*, byte[] *tags*, bool *forceOnline*, bool *isFastEMV* = false, string *ident* = " ")

Start EMV Transaction Request

Authorizes the EMV transaction for an ICC card

The tags will be returned in the callback routine.

Parameters

<i>amount</i>	Transaction amount value (tag value 9F02, string ident = "")
<i>amtOther</i>	Other amount value, if any (tag value 9F03, string ident = "")
<i>exponent</i>	Number of characters after decimile point
<i>type</i>	Transaction type (tag value 9C).
<i>timeout</i>	Timeout value in seconds.
<i>tags</i>	Any other tags to be included in the request. Passed as TLV Data. Example, tag 9F02 with amount 0x000000000100 would be 0x9F0206000000000100 If tags 9F02 (amount),9F03 (other amount), or 9C (transaction type) are included, they will take priority over these values supplied as individual parameters to this method.
<i>forceOnline</i>	TRUE = do not allow offline approval, FALSE = allow ICC to approve offline if terminal capable Note: To request tags to be included in default response, use tag DFEE1A, and specify tag list. Example four tags 9F02, 9F36, 95, 9F37 to be included in response = DFEE1A079F029F369F37
<i>isFastEMV</i>	If TRUE, it will populate the IDTTransactionData.fastEMV with ASCII data similar to IDTech FastEMV KB output, after performing an auto-authenticate and auto-complete with ResultCode = Could Not Contact Host
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString

21.20.2.108 **RETURN_CODE** IDTechSDK.IDT_VP3300.emv_trySetTerminalData (byte[] *tlv*, ref byte[] *rejectedTLV*, ref byte[] *convertedTLV*, bool *overwrite* = false, string *ident* = " ")

Try to Set Terminal Data

Attempts to set the Terminal Data. Use this functions to attempt to set the terminal data to the device. This function will allow the cross-device tag compatible writing of IDTech Legacy/New tag definitions

Parameters

<i>tlv</i>	TerminalData TLV Data
<i>rejectedTLV</i>	Contains the tags (if any) that were rejected by the firmware
<i>convertedTLV</i>	Contains the tags (if any) that were converted and accepted by the firmware
<i>overwrite</i>	TRUE = add TLV to existing tags, FALSE = replace existing tags with TLV
<i>ident</i>	Optional identifier

Return values

RETURN_CODE	Return codes listed as typedef enum in IDTCommon:RETURN_CODE. Values can be parsed with IDT_BTPay::device_getResponseCodeString:()
--------------------	--

21.20.2.109 `static int IDTechSDK.IDT_VP3300.getCommandTimeout (string ident = " ") [static]`

Get Command Timeout

Gets the default timeout (in milliseconds) waiting for a blocking command response

Return values

<i>time</i>	Time
-------------	------

21.20.2.110 `RETURN_CODE IDTechSDK.IDT_VP3300.icc_exchangeAPDU (string c_APDU, ref byte[] response, string ident = " ")`

Exchange APDU

Sends an APDU packet to the ICC. If successful, response is returned in APDUResult class instance in response parameter.

Parameters

<i>c_APDU</i>	APDU data packet
<i>response</i>	Unencrypted/encrypted parsed APDU response
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with `errorCode.getErrorString(string ident = "")`

21.20.2.111 `RETURN_CODE IDTechSDK.IDT_VP3300.icc_getICCReaderStatus (ref byte status, string ident = " ")`

Get Reader Status

Returns the reader status

Parameters

<i>status</i>	Pointer that will return with the ICCReaderStatus results. bit 0: 0 = ICC Power Not Ready, 1 = ICC Powered bit 1: 0 = Card not seated, 1 = card seated
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with `device_getResponseCodeString`

21.20.2.112 `RETURN_CODE IDTechSDK.IDT_VP3300.icc_powerOffICC (string ident = " ")`

Power Off ICC

Powers down the ICC

Parameters

<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.
--------------	--

Returns

RETURN_CODE: Values can be parsed with `errorCode.getErrorString(string ident = "")`

If Success, empty If Failure, ASCII encoded data of error string

21.20.2.113 RETURN_CODE IDTechSDK.IDT_VP3300.icc_powerOnICC (ref byte[] *ATR*, byte *interfaces*, string *ident* = " ")

Power On ICC

Power up the currently selected microprocessor card in the ICC reader

Parameters

<i>interfaces</i>	For NEO2 devices allowed interfaces for which to get the ATR. 0x20h = retrieve last ATR received from PICC 0x21h = SAM1 (SRED version only, string ident = "") 0x22h = SAM2 (SRED version only, string ident = "") For other devices interfaces equals to 0s
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with `errorCode.getErrorString(string ident = "")`

21.20.2.114 static void IDTechSDK.IDT_VP3300.lcd_retrieveMessage (DisplayMessages.DISPLAY_MESSAGE_LANGUAGE *lang*, DisplayMessages.DISPLAY_MESSAGE_IDENTIFIER *id*, ref string *line1*, ref string *line2*) [static]

Retrieve LCD Message

Returns the string value for a message ID returned for LCD messaging

Parameters

<i>lang</i>	Language.
<i>id</i>	Message ID
<i>line1</i>	Line 1 string value
<i>line2</i>	Line 2 string value
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with `device_getResponseCodeString`

21.20.2.115 RETURN_CODE IDTechSDK.IDT_VP3300.msr_cancelMSRSwipe (string *ident* = " ")

Disable MSR Swipe Cancels MSR swipe request.

Parameters

<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.
--------------	--

Returns

RETURN_CODE: Values can be parsed with `errorCode.getErrorString(string ident = "")`

21.20.2.116 `RETURN_CODE IDTechSDK.IDT_VP3300.msr_startMSRSwipe (int timeout, string ident = " ")`

Enable MSR Swipe

Enables MSR, waiting for swipe to occur.

Parameters

<i>timeout</i>	Swipe Timeout Value
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with `errorCode.getErrorString(string ident = "")`

21.20.2.117 `static String IDTechSDK.IDT_VP3300.SDK_Version () [static]`

SDK Version

- All Devices

Returns the current version of SDK

Returns

Framework version

21.20.2.118 `static void IDTechSDK.IDT_VP3300.setCallback (CallBack my_Callback) [static]`

Set Callback

Sets the class callback

21.20.2.119 `static void IDTechSDK.IDT_VP3300.setCallback (IntPtr my_Callback, SynchronizationContext context) [static]`

Set Callback

Sets the class callback

Parameters

<i>my_Callback</i>	The callback function to receive the response message from device. defined as follows. <code>public unsafe delegate void MFCCallBack(Parameters parameters, ident);</code>
<i>context</i>	The context of the UI thread

21.20.2.120 `static void IDTechSDK.IDT_VP3300.setCommandTimeout (int milliseconds, string ident = " ") [static]`

Set Command Timeout

Sets the default timeout (in milliseconds) waiting for a blocking command response

Parameters

<i>milliseconds</i>	Time
---------------------	------

21.20.2.121 `static bool IDTechSDK.IDT_VP3300.useSerialPort (int port) [static]`

Use Serial Port Interface

Instructs SDK to attempt to use the Serial Port for communication with [IDT_Vendi](#) using default baud rate

Parameters

<i>port</i>	Serial Port to use. Example COM1 = 1.
-------------	---------------------------------------

Returns

bool TRUE=successful, FALSE=failure

21.20.2.122 `static bool IDTechSDK.IDT_VP3300.useSerialPort (int port, int baud) [static]`

Use Serial Port Interface with baud rate

Instructs SDK to attempt to use the Serial Port for communication with [IDT_Vendi](#)

Parameters

<i>port</i>	Serial Port to use. Example COM1 = 1.
<i>baud</i>	Baud rate to override default. Example 115200;

Returns

bool TRUE=successful, FALSE=failure

21.20.3 Property Documentation

21.20.3.1 `IDT_VP3300 IDTechSDK.IDT_VP3300.SharedController [static],[get]`

Singleton Instance

Establishes an singleton instance of [IDT_VP3300](#) class.

Returns

Instance of [IDT_VP3300](#)

The documentation for this class was generated from the following file:

- `/Users/randy/Repo/universal-sdk/IDTechSDK_STD/DeviceHeaders/IDT_VP3300.cs`

21.21 IDTechSDK.IDT_VP8800 Class Reference

Public Member Functions

- RETURN_CODE [config_setEthernetMACAddress](#) (byte[] address, string ident="")
- RETURN_CODE [config_getEthernetMACAddress](#) (ref byte[] address, string ident="")
- RETURN_CODE [config_getPINKeySlot](#) (ref int slot, string ident="")
- RETURN_CODE [config_setPINKeySlot](#) (int slot, string ident="")
- RETURN_CODE [config_getDataKeySlot](#) (ref int slot, string ident="")
- RETURN_CODE [config_setDataKeySlot](#) (int slot, string ident="")
- RETURN_CODE [config_getStatusKeySlots](#) (ref byte[] keyslots, string ident="")
- RETURN_CODE [config_getKeySlotInfo](#) (int index, int slot, ref byte[] keyslot, string ident="")
- RETURN_CODE [config_getNetworkConfiguration](#) (ref bool isStatic, ref string address, ref string subnet, ref string gateway, ref string dns, string ident="")
- RETURN_CODE [config_setNetworkConfiguration](#) (bool isStatic, string address, string subnet, string gateway, string dns, string ident="")
- RETURN_CODE [config_getSerialNumber](#) (ref string response, string ident="")
- RETURN_CODE [device_getFirmwareVersion](#) (ref string response, string ident="")
- RETURN_CODE [device_getMerchantRecord](#) (int index, ref byte[] record, string ident="")
- RETURN_CODE [device_retrieveAIDList](#) (ref byte[][] response, string ident="")
- RETURN_CODE [device_sendDataCommand](#) (string cmd, bool calcLRC, ref byte[] response, string ident="")
- RETURN_CODE [device_sendDataCommand_ext](#) (string cmd, bool calcLRC, ref byte[] response, int timeout, bool noResponse, string ident="")
- RETURN_CODE [msr_flushTrackData](#) (string ident="")
- RETURN_CODE [msr_startMSRSwipe](#) (int timeout, string ident="")
- RETURN_CODE [lcd_customDisplayMode](#) (bool enable, string ident="")
- RETURN_CODE [lcd_displayText](#) (int xCord, int yCord, int xWidth, int yWidth, int font, int style, int displayProp, string text, ref byte[] identifier, string ident="")
- RETURN_CODE [lcd_displayButton](#) (int xCord, int yCord, int xWidth, int yWidth, int font, int style, int displayProp, string text, ref byte[] identifier, byte[] foreRGB=null, byte[] backRGB=null, string ident="")
- RETURN_CODE [device_updateDeviceFirmware](#) (byte[] firmwareData, string ident="")
- RETURN_CODE [device_startRKI](#) (bool isTest, string ident="")
- RETURN_CODE [lcd_setForeBackColor](#) (byte[] foreRGB, byte[] backRGB, string ident="")
- RETURN_CODE [lcd_captureSignature](#) (int timeout, int format, string ident="")
- RETURN_CODE [lcd_clearInputEvents](#) (string ident="")
- RETURN_CODE [lcd_getInputEvent](#) (byte timeout, string ident="")
- RETURN_CODE [device_sendPAE](#) (string command, ref string response, int timeout, string ident="")
- RETURN_CODE [lcd_resetInitialState](#) (string ident="")
- RETURN_CODE [msr_cancelMSRSwipe](#) (string ident="")
- RETURN_CODE [emv_cancelTransaction](#) (string ident="")
- RETURN_CODE [ctls_cancelTransaction](#) (string ident="")
- RETURN_CODE [ctls_displayOnlineAuthResult](#) (bool isOK, byte[] TLV, string ident="")
- RETURN_CODE [ctls_displayOnlineAuthResult_ext](#) (byte statusCode, byte[] TLV, string ident="")
- RETURN_CODE [emv_startTransaction](#) (double amount, double amtOther, int exponent, int type, int timeout, byte[] tags, bool forceOnline, string ident="")
- RETURN_CODE [emv_activateTransaction](#) (int timeout, byte[] tags, bool forceOnline, bool isFastEMV=false, string ident="")
- RETURN_CODE [emv_continueTransactionForCV](#) (int result, byte[] pinblock, string ident="")
- RETURN_CODE [ctls_startTransaction](#) (double amount, double amtOther, int exponent, int type, int timeout, byte[] tags, bool forceOnline, bool isFastEMV=false, string ident="")
- RETURN_CODE [ctls_activateTransaction](#) (int timeout, byte[] tags, bool forceOnline, bool isFastEMV=false, string ident="")
- RETURN_CODE [emv_completeTransaction](#) (bool commError, byte[] authCode, byte[] iad, byte[] tlvScripts, byte[] tlv, string ident="")

- RETURN_CODE [emv_callbackResponseLCD](#) (EMV_LCD_DISPLAY_MODE type, byte selection, string ident="")
- RETURN_CODE [emv_callbackResponsePIN](#) (EMV_PIN_MODE type, byte[] KSN, byte[] PIN, string ident="")
- RETURN_CODE [emv_callbackResponseMSR](#) (byte[] MSR, string ident="")
- RETURN_CODE [emv_getEMVKernelCheckValue](#) (ref string response, string ident="")
- RETURN_CODE [emv_clearTransactionLog](#) (string ident="")
- RETURN_CODE [emv_getCRLStatus](#) (ref byte[] status, string ident="")
- RETURN_CODE [emv_getTransactionLogRecord](#) (ref byte[] record, ref int remaining, string ident="")
- RETURN_CODE [emv_getExemptionLogStatus](#) (ref byte[] status, string ident="")
- RETURN_CODE [emv_getTransactionLogStatus](#) (ref byte[] status, string ident="")
- RETURN_CODE [emv_getEMVConfigurationCheckValue](#) (ref string response, string ident="")
- RETURN_CODE [emv_getEMVKernelVersionExt](#) (ref string response, string ident="")
- RETURN_CODE [emv_getEMVKernelVersion](#) (ref string response, string ident="")
- RETURN_CODE [emv_retrieveTransactionResult](#) (byte[] tags, ref IDTTransactionData tlv, string ident="")
- RETURN_CODE [emv_removeApplicationData](#) (byte[] AID, string ident="")
- RETURN_CODE [ctls_removeApplicationData](#) (byte[] AID, string ident="")
- RETURN_CODE [emv_removeCAPK](#) (byte[] capk, string ident="")
- RETURN_CODE [ctls_removeCAPK](#) (byte[] capk, string ident="")
- RETURN_CODE [emv_removeAllCAPK](#) (string ident="")
- RETURN_CODE [ctls_removeAllCAPK](#) (string ident="")
- RETURN_CODE [emv_removeCRL](#) (byte[] crl, string ident="")
- RETURN_CODE [emv_removeAllCRL](#) (string ident="")
- RETURN_CODE [emv_retrieveApplicationData](#) (byte[] AID, ref byte[] tlv, string ident="")
- RETURN_CODE [ctls_retrieveApplicationData](#) (byte[] AID, ref byte[] tlv, string ident="")
- RETURN_CODE [emv_removeTerminalData](#) (string ident="")
- RETURN_CODE [emv_removeTransactionAmountLog](#) (string ident="")
- RETURN_CODE [emv_retrieveAIDList](#) (ref byte[][] response, string ident="")
- RETURN_CODE [ctls_retrieveAIDList](#) (ref byte[][] response, string ident="")
- RETURN_CODE [emv_retrieveCAPK](#) (byte[] capk, ref byte[] key, string ident="")
- RETURN_CODE [ctls_retrieveCAPK](#) (byte[] capk, ref byte[] key, string ident="")
- RETURN_CODE [emv_retrieveCAPKList](#) (ref byte[] keys, string ident="")
- RETURN_CODE [device_controlLED](#) (byte indexLED, byte control, string ident="")
- RETURN_CODE [ctls_retrieveCAPKList](#) (ref byte[] keys, string ident="")
- RETURN_CODE [emv_retrieveCRLList](#) (ref byte[] list, string ident="")
- RETURN_CODE [emv_retrieveExceptionList](#) (ref byte[] list, string ident="")
- RETURN_CODE [ctls_retrieveTerminalData](#) (ref byte[] tlv, string ident="")
- RETURN_CODE [emv_retrieveTerminalData](#) (ref byte[] tlv, string ident="")
- RETURN_CODE [emv_setApplicationData](#) (byte[] tlv, string ident="")
- RETURN_CODE [ctls_setApplicationData](#) (byte[] tlv, string ident="")
- RETURN_CODE [ctls_setDefaultConfiguration](#) (string ident="")
- RETURN_CODE [ctls_setConfigurationGroup](#) (byte[] tlv, string ident="")
- RETURN_CODE [emv_setCAPK](#) (byte[] key, string ident="")
- RETURN_CODE [ctls_setCAPK](#) (byte[] key, string ident="")
- RETURN_CODE [emv_setCRL](#) (byte[] crl, string ident="")
- RETURN_CODE [emv_setException](#) (byte[] exception, string ident="")
- RETURN_CODE [emv_removeException](#) (byte[] exception, string ident="")
- RETURN_CODE [emv_removeAllExceptions](#) (string ident="")
- RETURN_CODE [emv_trySetTerminalData](#) (byte[] tlv, ref byte[] rejectedTLV, ref byte[] convertedTLV, bool overwrite=false, string ident="")
- RETURN_CODE [emv_setTerminalDataVP8800](#) (byte[] tlv, int config, string ident="")
- RETURN_CODE [emv_addTerminalData](#) (byte[] tlv, string ident="")
- RETURN_CODE [ctls_setTerminalData](#) (byte[] tlv, string ident="")
- RETURN_CODE [ctls_trySetTerminalData](#) (byte[] tlv, ref byte[] rejectedTLV, ref byte[] convertedTLV, string ident="")
- RETURN_CODE [device_pingDevice](#) (string ident="")

- RETURN_CODE [device_buzzer](#) (string ident="")
- RETURN_CODE [device_calibrateParameters](#) (byte delta, string ident="")
- RETURN_CODE [device_cancelTransaction](#) (string ident="")
- RETURN_CODE [device_controlIndicator](#) (int indicator, bool enable, string ident="")
- RETURN_CODE [device_controlUserInterface](#) (byte[] values, string ident="")
- RETURN_CODE [device_sendVivoCommandP2](#) (byte command, byte subCommand, byte[] data, ref byte[] response, string ident="")
- RETURN_CODE [device_sendVivoCommandP2_ext](#) (byte command, byte subCommand, byte[] data, ref byte[] response, int timeout, bool noResponse, string ident="")
- RETURN_CODE [device_sendVivoCommandP3](#) (byte command, byte subCommand, byte[] data, ref byte[] response, string ident="")
- RETURN_CODE [device_sendVivoCommandP3_ext](#) (byte command, byte subCommand, byte[] data, ref byte[] response, int timeout, bool noResponse, string ident="")
- RETURN_CODE [ctls_getConfigurationGroup](#) (int group, ref byte[] tlv, string ident="")
- RETURN_CODE [emv_getConfigurationGroup](#) (int group, ref byte[] tlv, string ident="")
- RETURN_CODE [device_getConfigurationGroup](#) (int group, ref byte[] tlv, string ident="")
- RETURN_CODE [emv_resetConfigurationGroup](#) (int group, string ident="")
- RETURN_CODE [device_resetConfigurationGroup](#) (int group, string ident="")
- RETURN_CODE [ctls_resetConfigurationGroup](#) (int group, string ident="")
- RETURN_CODE [ctls_removeConfigurationGroup](#) (int group, string ident="")
- RETURN_CODE [emv_removeConfigurationGroup](#) (int group, string ident="")
- RETURN_CODE [device_removeConfigurationGroup](#) (int group, string ident="")
- RETURN_CODE [device_setBurstMode](#) (byte mode, string ident="")
- RETURN_CODE [device_listDirectory](#) (string directoryName, bool recursive, bool onSD, ref string directory, string ident="")
- RETURN_CODE [device_createDirectory](#) (string directoryName, string ident="")
- RETURN_CODE [device_transferFile](#) (string fileName, byte[] file, bool isSD=false, string ident="")
- RETURN_CODE [device_getDriveFreeSpace](#) (ref int free, ref int used, string ident="")
- RETURN_CODE [device_deleteFile](#) (string filename, bool isSD=false, string ident="")
- RETURN_CODE [lcd_setBackgroundImage](#) (string file, bool enable, string ident="")
- RETURN_CODE [lcd_setDisplayImage](#) (string file, int posX, int posY, int posMode, bool touchEnable, bool clearScreen, string ident="")
- RETURN_CODE [lcd_startSlideShow](#) (string files, int posX, int posY, int posMode, bool touchEnable, bool recursion, bool touchTerminate, int delay, int loops, bool clearScreen, string ident="")
- RETURN_CODE [device_deleteDirectory](#) (string filename, string ident="")
- RETURN_CODE [device_setBuzzerLED](#) (byte buzzer, byte led, bool ledON, string ident="")
- RETURN_CODE [device_setPollMode](#) (byte mode, string ident="")
- RETURN_CODE [device_setMerchantRecord](#) (int index, bool enabled, string merchantID, string merchantURL, string ident="")
- RETURN_CODE [device_startTransaction](#) (double amount, double amtOther, int exponent, int type, int timeout, byte[] tags, byte[] options, bool isFastEMV=false, int transMode=0, string ident="")
- RETURN_CODE [device_activateTransaction](#) (int timeout, byte[] tags, byte[] options, bool isFastEMV=false, string ident="")
- RETURN_CODE [icc_powerOffICC](#) (string ident="")
- RETURN_CODE [icc_powerOnICC](#) (ref byte[] ATR, byte interfaces, string ident="")
- RETURN_CODE [icc_exchangeAPDU](#) (string c_APDU, ref byte[] response, string ident="")
- RETURN_CODE [device_enablePassThrough](#) (bool enablePassThrough, string ident="")
- string [device_getResponseCodeString](#) (RETURN_CODE eCode)
- RETURN_CODE [device_getTransactionResults](#) (ref IDTTransactionData results, string ident="")
- RETURN_CODE [pin_getEncryptedOnlinePIN](#) (int keyType, int timeout, string ident="")
- RETURN_CODE [pin_promptCreditDebit](#) (object currencySymbol, string displayAmount, int timeout, string ident="")
- RETURN_CODE [pin_getPAN](#) (bool getCSC, int timeout, string ident="")
- RETURN_CODE [device_StartRKI](#) (int type, string ident="")
- RETURN_CODE [felica_authentication](#) (byte[] key, string ident="")

- RETURN_CODE [felica_SendCommand](#) (byte[] command, ref byte[] response, string ident="")
- RETURN_CODE [felica_readWithMac](#) (int numBlocks, byte[] blockList, ref byte[] blocks, string ident="")
- RETURN_CODE [felica_writeWithMac](#) (int blockNumber, byte[] data, string ident="")
- RETURN_CODE [felica_read](#) (byte[] serviceCode, int numBlocks, byte[] blockList, ref byte[] blocks, string ident="")
- RETURN_CODE [felica_write](#) (byte[] serviceCode, int blockCount, byte[] blockList, byte[] data, ref byte[] statusFlag, string ident="")
- RETURN_CODE [ctls_nfcCommand](#) (byte[] nfcCmdPkt, ref byte[] response, string ident="")
- RETURN_CODE [felica_requestService](#) (byte[] nodeCode, ref byte[] response, string ident="")
- bool [createFastEMVData](#) (ref IDTTransactionData cData, string ident="")
- RETURN_CODE [device_updateFirmwareFromZip](#) (byte[] zipfile, string ident="", bool isForeground=false)
- RETURN_CODE [device_sendConfigurationFromZip](#) (byte[] zip, string filename, VC_OPERATION_TYPE type, bool matchFW, string ident="", bool isForeground=false)
- RETURN_CODE [device_sendConfiguration](#) (string filename, VC_OPERATION_TYPE type, bool matchFW, string ident="", bool isForeground=false)
- RETURN_CODE [device_readConfigurationToMemory](#) (string memo, Configs.InstallRules rules, List< Configs.DeviceCommand > cmds, string ident="", bool isForeground=false)
- RETURN_CODE [device_getConfigurationFromMemory](#) (ref string json, string ident="")
- RETURN_CODE [device_RemoteKeyInjection](#) (RKI_KEY_TYPE type, string keyName, string ident="", bool performOnForeground=false)

Static Public Member Functions

- static bool [useSerialPort](#) (int port)
- static int [getCommandTimeout](#) (string ident="")
- static void [setCommandTimeout](#) (int milliseconds, string ident="")
- static bool [useSerialPort](#) (int port, int baud, string ident="")
- static bool [useSerialPortLinux](#) (string path)
- static bool [useSerialPortLinux](#) (string path, int baud)
- static void [initSC](#) ()
- static void [setCallback](#) (Callback my_Callback)
- static void [setCallback](#) (IntPtr my_Callback, SynchronizationContext context)
- static String [SDK_Version](#) ()
- static RETURN_CODE [lcd_clearDisplay](#) (string ident="")
- static void [emv_autoAuthenticate](#) (bool authenticate, string ident="")
- static void [emv_autoAuthenticateTags](#) (bool authenticate, byte[] tags, string ident="")
- static void [emv_allowFallback](#) (bool allow, string ident="")
- static void [lcd_retrieveMessage](#) (DisplayMessages.DISPLAY_MESSAGE_LANGUAGE lang, DisplayMessages.DISPLAY_MESSAGE_IDENTIFIER id, ref string line1, ref string line2)

Properties

- static [IDT_VP8800 SharedController](#) [get]

21.21.1 Member Function Documentation

21.21.1.1 RETURN_CODE IDTechSDK.IDT_VP8800.config_getDataKeySlot (ref int slot, string ident = " ")

Get Data Key Slot

Returns the current key slot (in index #2) being used for Data encryption

Parameters

<i>slot</i>	Current key slot, valid values 0-9
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString

21.21.1.2 RETURN_CODE IDTechSDK.IDT_VP8800.config_getEthernetMACAddress (ref byte[] address, string ident = " ")

Get Device Ethernet MAC Address

Parameters

<i>address</i>	6-byte MAC Address
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString

21.21.1.3 RETURN_CODE IDTechSDK.IDT_VP8800.config_getKeySlotInfo (int index, int slot, ref byte[] keyslot, string ident = " ")

Get Key Slot Info

This command checks whether a valid DUKPT key is stored at the specified index and slot and if a valid key is found then some basic information related to the type of key is returned

Parameters

<i>index</i>	Key index: 0x01 = PIN, 0x02 = Data, 0x05 = MAC, 0x14 = KEK
<i>slot</i>	Key Slot: PIN/Data = 0-9, MAC = 0, KEK = 0 Key Slot Data. If 0x00, no key at the slot, otherwise: <ul style="list-style-type: none"> Byte 0 - Key State: 0x00 = Unused, 0x01 = Valid, 0x02 = End of Life, 0xFF = Not Available Byte 1-2 - Key Usage: "K0" = Key Encryption or Wrapping, "P0" = PIN Encryption, "D0" = Data Encryption, "M0" = MAC Verification, "B1" = IPEK Byte 3 - Algorithm: "T" = TDES, "D" = DES, "A" = AES Byte 4 - Mode of Use: "N" = No restriction, "E" = Encryption Only, "D" = Decryption Only, "B" = Both Encryption/Decryption
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString

21.21.1.4 RETURN_CODE IDTechSDK.IDT_VP8800.config_getNetworkConfiguration (ref bool isStatic, ref string address, ref string subnet, ref string gateway, ref string dns, string ident = " ")

Get Device Network Configuration

Parameters

<i>isStatic</i>	TRUE = Static IP, FALSE = DHCP
<i>address</i>	Device IP Address as string. Example "192.168.1.15"

Parameters

<i>subnet</i>	Device Subnet as string. Example "255.255.255.0"
<i>gateway</i>	Device Gateway address as a string. Example "8.8.8.8"
<i>dns</i>	Device DNS address as string. Example "192.168.1.22"
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString

21.21.1.5 RETURN_CODE IDTechSDK.IDT_VP8800.config_getPINKeySlot (ref int slot, string ident = " ")

Get PIN Key Slot

Returns the current key slot (in index #1) being used for PIN encryption

Parameters

<i>slot</i>	Current key slot, valid values 0-9
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString

21.21.1.6 RETURN_CODE IDTechSDK.IDT_VP8800.config_getSerialNumber (ref string response, string ident = " ")

Polls device for Serial Number

Parameters

<i>response</i>	Returns Serial Number
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString

21.21.1.7 RETURN_CODE IDTechSDK.IDT_VP8800.config_getStatusKeySlots (ref byte[] keyslots, string ident = " ")

Get status of the key slots

There are 10 keyslots for PIN Key at Index 1, slot #0-9 There are 10 keyslots for DATA Key at Index 2, slots #0-9
There is a keyslot for MAC Key at Index 5, slot 0 There is a keyslot for Key Encryption Key at Index 14, slot 0

This routine will return all the key slots values as one continuous byte[] of 4-byte key slot info.

- Byte 0 = Key Index Number
- Byte 1-2 = Key Slot Number
- Byte 3 = State: 0x00 = Unused, 0x01 = Valid, 0x02 = End of life, 0xFF = Not Available

Parameters

<i>keySlots</i>	Key slot data
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString

21.21.1.8 RETURN_CODE IDTechSDK.IDT_VP8800.config_setDataKeySlot (int *slot*, string *ident* = " ")

Set Data Key Slot

Sets the key slot to use for Data encryption (in index #2, string *ident* = "")

Parameters

<i>slot</i>	Key slot to use, valid values 0-9
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString

21.21.1.9 RETURN_CODE IDTechSDK.IDT_VP8800.config_setEthernetMACAddress (byte[] *address*, string *ident* = " ")

Set Device Ethernet MAC Address

Parameters

<i>address</i>	6-byte MAC Address
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString

21.21.1.10 RETURN_CODE IDTechSDK.IDT_VP8800.config_setNetworkConfiguration (bool *isStatic*, string *address*, string *subnet*, string *gateway*, string *dns*, string *ident* = " ")

Set Device Network Configuration

Parameters

<i>isStatic</i>	TRUE = Static IP, FALSE = DHCP
<i>address</i>	Device IP Address as string. Example "192.168.1.15"
<i>subnet</i>	Device Subnet as string. Example "255.255.255.0"
<i>gateway</i>	Device Gateway address as a string. Example "8.8.8.8"
<i>dns</i>	Device DNS address as string. Example "192.168.1.22"
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString

21.21.1.11 RETURN_CODE IDTechSDK.IDT_VP8800.config_setPINKeySlot (int slot, string ident = " ")

Set PIN Key Slot

Sets the key slot to use for PIN encryption (in index #1, string ident = "")

Parameters

<i>slot</i>	Key slot to use, valid values 0-9
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString

21.21.1.12 bool IDTechSDK.IDT_VP8800.createFastEMVData (ref IDTTTransactionData cData, string ident = " ")

Create Fast EMV Data

At the completion of a Fast EMV Transaction, after the final card decision is returned and the IDTTTransactionData object is provided, sending that cData object to this method will populate the .fastEMV element with string data that represents the Fast EMV data that would be returned from and IDTech FastEMV over KB protocol

Parameters

<i>cData</i>	The IDTTTransactionData object populated with card data.
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with errorCode.getErrorString(string ident = "")

21.21.1.13 RETURN_CODE IDTechSDK.IDT_VP8800.ctls_activateTransaction (int timeout, byte[] tags, bool forceOnline, bool isFastEMV = false, string ident = " ")

Start a CTLS Transaction Request

Authorizes the CTLS transaction for an CTLS card

The tags will be returned in the callback routine.

Parameters

<i>timeout</i>	Timeout value in seconds.
<i>tags</i>	The tags to be included in the request. Passed as TLV Data. Example, tag 9F02 with amount 0x000000000100 would be 0x9F0206000000000100
<i>forceOnline</i>	TRUE = do not allow offline approval, FALSE = allow ICC to approve offline if terminal capable
<i>isFastEMV</i>	If TRUE, it will populate the IDTTTransactionData.fastEMV with ASCII data similar to IDTech FastEMV KB output, after performing an auto-authenticate and auto-complete with ResultCode = Could Not Contact Host
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with `errorCode.getErrorString()`

NOTE ON APPLEPAY VAS: To enable ApplePay VAS, first a merchant record must be defined in one of the six available index positions (1-6) using `device_setMerchantRecord`, then container tag FFEE06 must be sent as part of the additional tags parameter of `ctls_startTransaction`. Tag FFEE06 must contain tag 9F26 and 9F22, and can optionally contain tags 9F2B and DFO1. Example FFEE06189F220201009F2604000000009F2B050100000000↵ DF010101 9F22 = two bytes = ApplePay Terminal Application Version Number. Hard defined as 0100 for now. (required, string ident = "") 9F26 = four bytes = ApplePay Terminal Capabilities Information (required, string ident = "")

- Byte 1 = RFU
- Byte 2 = Terminal Type
 - - Bit 7 = VAS Support (1=on, 0 = off, string ident = "")
 - - Bit 6 = Touch ID Required (1=on, 0 = off, string ident = "")
 - - Bit 5 = RFU
 - - Bit 4 = RFU
 - - Bit 0,1,2,3
 - - - 0 = Payment Terminal
 - - - 1 = Transit Terminal
 - - - 2 = Access Terminal
 - - - 3 = Wireless Handoff Terminal
 - - - 4 = App Handoff Terminal
 - - - 15 = Other Terminal
- Byte 3 = RFU
- Byte 4 = Terminal Mode
 - - 0 = ApplePay VAS OR ApplePay
 - - 1 = ApplePay VAS AND ApplePay
 - - 2 = ApplePay VAS ONLY
 - - 3 = ApplePay ONLY 9F2B = 5 bytes = ApplePay VAS Filter. Each byte filters for that specific merchant index (optional, string ident = "") DF01 = 1 byte = ApplePay VAS Protocol. (optional, string ident = "")
 - - Bit 0 : 1 = URL VAS, 0 = Full VAS
 - - Bit 1 : 1 = VAS Beeps, 0 = No VAS Beeps
 - - Bit 2 : 1 = Silent Comm Error, 2 = EMEA Comm Error
 - - Bit 3-7 : RFU

21.21.1.14 RETURN_CODE IDTechSDK.IDT_VP8800.ctls_cancelTransaction (string ident = " ")

Cancel Transaction

Cancels the currently executing EMV or CTLS transaction.

Parameters

<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.
--------------	--

Returns

RETURN_CODE: Values can be parsed with `errorCode.getErrorString()`

21.21.1.15 RETURN_CODE IDTechSDK.IDT_VP8800.ctls_displayOnlineAuthResult (bool *isOK*, byte[] *TLV*, string *ident* = " ")

Display Online Authorization Result

Use this command to display the status of an online authorization request on the reader's display (OK or NOT OK). Use this command after the reader sends an online request to the issuer.

Parameters

<i>isOK</i>	TRUE = OK, FALSE = NOT OK, string <i>ident</i> = ""
<i>TLV</i>	Optional TLV for AOSA
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with `device_getResponseCodeString`

21.21.1.16 RETURN_CODE IDTechSDK.IDT_VP8800.ctls_displayOnlineAuthResult_ext (byte *statusCode*, byte[] *TLV*, string *ident* = " ")

Display Online Authorization Result Extended

Use this command to display the status of an online authorization request on the reader's display (OK, NOT OK or ARC). Use this command after the reader sends an online request to the issuer.

Parameters

<i>statusCode</i>	0 = NOT OK, 1 = OK, 2 = ARC
<i>TLV</i>	Optional TLV for AOSA
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with `device_getResponseCodeString`

21.21.1.17 RETURN_CODE IDTechSDK.IDT_VP8800.ctls_getConfigurationGroup (int *group*, ref byte[] *tlv*, string *ident* = " ")

Get Configuration Group

Retrieves the Configuration for the specified Group. Group 0 = terminal settings.

Parameters

<i>group</i>	Configuration Group
--------------	---------------------

Parameters

<i>tlv</i>	return data
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with `errorCode.getErrorString()`

21.21.1.18 RETURN_CODE IDTechSDK.IDT_VP8800.ctls_nfcCommand (byte[] *nfcCmdPkt*, ref byte[] *response*, string *ident* = " ")

NFC Command

This command uses `nfcCmdPkt[0]` in command data field to implement different functions. This command should be used in Pass-Through mode and command with "Poll for a NFC Tag" data should be used first. Command with other data can only be used once the "Poll for a NFC Tag" command has indicated that a NFC tag is present.

Parameters

<i>nfcCmdPkt</i>	<p>System Code</p> <ul style="list-style-type: none"> • Poll for NFC Tag: nfcCmdPkt[0] = 0xff, nfcCmdPkt[1] = timeout value (in seconds, string ident = "") • Tag1 Static Get All Data: nfcCmdPkt[0] = 0x11 • Tag1 Static Read a Byte: nfcCmdPkt[0] = 0x12, nfcCmdPkt[1] = Address of Data • Tag1 Static Write a Byte: nfcCmdPkt[0] = 0x13 nfcCmdPkt[1] = Address of Data, nfcCmdPkt[2] = Data to be written • Tag1 Static Write a Byte NE: nfcCmdPkt[0] = 0x14, nfcCmdPkt[1] = Address of Data, nfcCmdPkt[2] = Data to be written • Tag1 Dynamic Read a Segment: nfcCmdPkt[0] = 0x15, nfcCmdPkt[1] = Address of Segment • Tag1 Dynamic Read 8 Bytes: nfcCmdPkt[0] = 0x16, nfcCmdPkt[1] = Address of Data • Tag1 Dynamic Write 8 Bytes: nfcCmdPkt[0] = 0x17, nfcCmdPkt[1] = Address of Data, nfcCmdPkt[2]~nfcCmdPkt[9] = Data to be written • Tag1 Dynamic Write 8 Bytes NE: nfcCmdPkt[0] = 0x18, nfcCmdPkt[1] = Address of Data, nfcCmdPkt[2]~nfcCmdPkt[9] = Data to be written • Tag2 Read Data (16 bytes): nfcCmdPkt[0] = 0x21, nfcCmdPkt[1] = Address of Data • Tag2 Write Data (4 bytes): nfcCmdPkt[0] = 0x22, nfcCmdPkt[1] = Address of Data, nfcCmdPkt[2]~nfcCmdPkt[5] = Data to be written • Tag2 Select Sect: nfcCmdPkt[0] = 0x23, nfcCmdPkt[1] = Sect number • Tag3 Read Data: – nfcCmdPkt[0] = 0x41, – nfcCmdPkt[1] = Number of services, value n – nfcCmdPkt[2]~nfcCmdPkt[2n+1]: Service code list – nfcCmdPkt[2n+2]: Number of blocks, value m. – nfcCmdPkt[2n+3....]: Block list, length is 2m~3m • Tag3 Write Data: – nfcCmdPkt[0] = 0x41, – nfcCmdPkt[1] = Number of services, value n – nfcCmdPkt[2]~nfcCmdPkt[2n+1]: Service code list – nfcCmdPkt[2n+2]: Number of blocks, value m. – nfcCmdPkt[2n+3....]: Block list, length is 2m~3m – nfcCmdPkt[...]: Block data, length is 16m • Tag4 Command: nfcCmdPkt[0] = 0x81, nfcCmdPkt[1]~nfcCmdPkt[n]:data
<i>response</i>	Response as explained in FeliCA Lite-S User's Manual
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with `errorCode.getErrorString(string ident = "")`

21.21.1.19 RETURN_CODE IDTechSDK.IDT_VP8800.ctls_removeAllCAPK (string ident = " ")

Remove All Certificate Authority Public Key

Removes all the CAPK for CTLS

Parameters

<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.
--------------	--

Returns

RETURN_CODE: Values can be parsed with `errorCode.getErrorString(string ident = "")`

21.21.1.20 RETURN_CODE IDTechSDK.IDT_VP8800.ctls_removeApplicationData (byte[] *AID*, string *ident* = " ")

Remove Application Data by AID

Removes the Application Data for CTLS as specified by the AID name passed as a parameter

Parameters

<i>AID</i>	Name of ApplicationID Must be between 5 and 16 bytes
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with `errorCode.getErrorString(string ident = "")`

21.21.1.21 RETURN_CODE IDTechSDK.IDT_VP8800.ctls_removeCAPK (byte[] *capk*, string *ident* = " ")

Remove Certificate Authority Public Key

Removes the CAPK for CTLS as specified by the RID/Index

Parameters

<i>capk</i>	6 byte CAPK = 5 bytes RID + 1 byte INDEX
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with `errorCode.getErrorString(string ident = "")`

21.21.1.22 RETURN_CODE IDTechSDK.IDT_VP8800.ctls_removeConfigurationGroup (int *group*, string *ident* = " ")

Remove Configuration Group

Removes the Configuration as specified by the Group. Must not by group 0

Parameters

<i>group</i>	Configuration Group
--------------	---------------------

Return values

<i>RETURN_CODE</i>	Return codes listed as typedef enum in IDTCommon:RETURN_CODE. Values can be parsed with <code>IDT_VP8800::device_getResponseCodeString:(string ident = "")</code>
--------------------	---

21.21.1.23 RETURN_CODE IDTechSDK.IDT_VP8800.ctls_resetConfigurationGroup (int *group*, string *ident* = " ")**Reset Configuration Group**

This command allows resetting a dataset to its default configuration. If the file exists, it will be overwritten. If not, it will be created.

Parameters

<i>group</i>	Configuration Group
--------------	---------------------

Return values

<i>RETURN_CODE</i>	Return codes listed as typedef enum in IDTCommon:RETURN_CODE. Values can be parsed with IDT_VP8800::device_getResponseCodeString :(string ident = "")
--------------------	---

21.21.1.24 RETURN_CODE IDTechSDK.IDT_VP8800.ctls_retrieveAIDList (ref byte *response*[[], string *ident* = " ")**Retrieve AID list**

Returns all the AID names and their assigned groups installed on the terminal for CTLS.

Parameters

<i>response</i>	array of TLV data objects: DFEE2D (group name) followed by 9F06 (AID), and DFEE4F (Interface 01 = CTLS, 02 = CONTACT, string ident = "")
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with `errorCode.getErrorString(string ident = "")`

21.21.1.25 RETURN_CODE IDTechSDK.IDT_VP8800.ctls_retrieveApplicationData (byte[] *AID*, ref byte[] *tlv*, string *ident* = " ")**Retrieve Application Data by AID**

Retrieves the CTLS Application Data for System Default Group 0 as specified by the AID name passed as a parameter.

Parameters

<i>AID</i>	Name of ApplicationID. Must be between 5 and 16 bytes
<i>tlv</i>	The TLV elements of the requested AID
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with `errorCode.getErrorString()`

21.21.1.26 RETURN_CODE IDTechSDK.IDT_VP8800.ctls_retrieveCAPK (byte[] *capk*, ref byte[] *key*, string *ident* = " ")**Retrieve Certificate Authority Public Key**

Retrieves the CAPK for CTLS as specified by the RID/Index passed as a parameter.

Parameters

<i>capk</i>	6 bytes CAPK = 5 bytes RID + 1 byte Index
<i>key</i>	Response returned as a CAKey format: [1 byte Hash Algorithm][1 byte Encryption Algorithm][20 bytes HashValue][4 bytes Public Key Exponent][2 bytes Modulus Length][Variable bytes Modulus] Where: <ul style="list-style-type: none"> • Hash Algorithm: The only algorithm supported is SHA-1. The value is set to 0x01 • Encryption Algorithm: The encryption algorithm in which this key is used. Currently support only one type: RSA. The value is set to 0x01. • HashValue: Which is calculated using SHA-1 over the following fields: RID & Index & Modulus & Exponent • Public Key Exponent: Actually, the real length of the exponent is either one byte or 3 bytes. It can have two values: 3 (Format is 0x00 00 00 03), or 65537 (Format is 0x00 01 00 01, string ident = "") • Modulus Length: LenL LenH Indicated the length of the next field. • Modulus: This is the modulus field of the public key. Its length is specified in the field above.
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with `errorCode.getErrorString()`

21.21.1.27 RETURN_CODE IDTechSDK.IDT_VP8800.ctls_retrieveCAPKList (ref byte[] *keys*, string *ident* = " ")

Retrieve the Certificate Authority Public Key list

Returns all the CAPK RID and Index installed on the terminal for CTLS.

Parameters

<i>keys</i>	[key1][key2]...[keyn], each key 6 bytes where key = 5 bytes RID + 1 byte index
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with `errorCode.getErrorString()`

21.21.1.28 RETURN_CODE IDTechSDK.IDT_VP8800.ctls_retrieveTerminalData (ref byte[] *tlv*, string *ident* = " ")

Retrieve Terminal Data

Retrieves the Terminal Data for CTLS. This is configuration group 0 (Tag FFEE - > FFEE0100). The terminal data can also be retrieved by `ctls_getConfiguraitonGroup(0)`.

Parameters

<i>tlv</i>	Response returned as a TLV
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with `errorCode.getErrorString()`

21.21.1.29 RETURN_CODE IDTechSDK.IDT_VP8800.ctls_setApplicationData (byte[] *tlv*, string *ident* = " ")

Set Application Data by AID

Sets the Application Data as specified by TLV data

Parameters

<i>tlv</i>	Application data in TLV format The first tag of the TLV data must be the group number (DFEE2D). Group 0 = System, all other Groups = User The second tag of the TLV data must be the AID (9F06) If tag DFEE4F is included, it must have a value of 0x01 to be a CTLS AID
------------	--

Example valid TLV, for Group #2, AID a0000000045010: "dfee2d01029f0607a0000000045010dfee4b0101dfee2e0110dfee4c0101dfee4d0101dfee4e0101dfee4f0101"

Tags:

- DFEE2D : Group Number. Mandatory First Tag. 0 = System, 1-255 = User
- 9F06 : AID. Mandatory Second Tag
- DFEE4B : Partial AID Matching. 01 = allowed, 00 = Disabled. Mandatory for Visa
- DFEE4C : Application Flow, System: Never, User: Mandatory – 0x01 = MasterCard – 0x02 = AMEX – 0x03 = MasterCard w/Strip Application – 0x06 = Visa – 0x0D = Discover – 0x0E = JCB – 0x15 = Reserved – 0x16 = Reserved – 0x17 = Reserved
- DFEE4D = PPSE Disable, Optional
- DFEE2E = Max AID Length, Mandatory if DFEE4B included. Visa must be set to 16
- DFEE2F = Disable System Aid (no effect on User AID, string *ident* = "")
- DFEE4F = Interface Support. 01 = CTLS, 02 = Contact. If missing, defaults to CTLS

Parameters

<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.
--------------	--

Returns

RETURN_CODE: Values can be parsed with `errorCode.getErrorString()`

21.21.1.30 RETURN_CODE IDTechSDK.IDT_VP8800.ctls_setCAPK (byte[] *key*, string *ident* = " ")

Set Certificate Authority Public Key

Sets the CAPK for CTLS as specified by the CAKey structure

Parameters

<i>key</i>	CAKey format: [5 bytes RID][1 byte Index][1 byte Hash Algorithm][1 byte Encryption Algorithm][20 bytes HashValue][4 bytes Public Key Exponent][2 bytes Modulus Length][Variable bytes Modulus] Where: <ul style="list-style-type: none"> • Hash Algorithm: The only algorithm supported is SHA-1. The value is set to 0x01 • Encryption Algorithm: The encryption algorithm in which this key is used. Currently support only one type: RSA. The value is set to 0x01. • HashValue: Which is calculated using SHA-1 over the following fields: RID & Index & Modulus & Exponent • Public Key Exponent: Actually, the real length of the exponent is either one byte or 3 bytes. It can have two values: 3 (Format is 0x00 00 00 03), or 65537 (Format is 0x00 01 00 01, string ident = "") • Modulus Length: LenL LenH Indicated the length of the next field. • Modulus: This is the modulus field of the public key. Its length is specified in the field above.
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with `errorCode.getErrorString()`

21.21.1.31 RETURN_CODE IDTechSDK.IDT_VP8800.ctls_setConfigurationGroup (byte[] tlv, string ident = " ")

Set Configuration Group

Sets the Configuration Group for CTLS as specified by the TLV data

Parameters

<i>tlv</i>	Configuration Group Data in TLV format The first tag of the TLV data must be the group number (DFEE2D). A second tag must exist
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with `errorCode.getErrorString()`

21.21.1.32 RETURN_CODE IDTechSDK.IDT_VP8800.ctls_setDefaultConfiguration (string ident = " ")

Set Default Configuration Group

Resets the device to default CTLS configuration group settings

Parameters

<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.
--------------	--

Returns

RETURN_CODE: Values can be parsed with `errorCode.getErrorString()`

21.21.1.33 RETURN_CODE IDTechSDK.IDT_VP8800.ctls_setTerminalData (byte[] tlv, string ident = " ")

Set Terminal Data

Sets the Terminal Data for CTLS as specified by the TLV. The first TLV must be Configuration Group Number (Tag DFEE2D). The terminal global data is group 0, so the first TLV would be DFEE2D0100. Other groups can be defined using this method (1 or greater), and those can be retrieved with `emv_getConfigurationGroup(int group)`, and deleted with `emv_removeConfigurationGroup(int group)`. You cannot delete group 0.

Parameters

<i>tlv</i>	TerminalData configuration data
------------	---------------------------------

Return values

<i>RETURN_CODE</i>	Return codes listed as typedef enum in IDTCommon:RETURN_CODE. Values can be parsed with IDT_VP8800::device_getResponseCodeString:(string ident = " ")
--------------------	---

21.21.1.34 RETURN_CODE IDTechSDK.IDT_VP8800.ctls_startTransaction (double amount, double amtOther, int exponent, int type, int timeout, byte[] tags, bool forceOnline, bool isFastEMV = false, string ident = " ")

Start a CTLS Transaction Request

Authorizes the CTLS transaction for an CTLS card

The tags will be returned in the callback routine.

Parameters

<i>amount</i>	Transaction amount value (tag value 9F02, string ident = "")
<i>amtOther</i>	Other amount value, if any (tag value 9F03, string ident = "")
<i>exponent</i>	Number of characters after decimile point
<i>type</i>	Transaction type (tag value 9C).
<i>timeout</i>	Timeout value in seconds.
<i>tags</i>	Any other tags to be included in the request. Passed as TLV Data. Example, tag 9F02 with amount 0x000000000100 would be 0x9F0206000000000100 If tags 9F02 (amount), 9F03 (other amount), or 9C (transaction type) are included, they will take priority over these values supplied as individual parameters to this method.
<i>forceOnline</i>	TRUE = do not allow offline approval, FALSE = allow ICC to approve offline if terminal capable
<i>isFastEMV</i>	If TRUE, it will populate the IDTTransactionData.fastEMV with ASCII data similar to IDTech FastEMV KB output, after performing an auto-authenticate and auto-complete with ResultCode = Could Not Contact Host
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with `errorCode.getErrorString()`

NOTE ON APPLEPAY VAS: To enable ApplePay VAS, first a merchant record must be defined in one of the six available index positions (1-6) using `device_setMerchantRecord`, then container tag FFEE06 must be sent as part

of the additional tags parameter of `ctls_startTransaction`. Tag FFEE06 must contain tag 9F26 and 9F22, and can optionally contain tags 9F2B and DF01. Example FFEE06189F220201009F2604000000009F2B050100000000↵ DF010101 9F22 = two bytes = ApplePay Terminal Application Version Number. Hard defined as 0100 for now. (required, string ident = "") 9F26 = four bytes = ApplePay Terminal Capabilities Information (required, string ident = "")

- Byte 1 = RFU
- Byte 2 = Terminal Type
 - - Bit 8 = VAS Support (1=on, 0 = off, string ident = "")
 - - Bit 7 = Touch ID Required (1=on, 0 = off, string ident = "")
 - - Bit 6 = RFU
 - - Bit 5 = RFU
 - - Bit 1,2,3,4
 - - - 0 = Payment Terminal
 - - - 1 = Transit Terminal
 - - - 2 = Access Terminal
 - - - 3 = Wireless Handoff Terminal
 - - - 4 = App Handoff Terminal
 - - - 15 = Other Terminal
- Byte 3 = RFU
- Byte 4 = Terminal Mode
 - - 0 = ApplePay VAS OR ApplePay
 - - 1 = ApplePay VAS AND ApplePay
 - - 2 = ApplePay VAS ONLY
 - - 3 = ApplePay ONLY 9F2B = 5 bytes = ApplePay VAS Filter. Each byte filters for that specific merchant index (optional, string ident = "") DF01 = 1 byte = ApplePay VAS Protocol. (optional, string ident = "")
 - - Bit 1 : 1 = URL VAS, 0 = Full VAS
 - - Bit 2 : 1 = VAS Beeps, 0 = No VAS Beeps
 - - Bit 3 : 1 = Silent Comm Error, 2 = EMEA Comm Error
 - - Bit 4-8 : RFU

21.21.1.35 RETURN_CODE `IDTechSDK.IDT_VP8800.ctls_trySetTerminalData (byte[] tlv, ref byte[] rejectedTLV, ref byte[] convertedTLV, string ident = " ")`

Try to Set CTLS Terminal Data

Attempts to set the CTLS Terminal Data. Use this functions to attempt to set the terminal data to the device. This function will allow the cross-device tag compatible writing of IDTech Legacy/New tag definitions

Parameters

<i>tlv</i>	TerminalData TLV Data
<i>rejectedTLV</i>	Contains the tags (if any) that were rejected by the firmware
<i>convertedTLV</i>	Contains the tags (if any) that were converted and accepted by the firmware
<i>ident</i>	Optional identifier #80152520-001 IDTech Windows SDK Guide for DotNet

Return values

<i>RETURN_CODE</i>	Return codes listed as typedef enum in IDTCommon:RETURN_CODE. Values can be parsed with IDT_BTPay::device_getResponseCodeString:()
--------------------	--

21.21.1.36 `RETURN_CODE IDTechSDK.IDT_VP8800.device_activateTransaction (int timeout, byte[] tags, byte[] options, bool isFastEMV = false, string ident = " ")`

Start EMV Transaction Request

Authorizes the EMV transaction for an ICC card

The tags will be returned in the callback routine.

Parameters

<i>timeout</i>	Timeout value in seconds.
<i>tags</i>	The tags to be included in the request. Passed as TLV Data. Example, tag 9F02 with amount 0x000000000100 would be 0x9F0206000000000100
<i>options</i>	<p>Selects interface and other device options. Must be 4 bytes</p> <ul style="list-style-type: none"> • Byte 0 <ul style="list-style-type: none"> • - b7, b6, b5 : must be 0 • - b4, b3: 00 = normal, 01 = fallback from ICC, 10 = fallback from CTLS, 11 = INVALID SELECTION • - b2 : use MagStrip Interface • - b1 : use Contact interface • - b0 : Use CTLS interface • Byte 1 <ul style="list-style-type: none"> • - RFU : Must be set to value 0 • Byte 2 <ul style="list-style-type: none"> • - b8, b7, b6, : must be 0 • - b5 : Enable display of transaction type in default LCD UI Message • - b4 : Disable ViVOpay Default Contactless Logo • - b3 : Disable ViVOpay default Buzzer UI • - b2 : Disable ViVOpay default LED UI • - b1 : Disable ViVOpay default LCD UI Messages • Byte 3 <ul style="list-style-type: none"> • - b8, b7, b6, b5, b4, b3, b2 : must be 0 • - b1 : Use Event (non-blcoking) Mode

Note: To request tags to be included in default response, use tag DFEE1A, and specify tag list. Example four tags 9F02, 9F36, 95, 9F37 to be included in response = DFEE1A079F029F369f9F37

Parameters

<i>isFastEMV</i>	If TRUE, it will populate the IDTTransactionData.fastEMV with ASCII data similar to IDTech FastEMV KB output, after performing an auto-authenticate and auto-complete with ResultCode = Could Not Contact Host
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString

21.21.1.37 RETURN_CODE IDTechSDK.IDT_VP8800.device_buzzer (string *ident* = " ")

Buzzer Device

Buzzer the reader. If success can hear one beep. Otherwise, returns timeout.

Parameters

<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.
--------------	--

Returns

RETURN_CODE: Values can be parsed with errorCode.getErrorString(string ident = "")

21.21.1.38 RETURN_CODE IDTechSDK.IDT_VP8800.device_calibrateParameters (byte *delta*, string *ident* = " ")

Calibrate reference parameters

Parameters

<i>delta</i>	Delta value (0x02 standard default value, string ident = "")
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with errorCode.getErrorString(string ident = "")

21.21.1.39 RETURN_CODE IDTechSDK.IDT_VP8800.device_cancelTransaction (string *ident* = " ")

Cancel Transaction

Cancels the currently executing device transaction.

Parameters

<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.
--------------	--

Returns

RETURN_CODE: Values can be parsed with errorCode.getErrorString()

21.21.1.40 **RETURN_CODE** IDTechSDK.IDT_VP8800.device_controlIndicator (int *indicator*, bool *enable*, string *ident* = " ")

Control Indicators

Control the reader. If connected, returns success. Otherwise, returns timeout.

Parameters

<i>indicator</i>	description as follows: 00h: ICC LED 01h: Blue MSR 02h: Red MSR 03h: Green MSR
<i>enable</i>	TRUE = ON, FALSE = OFF
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with `errorCode.getErrorString(string ident = "")`

21.21.1.41 **RETURN_CODE** IDTechSDK.IDT_VP8800.device_controlLED (byte *indexLED*, byte *control*, string *ident* = " ")

Control LED

Controls the LED for the reader. This command will only operate in pass-through mode

Parameters

<i>indexLED</i>	For LED <ul style="list-style-type: none"> • 00: LED 0 (Power LED, leftmost LED) • 01: LED 1 • 02: LED 2 • 03: LED 3 • FF: All LED's
<i>control</i>	LED Status: <ul style="list-style-type: none"> • 00: LED Off • 01: LED On
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with `device_getResponseCodeString`

21.21.1.42 **RETURN_CODE** IDTechSDK.IDT_VP8800.device_controlUserInterface (byte[] *values*, string *ident* = " ")

Control User Interface

Controls the User Interface: Display, Beep, LED

Parameters

<i>values</i>	<p>Four bytes to control the user interface Byte[0] = LCD Message Messages 00-07 are normally controlled by the reader.</p> <ul style="list-style-type: none"> • 00h: Idle Message (Welcome, string ident = "") • 01h: Present card (Please Present Card, string ident = "") • 02h: Time Out or Transaction cancel (No Card, string ident = "") • 03h: Transaction between reader and card is in the middle (Processing...) • 04h: Transaction Pass (Thank You, string ident = "") • 05h: Transaction Fail (Fail, string ident = "") • 06h: Amount (Amount \$ 0.00 Tap Card, string ident = "") • 07h: Balance or Offline Available funds (Balance \$ 0.00) Messages 08-0B are controlled by the terminal • 08h: Insert or Swipe card (Use Chip & PIN, string ident = "") • 09h: Try Again(Tap Again, string ident = "") • 0Ah: Tells the customer to present only one card (Present 1 card only) • 0Bh: Tells the customer to wait for authentication/authorization (Wait) • FFh: indicates the command is setting the LED/Buzzer only. Byte[1] = Beep Indicator • 00h: No beep • 01h: Single beep • 02h: Double beep • 03h: Three short beeps • 04h: Four short beeps • 05h: One long beep of 200 ms • 06h: One long beep of 400 ms • 07h: One long beep of 600 ms • 08h: One long beep of 800 ms Byte[2] = LED Number • 00h: LED 0 (Power LED) 01h: LED 1 • 02h: LED 2 • 03h: LED 3 • FFh: All LEDs Byte[3] = LED Status • 00h: LED Off • 01h: LED On
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with `errorCode.getErrorString(string ident = "")`

21.21.1.43 RETURN_CODE IDTechSDK.IDT_VP8800.device_createDirectory (string *directoryName*, string *ident* = " ")**Create Directory**

This command adds a subdirectory to the indicated path.

Parameters

<i>directoryName</i>	Directory Name. The data for this command is a ASCII string with the complete path and directory name you want to create. You do not need to specify the root directory. Indicate subdirectories with a forward slash (/).
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with `errorCode.getErrorString(string ident = "")`

21.21.1.44 RETURN_CODE IDTechSDK.IDT_VP8800.device_deleteDirectory (string *filename*, string *ident* = " ")**Delete Directory**

This command deletes an empty directory.

Parameters

<i>filename</i>	Complete path and file name of the directory you want to delete. You do not need to specify the root directory. Indicate subdirectories with a forward slash (/).
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with `errorCode.getErrorString(string ident = "")`

21.21.1.45 RETURN_CODE IDTechSDK.IDT_VP8800.device_deleteFile (string *filename*, bool *isSD* = false, string *ident* = " ")**Delete File**

This command deletes a file or group of files.

Parameters

<i>filename</i>	Complete path and file name of the file you want to delete. You do not need to specify the root directory. Indicate subdirectories with a forward slash (/).
<i>isSD</i>	TRUE = Delete from SDCard, FALSE = Delete from Flash
<i>ip</i>	Optional: IP address to execute command on (for IP connected devices, string <i>ident</i> = "")
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with `errorCode.getErrorString(string ident = "")`

21.21.1.46 **RETURN_CODE** IDTechSDK.IDT_VP8800.device_enablePassThrough (bool *enablePassThrough*, string *ident* = " ")

Enable Pass Through

Enables Pass Through Mode for direct communication with L1 interface (power on icc, send apdu, etc).

Parameters

<i>enablePassThrough</i>	true = pass through ON, false = pass through OFF
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with `errorCode.getErrorString(string ident = "")`

21.21.1.47 **RETURN_CODE** IDTechSDK.IDT_VP8800.device_getConfigurationFromMemory (ref string *json*, string *ident* = " ")

Read Device Configuration From Memory Retrieves the ViVOconfig configuration data from memory after a successful device_readConfigurationToMemory

Parameters

<i>json</i>	Returns the .json file data that can be saved to disk
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with `errorCode.getErrorString()`. When no data is available, return code = **RETURN_CODE_NO_DATA_AVAILABLE**

21.21.1.48 **RETURN_CODE** IDTechSDK.IDT_VP8800.device_getConfigurationGroup (int *group*, ref byte[] *tlv*, string *ident* = " ")

Get Configuration Group

Retrieves the Configuration for the specified Group. Group 0 = terminal settings.

Parameters

<i>group</i>	Configuration Group
<i>tlv</i>	return data
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with `errorCode.getErrorString()`

21.21.1.49 **RETURN_CODE** IDTechSDK.IDT_VP8800.device_getDriveFreeSpace (ref int *free*, ref int *used*, string *ident* = " ")

Drive Free Space

This command returns the free and used disk space on the flash drive.

Parameters

<i>free</i>	Free bytes available on device
<i>used</i>	Used bytes on on device
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with `errorCode.getErrorString(string ident = "")`

21.21.1.50 RETURN_CODE IDTechSDK.IDT_VP8800.device_getFirmwareVersion (ref string *response*, string *ident* = " ")

Polls device for Firmware Version

Parameters

<i>response</i>	Response returned of Firmware Version
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with `errorCode.getErrorString(string ident = "")`

21.21.1.51 RETURN_CODE IDTechSDK.IDT_VP8800.device_getMerchantRecord (int *index*, ref byte[] *record*, string *ident* = " ")

Get Merchant Record Gets the merchant record for ApplePay VAS

Parameters

<i>index</i>	Merchant Record index, valid values 1-6
<i>record</i>	Data returned containing 99 bytes: Byte 0 = Merchand Index Byte 1 = Merchant Enabled (1 = enabled, string ident = "") Byte 2 - 33 = Merchant Protocol Hash-256 value Byte 34 = Length of Merchant URL Bytes 35 - 99 = URL
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with `errorCode.getErrorString(string ident = "")`

21.21.1.52 string IDTechSDK.IDT_VP8800.device_getResponseCodeString (RETURN_CODE *eCode*)

Get the description of response result.

Parameters

<i>eCode</i>	the response result.
--------------	----------------------

Return values

<i>the</i>	string for description of response result
------------	---

- case 0: "no error, beginning task";
- case 1: "no response from reader";
- case 2: "invalid response data";
- case 3: "time out for task or CMD";
- case 4: "wrong parameter";
- case 5: "SDK is doing MSR or ICC task";
- case 6: "SDK is doing PINPad task";
- case 7: "SDK is doing CTLS task";
- case 8: "SDK is doing EMV task";
- case 9: "SDK is doing Other task";
- case 10: "err response or data";
- case 11: "no reader attached";
- case 12: "mono audio is enabled";
- case 13: "did connection";
- case 14: "audio volume is too low";
- case 15: "task or CMD be canceled";
- case 16: "UF wrong string format";
- case 17: "UF file not found";
- case 18: "UF wrong file format";
- case 19: "Attempt to contact online host failed";
- case 20: "Attempt to perform RKI failed";
- case 0x300: "Key Type(TDES) of Session Key is not same as the related Master Key.";
- case 0x400: "Related Key was not loaded.";
- case 0x500: "Key Same.";
- case 0x501: "Key is all zero";
- case 0x502: "TR-31 format error";
- case 0x702: "PAN is Error Key.";
- case 0x705: "No Internal MSR PAN (or Internal MSR PAN is erased timeout)";
- case 0X0C01: "Incorrect Frame Tag";
- case 0X0C02: "Incorrect Frame Type";
- case 0X0C03: "Unknown Frame Type";
- case 0X0C04: "Unknown Command";

- case 0X0C05: "Unknown Sub-Command";
- case 0X0C06: "CRC Error";
- case 0X0C07: "Failed";
- case 0X0C08: "Timeout";
- case 0X0C0A: "Incorrect Parameter";
- case 0X0C0B: "Command Not Supported";
- case 0X0C0C: "Sub-Command Not Supported";
- case 0X0C0D: "Parameter Not Supported / Status Abort Command";
- case 0X0C0F: "Sub-Command Not Allowed";
- case 0X0D01: "Incorrect Header Tag";
- case 0X0D02: "Unknown Command";
- case 0X0D03: "Unknown Sub-Command";
- case 0X0D04: "CRC Error in Frame";
- case 0X0D05: "Incorrect Parameter";
- case 0X0D06: "Parameter Not Supported";
- case 0X0D07: "Mal-formatted Data";
- case 0X0D08: "Timeout";
- case 0X0D0A: "Failed / NACK";
- case 0X0D0B: "Command not Allowed";
- case 0X0D0C: "Sub-Command not Allowed";
- case 0X0D0D: "Buffer Overflow (Data Length too large for reader buffer)";
- case 0X0D0E: "User Interface Event";
- case 0X0D11: "Communication type not supported, VT-1, burst, etc.";
- case 0X0D12: "Secure interface is not functional or is in an intermediate state.";
- case 0X0D13: "Data field is not mod 8";
- case 0X0D14: "Pad 0x80 not found where expected";
- case 0X0D15: "Specified key type is invalid";
- case 0X0D1: "Could not retrieve key from the SAM(InitSecureComm)";
- case 0X0D17: "Hash code problem";
- case 0X0D18: "Could not store the key into the SAM(InstallKey)";
- case 0X0D19: "Frame is too large";
- case 0X0D1A: "Unit powered up in authentication state but POS must resend the InitSecureComm command";
- case 0X0D1B: "The EEPROM may not be initialized because SecCommInterface does not make sense";
- case 0X0D1C: "Problem encoding APDU";
- case 0X0D20: "Unsupported Index(ILM) SAM Transceiver error – problem communicating with the SAM(Key Mgr)";

- case 0X0D2: "Unexpected Sequence Counter in multiple frames for single bitmap(ILM) Length error in data returned from the SAM(Key Mgr)";
- case 0X0D22: "Improper bit map(ILM)";
- case 0X0D23: "Request Online Authorization";
- case 0X0D24: "ViVOCard3 raw data read successful";
- case 0X0D25: "Message index not available(ILM) ViVOcomm activate transaction card type(ViVOcomm)";
- case 0X0D26: "Version Information Mismatch(ILM)";
- case 0X0D27: "Not sending commands in correct index message index(ILM)";
- case 0X0D28: "Time out or next expected message not received(ILM)";
- case 0X0D29: "ILM languages not available for viewing(ILM)";
- case 0X0D2A: "Other language not supported(ILM)";
- case 0X0D41: "Unknown Error from SAM";
- case 0X0D42: "Invalid data detected by SAM";
- case 0X0D43: "Incomplete data detected by SAM";
- case 0X0D44: "Reserved";
- case 0X0D45: "Invalid key hash algorithm";
- case 0X0D46: "Invalid key encryption algorithm";
- case 0X0D47: "Invalid modulus length";
- case 0X0D48: "Invalid exponent";
- case 0X0D49: "Key already exists";
- case 0X0D4A: "No space for new RID";
- case 0X0D4B: "Key not found";
- case 0X0D4C: "Crypto not responding";
- case 0X0D4D: "Crypto communication error";
- case 0X0D4E: "Module-specific error for Key Manager";
- case 0X0D4F: "All key slots are full (maximum number of keys has been installed)";
- case 0X0D50: "Auto-Switch OK";
- case 0X0D51: "Auto-Switch failed";
- case 0X0D90: "Account DUKPT Key not exist";
- case 0X0D91: "Account DUKPT Key KSN exhausted";
- case 0x0D00: "This Key had been loaded.";
- case 0x0E00: "Base Time was loaded.";
- case 0x0F00: "Encryption Or Decryption Failed.";
- case 0x1000: "Battery Low Warning (It is High Priority Response while Battery is Low.)";
- case 0x1800: "Send "Cancel Command" after send "Get Encrypted PIN" & "Get Numeric" & "Get Amount"";
- case 0x1900: "Press "Cancel" key after send "Get Encrypted PIN" & "Get Numeric" & "Get Amount"";

- case 0x30FF: "Security Chip is not connect";
- case 0x3000: "Security Chip is deactivation & Device is In Removal Legally State.";
- case 0x3101: "Security Chip is activation & Device is In Removal Legally State.";
- case 0x5500: "No Admin DUKPT Key.";
- case 0x5501: "Admin DUKPT Key STOP.";
- case 0x5502: "Admin DUKPT Key KSN is Error.";
- case 0x5503: "Get Authentication Code1 Failed.";
- case 0x5504: "Validate Authentication Code Error.";
- case 0x5505: "Encrypt or Decrypt data failed.";
- case 0x5506: "Not Support the New Key Type.";
- case 0x5507: "New Key Index is Error.";
- case 0x5508: "Step Error.";
- case 0x5509: "KSN Error";
- case 0x550A: "MAC Error.";
- case 0x550B: "Key Usage Error.";
- case 0x550C: "Mode Of Use Error.";
- case 0x550F: "Other Error.";
- case 0x6000: "Save or Config Failed / Or Read Config Error.";
- case 0x6200: "No Serial Number.";
- case 0x6900: "Invalid Command - Protocol is right, but task ID is invalid.";
- case 0x6A01: "Unsupported Command – Protocol and task ID are right, but command is invalid – In this State";
- case 0x6A00: "Unsupported Command - Protocol and task ID are right, but command is invalid.";
- case 0x6B00: "Unknown parameter in command - Protocol task ID and command are right, but parameter is invalid.";
- case 0x6C00: "Unknown parameter in command – Protocol task ID and command are right, but length is out of the requirement.";
- case 0x7200: "Device is suspend (MKSK suspend or press password suspend).";
- case 0x7300: "PIN DUKPT is STOP (21 bit 1).";
- case 0x7400: "Device is Busy.";
- case 0xE100: "Can not enter sleep mode";
- case 0xE200: "File has existed";
- case 0xE300: "File has not existed";
- case 0xE313: "IO line low -- Card error after session start";
- case 0xE400: "Open File Error";
- case 0xE500: "SmartCard Error";
- case 0xE600: "Get MSR Card data is error";

- case 0xE700: "Command time out";
- case 0xE800: "File read or write is error";
- case 0xE900: "Active 1850 error!";
- case 0xEA00: "Load bootloader error";
- case 0xEF00: "Protocol Error- STX or ETX or check error.";
- case 0xEB00: "Picture is not exist";
- case 0x2C02: "No Microprocessor ICC seated";
- case 0x2C06: "no card seated to request ATR";
- case 0x2D01: "Card Not Supported,";
- case 0x2D03: "Card Not Supported, wants CRC";
- case 0x690D: "Command not supported on reader without ICC support";
- case 0x8100: "ICC error time out on power-up";
- case 0x8200: "invalid TS character received - Wrong operation step";
- case 0x8300: "Decode MSR Error";
- case 0x8400: "TriMagII no Response";
- case 0x8500: "No Swipe MSR Card";
- case 0x8510: "No Financial Card";
- case 0x8600: "Unsupported F, D, or combination of F and D";
- case 0x8700: "protocol not supported EMV TD1 out of range";
- case 0x8800: "power not at proper level";
- case 0x8900: "ATR length too long";
- case 0x8B01: "EMV invalid TA1 byte value";
- case 0x8B02: "EMV TB1 required";
- case 0x8B03: "EMV Unsupported TB1 only 00 allowed";
- case 0x8B04: "EMV Card Error, invalid BWI or CWI";
- case 0x8B06: "EMV TB2 not allowed in ATR";
- case 0x8B07: "EMV TC2 out of range";
- case 0x8B08: "EMV TC2 out of range";
- case 0x8B09: "per EMV96 TA3 must be > 0xF";
- case 0x8B10: "ICC error on power-up";
- case 0x8B11: "EMV T=1 then TB3 required";
- case 0x8B12: "Card Error, invalid BWI or CWI";
- case 0x8B13: "Card Error, invalid BWI or CWI";
- case 0x8B17: "EMV TC1/TB3 conflict-";
- case 0x8B20: "EMV TD2 out of range must be T=1";
- case 0x8C00: "TCK error";

- case 0xA304: "connector has no voltage setting";
- case 0xA305: "ICC error on power-up invalid (SBLK(IFSD) exchange";
- case 0xE301: "ICC error after session start";
- case 0xFF00: "Request to go online";
- case 0xFF01: "EMV: Accept the offline transaction";
- case 0xFF02: "EMV: Decline the offline transaction";
- case 0xFF03: "EMV: Accept the online transaction";
- case 0xFF04: "EMV: Decline the online transaction";
- case 0xFF05: "EMV: Application may fallback to magstripe technology";
- case 0xFF06: "EMV: ICC detected tah the conditions of use are not satisfied";
- case 0xFF07: "EMV: ICC didn't accept transaction";
- case 0xFF08: "EMV: Transaction was cancelled";
- case 0xFF09: "EMV: Application was not selected by kernel or ICC format error or ICC missing data error";
- case 0xFF0A: "EMV: Transaction is terminated";
- case 0xFF0B: "EMV: Other EMV Error";
- case 0xFFFF: "NO RESPONSE";
- case 0xF002: "ICC communication timeout";
- case 0xF003: "ICC communication Error";
- case 0xF00F: "ICC Card Seated and Highest Priority, disable MSR work request";
- case 0xF200: "AID List / Application Data is not exist";
- case 0xF201: "Terminal Data is not exist";
- case 0xF202: "TLV format is error";
- case 0xF203: "AID List is full";
- case 0xF204: "Any CA Key is not exist";
- case 0xF205: "CA Key RID is not exist";
- case 0xF206: "CA Key Index it not exist";
- case 0xF207: "CA Key is full";
- case 0xF208: "CA Key Hash Value is Error";
- case 0xF209: "Transaction format error";
- case 0xF20A: "The command will not be processing";
- case 0xF20B: "CRL is not exist";
- case 0xF20C: "CRL number exceed max number";
- case 0xF20D: "Amount,Other Amount,Trasaction Type are missing";
- case 0xF20E: "The Identification of algorithm is mistake";
- case 0xF20F: "No Financial Card";
- case 0xF210: "In Encrypt Result state, TLV total Length is greater than Max Length";

- case 0x1001: "INVALID ARG";
- case 0x1002: "FILE_OPEN_FAILED";
- case 0x1003: "FILE OPERATION_FAILED";
- case 0x2001: "MEMORY_NOT_ENOUGH";
- case 0x3002: "SMARTCARD_FAIL";
- case 0x3003: "SMARTCARD_INIT_FAILED";
- case 0x3004: "FALLBACK_SITUATION";
- case 0x3005: "SMARTCARD_ABSENT";
- case 0x3006: "SMARTCARD_TIMEOUT";
- case 0x5001: "EMV_PARSING_TAGS_FAILED";
- case 0x5002: "EMV_DUPLICATE_CARD_DATA_ELEMENT";
- case 0x5003: "EMV_DATA_FORMAT_INCORRECT";
- case 0x5004: "EMV_NO_TERM_APP";
- case 0x5005: "EMV_NO_MATCHING_APP";
- case 0x5006: "EMV_MISSING_MANDATORY_OBJECT";
- case 0x5007: "EMV_APP_SELECTION_RETRY";
- case 0x5008: "EMV_GET_AMOUNT_ERROR";
- case 0x5009: "EMV_CARD_REJECTED";
- case 0x5010: "EMV_AIP_NOT_RECEIVED";
- case 0x5011: "EMV_AFL_NOT_RECEIVED";
- case 0x5012: "EMV_AFL_LEN_OUT_OF_RANGE";
- case 0x5013: "EMV_SFI_OUT_OF_RANGE";
- case 0x5014: "EMV_AFL_INCORRECT";
- case 0x5015: "EMV_EXP_DATE_INCORRECT";
- case 0x5016: "EMV_EFF_DATE_INCORRECT";
- case 0x5017: "EMV_ISS_COD_TBL_OUT_OF_RANGE";
- case 0x5018: "EMV_CRYPTOGAM_TYPE_INCORRECT";
- case 0x5019: "EMV_PSE_NOT_SUPPORTED_BY_CARD";
- case 0x5020: "EMV_USER_SELECTED_LANGUAGE";
- case 0x5021: "EMV_SERVICE_NOT_ALLOWED";
- case 0x5022: "EMV_NO_TAG_FOUND";
- case 0x5023: "EMV_CARD_BLOCKED";
- case 0x5024: "EMV_LEN_INCORRECT";
- case 0x5025: "CARD_COM_ERROR";
- case 0x5026: "EMV_TSC_NOT_INCREASED";
- case 0x5027: "EMV_HASH_INCORRECT";

- case 0x5028: "EMV_NO_ARC";
- case 0x5029: "EMV_INVALID_ARC";
- case 0x5030: "EMV_NO_ONLINE_COMM";
- case 0x5031: "TRAN_TYPE_INCORRECT";
- case 0x5032: "EMV_APP_NO_SUPPORT";
- case 0x5033: "EMV_APP_NOT_SELECT";
- case 0x5034: "EMV_LANG_NOT_SELECT";
- case 0x5035: "EMV_NO_TERM_DATA";
- case 0x6001: "CVM_TYPE_UNKNOWN";
- case 0x6002: "CVM_AIP_NOT_SUPPORTED";
- case 0x6003: "CVM_TAG_8E_MISSING";
- case 0x6004: "CVM_TAG_8E_FORMAT_ERROR";
- case 0x6005: "CVM_CODE_IS_NOT_SUPPORTED";
- case 0x6006: "CVM_COND_CODE_IS_NOT_SUPPORTED";
- case 0x6007: "NO_MORE_CVM";
- case 0x6008: "PIN_BYPASSED_BEFORE";
- case 0x7001: "PK_BUFFER_SIZE_TOO_BIG";
- case 0x7002: "PK_FILE_WRITE_ERROR";
- case 0x7003: "PK_HASH_ERROR";
- case 0x8001: "NO_CARD_HOLDER_CONFIRMATION";
- case 0x8002: "GET_ONLINE_PIN";
- case 0xD000: "Data not exist";
- case 0xD001: "Data access error";
- case 0xD100: "RID not exist";
- case 0xD101: "RID existed";
- case 0xD102: "Index not exist";
- case 0xD200: "Maximum exceeded";
- case 0xD201: "Hash error";
- case 0xD205: "System Busy";
- case 0x0E01: "Unable to go online";
- case 0x0E02: "Technical Issue";
- case 0x0E03: "Declined";
- case 0x0E04: "Issuer Referral transaction";
- case 0x0F01: "Decline the online transaction";
- case 0x0F02: "Request to go online";
- case 0x0F03: "Transaction is terminated";

- case 0x0F05: "Application was not selected by kernel or ICC format error or ICC missing data error";
- case 0x0F07: "ICC didn't accept transaction";
- case 0x0F0A: "Application may fallback to magstripe technology";
- case 0x0F0C: "Transaction was cancelled";
- case 0x0F0D: "Timeout";
- case 0x0F0F: "Other EMV Error";
- case 0x0F10: "Accept the offline transaction";
- case 0x0F11: "Decline the offline transaction";
- case 0x0F21: "ICC detected tah the conditions of use are not satisfied";
- case 0x0F22: "No app were found on card matching terminal configuration";
- case 0x0F23: "Terminal file does not exist";
- case 0x0F24: "CAPK file does not exist";
- case 0x0F25: "CRL Entry does not exist";
- case 0x0FFE: "code when blocking is disabled";
- case 0x0FFF: "code when command is not applicable on the selected device";
- case 0xF005: "ICC Encrypted C-APDU Data Structure Length Error Or Format Error.";
- case 0xBBE0: "CM100 Success";
- case 0xBBE1: "CM100 Parameter Error";
- case 0xBBE2: "CM100 Low Output Buffer";
- case 0xBBE3: "CM100 Card Not Found";
- case 0xBBE4: "CM100 Collision Card Exists";
- case 0xBBE5: "CM100 Too Many Cards Exist";
- case 0xBBE6: "CM100 Saved Data Does Not Exist";
- case 0xBBE8: "CM100 No Data Available";
- case 0xBBE9: "CM100 Invalid CID Returned";
- case 0xBBEA: "CM100 Invalid Card Exists";
- case 0xBBEC: "CM100 Command Unsupported";
- case 0xBBED: "CM100 Error In Command Process";
- case 0xBBEE: "CM100 Invalid Command";
- case 0X9031: "Unknown command";
- case 0X9032: "Wrong parameter (such as the length of the command is incorrect)";
- case 0X9038: "Wait (the command couldn't be finished in BWT)";
- case 0X9039: "Busy (a previously command has not been finished)";
- case 0X903A: "Number of retries over limit";
- case 0X9040: "Invalid Manufacturing system data";
- case 0X9041: "Not authenticated";

- case 0X9042: "Invalid Master DUKPT Key";
- case 0X9043: "Invalid MAC Key";
- case 0X9044: "Reserved for future use";
- case 0X9045: "Reserved for future use";
- case 0X9046: "Invalid DATA DUKPT Key";
- case 0X9047: "Invalid PIN Pairing DUKPT Key";
- case 0X9048: "Invalid DATA Pairing DUKPT Key";
- case 0X9049: "No nonce generated";
- case 0X9949: "No GUID available. Perform getVersion first.";
- case 0X9950: "MAC Calculation unsuccessful. Check BDK value.";
- case 0X904A: "Not ready";
- case 0X904B: "Not MAC data";
- case 0X9050: "Invalid Certificate";
- case 0X9051: "Duplicate key detected";
- case 0X9052: "AT checks failed";
- case 0X9053: "TR34 checks failed";
- case 0X9054: "TR31 checks failed";
- case 0X9055: "MAC checks failed";
- case 0X9056: "Firmware download failed";
- case 0X9060: "Log is full";
- case 0X9061: "Removal sensor unengaged";
- case 0X9062: "Any hardware problems";
- case 0X9070: "ICC communication timeout";
- case 0X9071: "ICC data error (such check sum error)";
- case 0X9072: "Smart Card not powered up";

21.21.1.53 `RETURN_CODE IDTechSDK.IDT_VP8800.device_getTransactionResults (ref IDTTransactionData results, string ident = " ")`

Get Transaction Results Gets the transaction results when the reader is functioning in "Auto Poll" mode

Parameters

<i>results</i>	The transaction results
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with `errorCode.getErrorString()`. When no data is available, return code = `RETURN_CODE_NO_DATA_AVAILABLE`

21.21.1.54 **RETURN_CODE** IDTechSDK.IDT_VP8800.device_listDirectory (string *directoryName*, bool *recursive*, bool *onSD*, ref string *directory*, string *ident* = " ")

List Directory

This command retrieves a directory listing of user accessible files from the reader.

Parameters

<i>directoryName</i>	Directory Name. If null, root directory is listed
<i>recursive</i>	Included sub-directories
<i>onSD</i>	TRUE = use flash storage The returned directory information
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with `errorCode.getErrorString(string ident = "")`

21.21.1.55 **RETURN_CODE** IDTechSDK.IDT_VP8800.device_pingDevice (string *ident* = " ")

Ping Device

Pings the reader. If connected, returns success. Otherwise, returns timeout.

Parameters

<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.
--------------	--

Returns

RETURN_CODE: Values can be parsed with `errorCode.getErrorString(string ident = "")`

21.21.1.56 **RETURN_CODE** IDTechSDK.IDT_VP8800.device_readConfigurationToMemory (string *memo*, Configs.InstallRules *rules*, List< Configs.DeviceCommand > *cmds*, string *ident* = " ", bool *isForeground* = false)

Read Device Configuration To Memory Executes a ViVOconfig read of device settings and stores the results in memory

Once a ViVOconfig read starts, all messages will be returned to the callback as `DeviceState.ViVOconfig`. The ViVOconfig operation is complete when the return code from a `DeviceState.ViVOconfig` message is one of the following values:

- **RETURN_CODE_SDK_VIVOCONFIG_READ_CONFIG_SUCCESS** = Read process completed successfully. Execute `device_getConfigurationFromMemory` to retrieve the configuration data
- **RETURN_CODE_SDK_VIVOCONFIG_READ_CONFIG_FAILED** = Read process FAILED.

Parameters

<i>memo</i>	Optional: populates the memo field of the captured .json file
<i>rules</i>	Optional: populates the .json file with available install rules
<i>cmds</i>	Optional: populates the .json file with commands to execute
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.
<i>isForeground</i>	If TRUE, will block program until update complete. Otherwise, FALSE performs update on background.

Returns

RETURN_CODE: Values can be parsed with `errorCode.getErrorString()`. When no data is available, return code = RETURN_CODE_NO_DATA_AVAILABLE

21.21.1.57 RETURN_CODE IDTechSDK.IDT_VP8800.device_RemoteKeyInjection (RKI_KEY_TYPE *type*, string *keyName*, string *ident* = " ", bool *performOnForeground* = false)

Remote Key Injection Performs a remote key injection for the device

Parameters

<i>type</i>	Remote Key Injection Type
<i>keyName</i>	Name of key (optional)
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.
<i>performOnForeground</i>	TRUE = block during process (default is false)

Returns

RETURN_CODE: Values can be parsed with `errorCode.getErrorString()`. When no data is available, return code = RETURN_CODE_NO_DATA_AVAILABLE

21.21.1.58 RETURN_CODE IDTechSDK.IDT_VP8800.device_removeConfigurationGroup (int *group*, string *ident* = " ")

Remove Configuration Group

Removes the Configuration as specified by the Group. Must not by group 0

Parameters

<i>group</i>	Configuration Group
--------------	---------------------

Return values

RETURN_CODE	Return codes listed as typedef enum in IDTCommon:RETURN_CODE. Values can be parsed with IDT_VP8800::device_getResponseCodeString :(string ident = "")
-------------	---

21.21.1.59 RETURN_CODE IDTechSDK.IDT_VP8800.device_resetConfigurationGroup (int *group*, string *ident* = " ")

Reset Configuration Group

This command allows resetting a dataset to its default configuration. If the file exists, it will be overwritten. If not, it will be created.

Parameters

<i>group</i>	Configuration Group
--------------	---------------------

Return values

RETURN_CODE	Return codes listed as typedef enum in IDTCommon:RETURN_CODE. Values can be parsed with IDT_VP8800::device_getResponseCodeString :(string ident = "")
-------------	---

21.21.1.60 **RETURN_CODE** IDTechSDK.IDT_VP8800.device_retrieveAIDList (ref byte *response*[[], string *ident* = " ")

Retrieve AID list

Returns all the AID names and their assigned groups installed on the terminal for CTLS/CONTACT.

Parameters

<i>response</i>	array of 2-tag TLV data objects: DFEE2D (group name) followed by 9F06 (AID, string ident = "")
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with `errorCode.getErrorString(string ident = "")`

21.21.1.61 **RETURN_CODE** IDTechSDK.IDT_VP8800.device_sendConfiguration (string *filename*, VC_OPERATION_TYPE *type*, bool *matchFW*, string *ident* = " ", bool *isForeground* = false)

Send Configuration Executes a ViVOconfig update or verify to a device

Once a ViVOconfig write or verify starts, all messages will be returned to the callback as `DeviceState.ViVOconfig`. The ViVOconfig operation is complete when the return code from a `DeviceState.ViVOconfig` message is one of the following values:

- **RETURN_CODE_SDK_VIVOCONFIG_VERIFY_SUCCESS** = Verification process completed successfully. No differences found.
- **RETURN_CODE_SDK_VIVOCONFIG_VERIFY_SUCCESS_WITH_WARNING** = Verification process completed with warnings.
- **RETURN_CODE_SDK_VIVOCONFIG_VERIFY_FAILED** = Verification process FAILED
- **RETURN_CODE_SDK_VIVOCONFIG_WRITE_SUCCESS** = Write process completed successfully.
- **RETURN_CODE_SDK_VIVOCONFIG_WRITE_SUCCESS_WITH_WARNING** = Write process completed with warnings
- **RETURN_CODE_SDK_VIVOCONFIG_WRITE_FAILED** = Write process FAILED

Parameters

<i>filename</i>	The .json configuration file to update the device with NOTE: You can also use this parameter to pass the path of where the .json file is located at on local storage instead
<i>filename</i>	The name of the .json configuration file
<i>type</i>	VC_OPERATION_TYPE

- **VC_OPERATION_TYPE_WRITE** = Write File To Device, Hash must validate
- **VC_OPERATION_TYPE_VERIFY** = Verify Device With File
- **VC_OPERATION_TYPE_WRITE_IGNORE_HASH** = Write File To Device, Ignore Hash
- **VC_OPERATION_TYPE_WRITE_FIX_HASH** = Write File To Device, Fix Hash If Necessary

Parameters

<i>matchFW</i>	TRUE = Device FW must match file FW, FALSE = Don't validate device FW against file
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Parameters

<i>isForeground</i>	If TRUE, will block program until update complete. Otherwise, FALSE performs update on background.
---------------------	--

Returns

RETURN_CODE: Values can be parsed with `errorCode.getErrorString()`. When no data is available, return code = RETURN_CODE_NO_DATA_AVAILABLE

21.21.1.62 RETURN_CODE `IDTechSDK.IDT_VP8800.device_sendConfigurationFromZip (byte[] zip, string filename, VC_OPERATION_TYPE type, bool matchFW, string ident = " ", bool isForeground = false)`

Send Configuration From Zip Executes a ViVOconfig update or verify to a device using files from a .zip archive

Once a ViVOconfig write or update starts, all messages will be returned to the callback as `DeviceState.ViVOconfig`. The ViVOconfig operation is complete when the return code from a `DeviceState.ViVOconfig` message is one of the following values:

- RETURN_CODE_SDK_VIVOCONFIG_VERIFY_SUCCESS = Verification process completed successfully. No differences found.
- RETURN_CODE_SDK_VIVOCONFIG_VERIFY_SUCCESS_WITH_WARNING = Verification process completed with warnings.
- RETURN_CODE_SDK_VIVOCONFIG_VERIFY_FAILED = Verification process FAILED
- RETURN_CODE_SDK_VIVOCONFIG_WRITE_SUCCESS = Write process completed successfully.
- RETURN_CODE_SDK_VIVOCONFIG_WRITE_SUCCESS_WITH_WARNING = Write process completed with warnings
- RETURN_CODE_SDK_VIVOCONFIG_WRITE_FAILED = Write process FAILED

Parameters

<i>zip</i>	.Zip archive containing the .json configuration file and also any associated image assets NOTE: You can also use this parameter to pass the path of where the .zip file is located at on local storage instead
<i>filename</i>	The name of the .json configuration file
<i>type</i>	VC_OPERATION_TYPE

- VC_OPERATION_TYPE_WRITE = Write File To Device, Hash must validate
- VC_OPERATION_TYPE_VERIFY = Verify Device With File
- VC_OPERATION_TYPE_WRITE_IGNORE_HASH = Write File To Device, Ignore Hash
- VC_OPERATION_TYPE_WRITE_FIX_HASH = Write File To Device, Fix Hash If Necessary

Parameters

<i>matchFW</i>	TRUE = Device FW must match file FW, FALSE = Don't validate device FW against file
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.
<i>isForeground</i>	If TRUE, will block program until update complete. Otherwise, FALSE performs update on background.

Returns

RETURN_CODE: Values can be parsed with `errorCode.getErrorString()`. When no data is available, return code = RETURN_CODE_NO_DATA_AVAILABLE

21.21.1.63 RETURN_CODE IDTechSDK.IDT_VP8800.device_sendDataCommand (string *cmd*, bool *calcLRC*, ref byte[] *response*, string *ident* = " ")

Send a data command to the device

Sends a command to the device.

Parameters

<i>cmd</i>	String representation of command to execute
<i>calcLRC</i>	If TRUE, this will wrap command with start/length/lrc/sum/end: '{STX}{Len_Low}{Len_High} data {CheckLRC} {CheckSUM} {ETX}'
<i>response</i>	Response data
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with `device_getResponseCodeString`

21.21.1.64 RETURN_CODE IDTechSDK.IDT_VP8800.device_sendDataCommand_ext (string *cmd*, bool *calcLRC*, ref byte[] *response*, int *timeout*, bool *noResponse*, string *ident* = " ")

Send a data command to the device - extended

Sends a command to the device.

Parameters

<i>cmd</i>	String representation of command to execute
<i>calcLRC</i>	If TRUE, this will wrap command as NGA with start/length/lrc/sum/end: '{STX}{Len_Low}{Len_High} data {CheckLRC} {CheckSUM} {ETX}'
<i>response</i>	Response data
<i>timeout</i>	Timeout value waiting for response, in milliseconds (1000 = 1 second, string ident = "")
<i>noResponse</i>	if TRUE, this will not wait for a response and immediately return SUCCESS
<i>calcITP</i>	If TRUE, this will wrap command as ITP with start/end/lrc: '{STX} data {ETX}{CheckLRC}'
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with `device_getResponseCodeString`

21.21.1.65 RETURN_CODE IDTechSDK.IDT_VP8800.device_sendPAE (string *command*, ref string *response*, int *timeout*, string *ident* = " ")

Send Payment Application Engine Command

Executes a PAE command

Parameters

<i>command</i>	ASCII command string, should start with "*PAE"
<i>response</i>	command response
<i>timeout</i>	timeout waiting for PAE response
<i>ip</i>	Optional IP address when connected via TCP/IP
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with `errorCode.getErrorString(string ident = "")`

21.21.1.66 RETURN_CODE IDTechSDK.IDT_VP8800.device_sendVivoCommandP2 (byte *command*, byte *subCommand*, byte[] *data*, ref byte[] *response*, string *ident* = " ")

Send Vivo Command Protocol 2

Sends a protocol 2 command to Vivo readers (IDG/NEO, string *ident* = "")

Parameters

<i>command</i>	Command
<i>subCommand</i>	Sub-Command
<i>data</i>	Data. May be null
<i>response</i>	Response
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with `errorCode.getErrorString(string ident = "")`

21.21.1.67 RETURN_CODE IDTechSDK.IDT_VP8800.device_sendVivoCommandP2_ext (byte *command*, byte *subCommand*, byte[] *data*, ref byte[] *response*, int *timeout*, bool *noResponse*, string *ident* = " ")

Send Vivo Command Protocol 2 Extended

Sends a protocol 2 command to Vivo readers (IDG/NEO, string *ident* = "")

Parameters

<i>command</i>	Command
<i>subCommand</i>	Sub-Command
<i>data</i>	Data. May be null
<i>response</i>	Response
<i>timeout</i>	Timeout, in milliseconds (3000 = 3 seconds, string <i>ident</i> = "")
<i>noResponse</i>	TRUE = don't wait for response, FALSE = wait for response defined by timeout
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with `errorCode.getErrorString(string ident = "")`

21.21.1.68 **RETURN_CODE** IDTechSDK.IDT_VP8800.device_sendVivoCommandP3 (*byte command*, *byte subCommand*, *byte[] data*, *ref byte[] response*, *string ident = " "*)

Send Vivo Command Protocol 3

Sends a protocol 3 command to Vivo readers (IDG/NEO, string ident = "")

Parameters

<i>command</i>	Command
<i>subCommand</i>	Sub-Command
<i>data</i>	Data. May be null
<i>response</i>	Response
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with `errorCode.getErrorString(string ident = "")`

21.21.1.69 **RETURN_CODE** IDTechSDK.IDT_VP8800.device_sendVivoCommandP3_ext (*byte command*, *byte subCommand*, *byte[] data*, *ref byte[] response*, *int timeout*, *bool noResponse*, *string ident = " "*)

Send Vivo Command Protocol 3 Extended

Sends a protocol 3 command to Vivo readers (IDG/NEO, string ident = "")

Parameters

<i>command</i>	Command
<i>subCommand</i>	Sub-Command
<i>data</i>	Data. May be null
<i>response</i>	Response
<i>timeout</i>	Timeout, in milliseconds (3000 = 3 seconds, string ident = "")
<i>noResponse</i>	TRUE = don't wait for response, FALSE = wait for response defined by timeout
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with `errorCode.getErrorString(string ident = "")`

21.21.1.70 **RETURN_CODE** IDTechSDK.IDT_VP8800.device_setBurstMode (*byte mode*, *string ident = " "*)

Send Burst Mode

Sets the burst mode for the device.

Parameters

<i>mode</i>	0 = OFF, 1 = Always On, 2 = Auto Exit
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with `errorCode.getErrorString(string ident = "")`

21.21.1.71 RETURN_CODE IDTechSDK.IDT_VP8800.device_setBuzzerLED (byte *buzzer*, byte *led*, bool *ledON*, string *ident* = " ")

Set Buzzer/LED

Sets the readers buzzer and LED's.

Parameters

<i>buzzer</i>	<ul style="list-style-type: none"> • 00h: No beep • 01h: Single beep • 02h: Double beep • 03h: Three short beeps • 04h: Four short beeps • 05h: One long beep of 200 ms • 06h: One long beep of 400 ms • 07h: One long beep of 600 ms • 08h: One long beep of 800 ms
<i>led</i>	<ul style="list-style-type: none"> • 00h: LED 0 (Leftmost LED, string ident = "") • 01h: LED 1 • 02h: LED 2 • 03h: LED 3 • FFh: All LEDs
<i>ledON</i>	TRUE = ON, FALSE = OFF
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with `errorCode.getErrorString(string ident = "")`

21.21.1.72 RETURN_CODE IDTechSDK.IDT_VP8800.device_setMerchantRecord (int *index*, bool *enabled*, string *merchantID*, string *merchantURL*, string *ident* = " ")

Set Merchant Record Sets the merchant record for ApplePay VAS

Parameters

<i>index</i>	Merchant Record index, valid values 1-6
<i>enabled</i>	Merchant Enabled/Valid flag

Parameters

<i>merchantID</i>	Merchant unique identifier registered with Apple. Example com.idtechproducts.applePay
<i>merchantURL</i>	Merchant URL, when applicable
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with `errorCode.getErrorString(string ident = "")`

21.21.1.73 RETURN_CODE IDTechSDK.IDT_VP8800.device_setPollMode (byte *mode*, string *ident* = " ")

Send Poll Mode

Sets the poll mode for the device. Auto Poll keeps reader active, Poll On Demand only polls when requested by terminal

Parameters

<i>mode</i>	0 = Auto Poll, 1 = Poll On Demand
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with `errorCode.getErrorString(string ident = "")`

21.21.1.74 RETURN_CODE IDTechSDK.IDT_VP8800.device_startRKI (bool *isTest*, string *ident* = " ")

Start Remote Key Injection

Starts a remote key injection request with IDTech RKI servers.

Parameters

<i>isTest</i>	TRUE = Demo Device, FALSE = Production Device
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with `device_getResponseCodeString`

21.21.1.75 RETURN_CODE IDTechSDK.IDT_VP8800.device_StartRKI (int *type*, string *ident* = " ")

Start Remote Key Injection

Starts a remote key injection request with IDTech RKI servers. Set/Get RKI url with `IDT_Device.RKI_URL`.

Parameters

<i>type</i>	0 = Symmetric RKI Demo Unit 1 = Symmetric RKI Production Unit 2 = PKI-RKI Demo Unit 3 = PKI-RKI Production Unit
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString

21.21.1.76 RETURN_CODE IDTechSDK.IDT_VP8800.device_startTransaction (double *amount*, double *amtOther*, int *exponent*, int *type*, int *timeout*, byte[] *tags*, byte[] *options*, bool *isFastEMV* = false, int *transMode* = 0, string *ident* = " ")

Start EMV Transaction Request

Authorizes the EMV transaction for an ICC card

The tags will be returned in the callback routine.

Parameters

<i>amount</i>	Transaction amount value (tag value 9F02, string ident = "")
<i>amtOther</i>	Other amount value, if any (tag value 9F03, string ident = "")
<i>exponent</i>	Number of characters after decimile point
<i>type</i>	Transaction type (tag value 9C).
<i>timeout</i>	Timeout value in seconds.
<i>tags</i>	Any other tags to be included in the request. Passed as TLV Data. Example, tag 9F02 with amount 0x0000000000100 would be 0x9F02060000000000100 If tags 9F02 (amount), 9F03 (other amount), or 9C (transaction type) are included, they will take priority over these values supplied as individual parameters to this method.
<i>options</i>	<p>Selects interface and other device options. Must be 4 bytes</p> <ul style="list-style-type: none"> • Byte 0 <ul style="list-style-type: none"> • - b8, b7, b6 : must be 0 • - b5, b4: 00 = normal, 01 = fallback from ICC, 10 = fallback from CTLS, 11 = INVALID SELECTION • - b3 : use MagStrip Interface • - b2 : use Contact interface • - b1 : Use CTLS interface • Byte 1 <ul style="list-style-type: none"> • - RFU : Must be set to value 0 • Byte 2 <ul style="list-style-type: none"> • - b8, b7, b6, : must be 0 • - b5 : Enable display of transaction type in default LCD UI Message • - b4 : Disable ViVOpay Default Contactless Logo • - b3 : Disable ViVOpay default Buzzer UI • - b2 : Disable ViVOpay default LED UI • - b1 : Disable ViVOpay default LCD UI Messages • Byte 3 <ul style="list-style-type: none"> • - b8, b7, b6, b5, b4, b3, b2 : must be 0 • - b1 : Use Event (non-blcking) Mode

Note: To request tags to be included in default response, use tag DFEE1A, and specify tag list. Example four tags 9F02, 9F36, 95, 9F37 to be included in response = DFEE1A079F029F369f9F37

Parameters

<i>isFastEMV</i>	If TRUE, it will populate the IDTTransactionData.fastEMV with ASCII data similar to IDTech FastEMV KB output, after performing an auto-authenticate and auto-complete with ResultCode = Could Not Contact Host
<i>transMode</i>	0 = normal, 1 = fallback from ICC, 2 = fallback from contactless
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString

21.21.1.77 RETURN_CODE IDTechSDK.IDT_VP8800.device_transferFile (string *fileName*, byte[] *file*, bool *isSD* = false, string *ident* = " ")

Transfer File

This command transfers a data file to the reader.

Parameters

<i>fileName</i>	Filename. The data for this command is a ASCII string with the complete path and file name you want to create. You do not need to specify the root directory. Indicate subdirectories with a forward slash (/).
<i>file</i>	The data file.
<i>isSD</i>	TRUE = tranfer to SD Card, FALSE = transfer to Flash.
<i>ip</i>	Optional: IP address to execute command on (for IP connected devices, string ident = "")
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with errorCode.getErrorString(string ident = "")

21.21.1.78 RETURN_CODE IDTechSDK.IDT_VP8800.device_updateDeviceFirmware (byte[] *firmwareData*, string *ident* = " ")

Update Firmware

Updates the firmware .

Parameters

<i>firmwareData</i>	Signed binary data of a firmware file provided by IDTech
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with errorCode.getErrorString(string ident = "")

After you pass the firmwareData file, a new thread will start to execute the firmware download. You will receive status of the progress through callbacks to the IDTechSDK.Callback() delegate. The following parameters will be

passed back:

- sender = IDT_DEVICE_Types.IDT_DEVICE_AUGUSTA
- state = DeviceState.FirmwareUpdate
- transactionResultCode = status of the firmware update (starting, entering bootloader, applying update, block success, firmware success, string ident = "")
- data = File Progress. Four bytes, with bytes [0][1] = current block, and bytes [2][3] = total blocks. 0x00030010 = block 3 of 16

Example code starting a firmware update

```
OpenFileDialog diag = new OpenFileDialog(ident);
diag.Filter = "NGA FW Files|*.fm";

if (diag.ShowDialog() == DialogResult.OK, string ident = "")
{
    byte[] file = File.ReadAllBytes(diag.FileName, ident);
    RETURN_CODE rt = IDT_Device.SharedController.device_updateDeviceFirmware(file, ident);
    if (rt == RETURN_CODE.RETURN_CODE_DO_SUCCESS, string ident = "")
    {
        //Was a success
    }
    else
    {
        //Error starting firmware download
    }
}
```

Example monitoring firmware update status / success

```
private void MessageCallBack(IDTechSDK.IDT_DEVICE_Types type, DeviceState state, byte[] data,
    IDTTransactionData cardData, EMV_Callback emvCallback, RETURN_CODE transactionResultCode, string ident = "")
{
    switch (state, string ident = "")
    {
        case DeviceState.FirmwareUpdate:
            switch (transactionResultCode, string ident = "")
            {
                case RETURN_CODE.RETURN_CODE_FW_STARTING_UPDATE:
                    SetOutputText(IP2 + " Starting Firmware Update\n", ident);
                    break;
                case RETURN_CODE.RETURN_CODE_DO_SUCCESS:
                    SetOutputText(IP2 + " Firmware Update Successful\n", ident);
                    if (type == IDT_DEVICE_Types.IDT_DEVICE_VP8800) SetOutputText("The device will now
reboot and validate the firmware file. This process may take up to 5 minutes. You may proceed once device is
recognized\n", ident);
                    break;
                case RETURN_CODE.RETURN_CODE_APPLYING_FIRMWARE_UPDATE:
                    SetOutputText(IP2 + " Applying Firmware Update....\n", ident);
                    break;
                case RETURN_CODE.RETURN_CODE_ENTERING_BOOTLOADER_MODE:
                    SetOutputText(IP2 + " Entering Bootloader Mode....\n", ident);
                    break;
                case RETURN_CODE.RETURN_CODE_BLOCK_TRANSFER_SUCCESS:
                    int start = data[0];
                    int end = data[1];
                    if (data.Length == 4, string ident = "")
                    {
                        start = data[0] * 0x100 + data[1];
                        end = data[2] * 0x100 + data[3];
                    }

                    SetOutputTextCS(IP2 + " Sent block " + start.ToString() + " of " + end.ToString() +
"\n", ident);
                    break;
                case RETURN_CODE.RETURN_CODE_ERASING_SPI:
                    int start1 = data[0];
                    int end1 = data[1];
                    if (data.Length == 4, string ident = "")
                    {
                        start1 = data[0] * 0x100 + data[1];
                        end1 = data[2] * 0x100 + data[3];
                    }
            }
    }
}
```

```

        }

        SetOutputTextCS(IP2 + " Erasing SPI Flash: " + start1.ToString() + "% complete", ident)
    ;
        break;
    default:
        SetOutputText (IP2 + " Firmware Update Error Code: " + "0x" +
            String.Format("{0:X}", (ushort) transactionResultCode) + ": " +
            IDTechSDK.errorCode.getErrorString(transactionResultCode) + "\r\n", ident);
        break;
    }
    break;
}
}
}

```

21.21.1.79 RETURN_CODE IDTechSDK.IDT_VP8800.device_updateFirmwareFromZip (byte[] zipfile, string ident = "", bool isForeground = false)

Update Firmware From Zip

Performs one or more device firmware updates from firmware files passed as a compressed archive with a valid configuration file (.json format)

Parameters

<i>zipfile</i>	Zip file containing on or more firmware update files and a .json configuration file
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.
<i>isForeground</i>	If TRUE, will block program until firmware update complete. Otherwise, FALSE performs FW update on background.

Returns

RETURN_CODE: Values can be parsed with `errorCode.getErrorString(string ident = "")`

21.21.1.80 RETURN_CODE IDTechSDK.IDT_VP8800.emv_activateTransaction (int timeout, byte[] tags, bool forceOnline, bool isFastEMV = false, string ident = "")

Start EMV Transaction Request

Authorizes the EMV transaction for an ICC card

The tags will be returned in the callback routine.

Parameters

<i>timeout</i>	Timeout value in seconds.
<i>tags</i>	The tags to be included in the request. Passed as TLV Data. Example, tag 9F02 with amount 0x000000000100 would be 0x9F0206000000000100
<i>forceOnline</i>	TRUE = do not allow offline approval, FALSE = allow ICC to approve offline if terminal capable Note: To request tags to be included in default response, use tag DFEE1A, and specify tag list. Example four tags 9F02, 9F36, 95, 9F37 to be included in response = DFEE1A079F029F369F37 Use tag DFEF1F to determine if the Authentication step must be executed. DFEF1F 02 XX YY, where XX = 00 forces Authentication step, and YY = RFU. DFEF1F020000 will force authentication.
<i>isFastEMV</i>	If TRUE, it will populate the IDTTransactionData.fastEMV with ASCII data similar to IDTech FastEMV KB output, after performing an auto-authenticate and auto-complete with ResultCode = Could Not Contact Host
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString

21.21.1.81 RETURN_CODE IDTechSDK.IDT_VP8800.emv_addTerminalData (byte[] *tlv*, string *ident* = " ")

Add Terminal Data

Adds to the exosting Terminal Data

Parameters

<i>tlv</i>	TerminalData TLV Data
<i>ident</i>	Optional identifier

Return values

<i>RETURN_CODE</i>	Return codes listed as typedef enum in IDTCommon:RETURN_CODE. Values can be parsed with IDT_BTPay::device_getResponseCodeString:()
--------------------	--

21.21.1.82 static void IDTechSDK.IDT_VP8800.emv_allowFallback (bool *allow*, string *ident* = " ") [static]

Allow fallback for EMV transactions. Default is TRUE

Parameters

<i>allow</i>	TRUE = allow fallback, FALSE = don't allow fallback
--------------	---

21.21.1.83 static void IDTechSDK.IDT_VP8800.emv_autoAuthenticate (bool *authenticate*, string *ident* = " ") [static]

Enables authenticate for EMV transactions. If a startEMVTransaction results in code 0x0010 (start transaction success), then emv_authenticateEMVTransaction can automatically execute if parameter is set to TRUE

Parameters

<i>authenticate</i>	TRUE = auto authenticate, FALSE = manually authenticate
---------------------	---

21.21.1.84 static void IDTechSDK.IDT_VP8800.emv_autoAuthenticateTags (bool *authenticate*, byte[] *tags*, string *ident* = " ") [static]

Enables authenticate for EMV transactions. If a startEMVTransaction results in code 0x0010 (start transaction success), then emv_authenticateEMVTransaction can automatically execute if parameter is set to TRUE

Parameters

<i>authenticate</i>	TRUE = auto authenticate, FALSE = manually authenticate
<i>tags</i>	Tags to pass during authentication stage;

21.21.1.85 **RETURN_CODE** IDTechSDK.IDT_VP8800.emv_callbackResponseLCD (*EMV_LCD_DISPLAY_MODE type*, *byte selection*, *string ident* = " ")

Callback Response LCD Display

Provides menu selection responses to the kernel after a callback was received with DeviceState.EMVCallback, and callbackType = EMV_CALLBACK_TYPE.EMV_CALLBACK_TYPE_LCD, and lcd_displayMode = EMV_LCD_DISPLAY_MODE_MENU, EMV_LCD_DISPLAY_MODE_PROMPT, or EMV_LCD_DISPLAY_MODE_LANGUAGE_SELECT

Parameters

<i>type</i>	If Cancel key pressed during menu selection, then value is EMV_LCD_DISPLAY_MODE_CANCEL. Otherwise, value can be EMV_LCD_DISPLAY_MODE_MENU, EMV_LCD_DISPLAY_MODE_PROMPT, or EMV_LCD_DISPLAY_MODE_LANGUAGE_SELECT
<i>selection</i>	If type = EMV_LCD_DISPLAY_MODE_MENU or EMV_LCD_DISPLAY_MODE_LANGUAGE_SELECT, provide the selection ID line number. Otherwise, if type = EMV_LCD_DISPLAY_MODE_PROMPT supply either 0x43 ('C') for Cancel, or 0x45 ('E') for Enter/accept
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with errorCode.getErrorString()

21.21.1.86 **RETURN_CODE** IDTechSDK.IDT_VP8800.emv_callbackResponseMSR (*byte[] MSR*, *string ident* = " ")

Callback Response MSR Entry

Provides MSR information to kernel after a callback was received with DeviceState.EMVCallback, and callbackType = EMV_CALLBACK_MSR

Parameters

<i>MSR</i>	Swiped track data
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with errorCode.getErrorString()

21.21.1.87 **RETURN_CODE** IDTechSDK.IDT_VP8800.emv_callbackResponsePIN (*EMV_PIN_MODE type*, *byte[] KSN*, *byte[] PIN*, *string ident* = " ")

Callback Response PIN Entry

Provides (or cancels) PIN entry information to kernel after a callback was received with DeviceState.EMVCallback, and callbackType = EMV_CALLBACK_TYPE_PINPAD

Parameters

<i>type</i>	If Cancel key pressed during PIN entry, then value is EMV_PIN_MODE_CANCEL. Otherwise, value can be EMV_PIN_MODE_ONLINE_DUKPT, EMV_PIN_MODE_ONLINE_MKSK, or EMV_PIN_MODE_OFFLINE
-------------	---

Parameters

<i>KSN</i>	If enciphered PIN, this is either the PIN DUKPT Key (EMV_PIN_MODE_ONLINE_DUKPT) or PIN Session Key (EMV_PIN_MODE_ONLINE_MKSK), or PIN Pairing DUKPT key (EMV_PIN_MODE_OFFLINE, string ident = "")
<i>PIN</i>	If enciphered PIN, this is encrypted PIN block. If device does not implement pairing function, this is plaintext PIN
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString

21.21.1.88 RETURN_CODE IDTechSDK.IDT_VP8800.emv_cancelTransaction (string ident = " ")

Cancel Transaction

Cancels the currently executing EMV or CTLS transaction.

Parameters

<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.
--------------	--

Returns

RETURN_CODE: Values can be parsed with errorCode.getErrorString()

21.21.1.89 RETURN_CODE IDTechSDK.IDT_VP8800.emv_clearTransactionLog (string ident = " ")

Clear Transaction Log

Clears the transaction log.

Parameters

<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.
--------------	--

Returns

RETURN_CODE: Values can be parsed with errorCode.getErrorString()

21.21.1.90 RETURN_CODE IDTechSDK.IDT_VP8800.emv_completeTransaction (bool commError, byte[] authCode, byte[] iad, byte[] tlvScripts, byte[] tlv, string ident = " ")

Complete EMV Transaction Request

Completes the EMV transaction for an ICC card when online authorization request is received from emv_↔ authenticateTransaction

The tags will be returned in the callback routine.

Parameters

<i>commError</i>	Communication error with host. Set to TRUE if host was unreachable, or FALSE if host response received. If Communication error, authCode, iad, tlvScripts can be null.
<i>authCode</i>	Authorization code from host. Two bytes. Example 0x3030. (Tag value 8A). Required
<i>iad</i>	Issuer Authentication Data, if any. Example 0x11223344556677883030 (tag value 91).
<i>tlvScripts</i>	71/72 scripts, if any
<i>tlv</i>	Additional TVL data to return with transaction results (if any, string ident = "")
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString

NOTE: There are three possible outcomes for Authorization Code: Approval, Refer To Bank, Decline. The kernel maps these three outcomes to valid authorization response codes using tag DFEE1B through 8 bytes: {2 bytes Approval Code}{2 bytes Referral Code}{2 bytes Decline Code}{2 bytes RFU} If your gateway uses "00" for Approval, "01" for Referral, and "05" for Decline, then DFEE1B 08 3030 3031 3035 0000 If you use an authorization code value that is not defined in DFEE1B, the kernel will use the DECLINE value of DFEE1B by default.

21.21.1.91 RETURN_CODE IDTechSDK.IDT_VP8800.emv_continueTransactionForCV (int result, byte[] pinblock, string ident = "")

Continue Transaction for Cardholder Verification

Use this command to send the results of Online PIN Request or Signature Request and to continue a Contact EMV transaction. If the prior response Status Code is 0x31h (Request Online PIN) or 0x32h (Request Signature), this is the next command to send.

The tags will be returned in the callback routine.

```
@param result Cardholder Verification result:
- 0: Success (PIN or Signature, string ident = "")
- 1: Cancelled PIN request
- 2: PIN pad not working
- 3: Timeout
- 4: Error
@param pinblock Encrypted PIN block if Success PIN result
@param ident Device ID to send command to. If not specified, current SDK default device will be used.
```

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString

21.21.1.92 RETURN_CODE IDTechSDK.IDT_VP8800.emv_getConfigurationGroup (int group, ref byte[] tlv, string ident = "")

Get Configuration Group

Retrieves the Configuration for the specified Group. Group 0 = terminal settings.

Parameters

<i>group</i>	Configuration Group
<i>tlv</i>	return data
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with `errorCode.getErrorString()`

21.21.1.93 RETURN_CODE IDTechSDK.IDT_VP8800.emv_getCRLStatus (ref byte[] *status*, string *ident* = " ")

Get CRL Status

This command returns information about the EMV Certificate Revocation List. The version number, record size, and number of records contained in the file are returned.

Parameters

<i>status</i>	12 bytes returned <ul style="list-style-type: none"> • bytes 0-3 = Version Number • bytes 4-7 = Number of records • bytes 8-11 = Size of record
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with `errorCode.getErrorString()`

21.21.1.94 RETURN_CODE IDTechSDK.IDT_VP8800.emv_getEMVConfigurationCheckValue (ref string *response*, string *ident* = " ")

Polls device for EMV Configuration Check Value**Parameters**

<i>response</i>	Response returned of Check Value of Configuration
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with `errorCode.getErrorString(string ident = "")`

21.21.1.95 RETURN_CODE IDTechSDK.IDT_VP8800.emv_getEMVKernelCheckValue (ref string *response*, string *ident* = " ")

Polls device for EMV Kernel Check Value**Parameters**

<i>response</i>	Response returned of Check Value of Kernel
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with `errorCode.getErrorString(string ident = "")`

21.21.1.96 **RETURN_CODE** IDTechSDK.IDT_VP8800.emv_getEMVKernelVersion (ref string *response*, string *ident* = " ")

Polls device for EMV Kernel Version

Parameters

<i>response</i>	Response returned of Kernel Version
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with `errorCode.getErrorString(string ident = "")`

21.21.1.97 **RETURN_CODE** IDTechSDK.IDT_VP8800.emv_getEMVKernelVersionExt (ref string *response*, string *ident* = " ")

Polls device for Extended EMV Kernel Version

Parameters

<i>response</i>	Response returned of Kernel Version
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with `errorCode.getErrorString(string ident = "")`

21.21.1.98 **RETURN_CODE** IDTechSDK.IDT_VP8800.emv_getExemptionLogStatus (ref byte[] *status*, string *ident* = " ")

Get EMV Exception Log Status

This command returns information about the EMV Exception log. The version number, record size, and number of records contained in the file are returned.

Parameters

<i>status</i>	12 bytes returned <ul style="list-style-type: none"> • bytes 0-3 = Version Number • bytes 4-7 = Number of records • bytes 8-11 = Size of record
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with `errorCode.getErrorString()`

21.21.1.99 **RETURN_CODE** IDTechSDK.IDT_VP8800.emv_getTransactionLogRecord (ref byte[] *record*, ref int *remaining*, string *ident* = " ")

Get Transaction Log Record

Retrieves oldest transaction record on the Transaction Log. At successful completion, the oldest transaction record is deleted from the transaction log

Parameters

<i>record</i>	Transaction Record
<i>remaining</i>	Number of records remaining on the transaction log

Length	Description	Type
4	Transaction Log State (TLS)	Enum (4-byte number, LSB first), SENT ONLINE = 0, NOT SENT = 1
4	Transaction Log Content (TLC)	Enum (4-byte number, LSB first), BATCH = 0, OFFLINE ADVICE = 1, ONLINEADVICE = 2, REVERSAL = 3
4	AppExpDate	unsigned char [4]
3	AuthRespCode	unsigned char [3]
3	MerchantCategoryCode	unsigned char [3]
16	MerchantID	unsigned char [16]
2	PosEntryMode	unsigned char [2]
9	TermID	unsigned char [9]
3	AIP	unsigned char [3]
3	ATC	unsigned char [3]
33	IssuerAppData	unsigned char [33]
6	TVR	unsigned char [6]
3	TSI	unsigned char [3]
11	Pan	unsigned char [11]
2	PanSQNCNum	unsigned char [2]
3	TermCountryCode	unsigned char [3]
7	TranAmount	unsigned char [7]
3	TranCurCode	unsigned char [3]
4	TranDate	unsigned char [4]
2	TranType	unsigned char [2]
9	IFDSerialNum	unsigned char [9]
12	AcquirerID	unsigned char [12]
2	CID	unsigned char [2]
9	AppCryptogram	unsigned char [9]
5	UnpNum	unsigned char [5]
7	AmountAuth	unsigned char [7]
4	AppEffDate	unsigned char [4]
4	CVMResults	unsigned char [4]
129	IssScriptResults	unsigned char [129]
4	TermCap	unsigned char [4]
2	TermType	unsigned char [2]
20	Track2	unsigned char [20]
4	TranTime	unsigned char [4]
7	AmountOther	unsigned char [7]
1	Unused	Unsigned char [1]

Parameters

<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.
--------------	--

Returns

RETURN_CODE: Values can be parsed with `errorCode.getErrorString()`

21.21.1.100 RETURN_CODE IDTechSDK.IDT_VP8800.emv_getTransactionLogStatus (ref byte[] *status*, string *ident* = " ")

Get Transaction Log Status

This command returns information about the EMV transaction log. The version number, record size, and number of records contained in the file are returned.

Parameters

<i>status</i>	12 bytes returned <ul style="list-style-type: none"> • bytes 0-3 = Version Number • bytes 4-7 = Number of records • bytes 8-11 = Size of record
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with `errorCode.getErrorString()`

21.21.1.101 RETURN_CODE IDTechSDK.IDT_VP8800.emv_removeAllCAPK (string *ident* = " ")

Remove All Certificate Authority Public Key

Removes all the CAPK for EMV Kernel

Parameters

<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.
--------------	--

Returns

RETURN_CODE: Values can be parsed with `errorCode.getErrorString(string ident = "")`

21.21.1.102 RETURN_CODE IDTechSDK.IDT_VP8800.emv_removeAllCRL (string *ident* = " ")

Remove All Certificate Revocation List Entries

Removes all CRLentry entries

Parameters

<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.
--------------	--

Returns

RETURN_CODE: Values can be parsed with `errorCode.getErrorString(string ident = "")`

21.21.1.103 RETURN_CODE IDTechSDK.IDT_VP8800.emv_removeAllExceptions (string *ident* = " ")

Remove All EMV Exceptions

Removes all entries from the EMV Exception List

Parameters

<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.
--------------	--

Returns

RETURN_CODE: Values can be parsed with `errorCode.getErrorString()`

21.21.1.104 RETURN_CODE IDTechSDK.IDT_VP8800.emv_removeApplicationData (byte[] *AID*, string *ident* = " ")

Remove Application Data by AID

Removes the Application Data for EMV Kernel as specified by the AID name passed as a parameter

Parameters

<i>AID</i>	Name of ApplicationID Must be between 5 and 16 bytes
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with `errorCode.getErrorString(string ident = "")`

21.21.1.105 RETURN_CODE IDTechSDK.IDT_VP8800.emv_removeCAPK (byte[] *capk*, string *ident* = " ")

Remove Certificate Authority Public Key

Removes the CAPK for EMV Kernel as specified by the RID/Index

Parameters

<i>capk</i>	6 byte CAPK = 5 bytes RID + 1 byte INDEX
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with `errorCode.getErrorString(string ident = "")`

21.21.1.106 RETURN_CODE IDTechSDK.IDT_VP8800.emv_removeConfigurationGroup (int *group*, string *ident* = " ")

Remove Configuration Group

Removes the Configuration as specified by the Group. Must not by group 0

Parameters

<i>group</i>	Configuration Group
--------------	---------------------

Return values

<i>RETURN_CODE</i>	Return codes listed as typedef enum in IDTCommon:RETURN_CODE. Values can be parsed with IDT_VP8800::device_getResponseCodeString :(string ident = "")
--------------------	---

21.21.1.107 **RETURN_CODE** IDTechSDK.IDT_VP8800.emv_removeCRL (byte[] *crl*, string *ident* = " ")

Remove Certificate Revocation List Entry

Removes CRL entry as specified by the RID and Index passed as 6 bytes, or RID, Index, and serial number passed as 9 bytes

Parameters

<i>crl</i>	containing the CRL to remove: where [CRL] is 9 bytes: [5 bytes RID][1 byte CAPK Index][3 bytes serial number] OR [CRL] is 6 bytes: [5 bytes RID][1 byte CAPK Index]
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with [device_getResponseCodeString](#)

21.21.1.108 **RETURN_CODE** IDTechSDK.IDT_VP8800.emv_removeException (byte[] *exception*, string *ident* = " ")

Remove EMV Exception

Removes an entry to the EMV Exception List

Parameters

<i>exception</i>	EMV Exception entry containing the PAN and Sequence Number where [Exception] is 12 bytes: [1 byte Len][10 bytes PAN][1 byte Sequence Number] PAN, in compressed numeric, padded with F if required (example 0x5413339000001596FFFF, string ident = "")
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with [errorCode.getErrorString\(\)](#)

21.21.1.109 **RETURN_CODE** IDTechSDK.IDT_VP8800.emv_removeTerminalData (string *ident* = " ")

Remove Terminal Data

Removes the Terminal Data

Parameters

<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.
--------------	--

Returns

RETURN_CODE: Values can be parsed with `device_getResponseCodeString`

21.21.1.110 RETURN_CODE IDTechSDK.IDT_VP8800.emv_removeTransactionAmountLog (string *ident* = " ")**Remove Transaction Amount Log**

This command can delete transaction amount log in reader. (When EMV transaction is offline approved, or online, transaction amount log saves to reader., string *ident* = "")

Parameters

<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.
--------------	--

Returns

RETURN_CODE: Values can be parsed with `device_getResponseCodeString`

21.21.1.111 RETURN_CODE IDTechSDK.IDT_VP8800.emv_resetConfigurationGroup (int *group*, string *ident* = " ")**Reset Configuration Group**

This command allows resetting a dataset to its default configuration. If the file exists, it will be overwritten. If not, it will be created.

Parameters

<i>group</i>	Configuration Group
--------------	---------------------

Return values

<i>RETURN_CODE</i>	Return codes listed as typedef enum in IDTCommon:RETURN_CODE. Values can be parsed with IDT_VP8800::device_getResponseCodeString :(string <i>ident</i> = "")
--------------------	--

21.21.1.112 RETURN_CODE IDTechSDK.IDT_VP8800.emv_retrieveAIDList (ref byte *response*[[], string *ident* = " ")**Retrieve AID list**

Returns all the AID names and their assigned groups installed on the terminal for CONTACT.

Parameters

<i>response</i>	array of 2-tag TLV data objects: DFEE2D (group name) followed by 9F06 (AID, string <i>ident</i> = "")
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with `errorCode.getErrorString`(string *ident* = "")

21.21.1.113 **RETURN_CODE** IDTechSDK.IDT_VP8800.emv_retrieveApplicationData (byte[] *AID*, ref byte[] *tlv*, string *ident* = " ")

Retrieve Application Data by AID

Retrieves the Application Data for EMV Kernel as specified by the AID name passed as a parameter.

Parameters

<i>AID</i>	Name of ApplicationID. Must be between 5 and 16 bytes
<i>tlv</i>	The TLV elements of the requested AID
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with `errorCode.getErrorString()`

21.21.1.114 **RETURN_CODE** IDTechSDK.IDT_VP8800.emv_retrieveCAPK (byte[] *capk*, ref byte[] *key*, string *ident* = " ")

Retrieve Certificate Authority Public Key

Retrieves the CAPK for EMV Kernel as specified by the RID/Index passed as a parameter.

Parameters

<i>capk</i>	6 bytes CAPK = 5 bytes RID + 1 byte Index
<i>key</i>	Response returned as a CAKey format: [1 byte Hash Algorithm][1 byte Encryption Algorithm][20 bytes HashValue][4 bytes Public Key Exponent][2 bytes Modulus Length][Variable bytes Modulus] Where: <ul style="list-style-type: none"> • Hash Algorithm: The only algorithm supported is SHA-1. The value is set to 0x01 • Encryption Algorithm: The encryption algorithm in which this key is used. Currently support only one type: RSA. The value is set to 0x01. • HashValue: Which is calculated using SHA-1 over the following fields: RID & Index & Modulus & Exponent • Public Key Exponent: Actually, the real length of the exponent is either one byte or 3 bytes. It can have two values: 3 (Format is 0x00 00 00 03), or 65537 (Format is 0x00 01 00 01, string <i>ident</i> = "") • Modulus Length: LenL LenH Indicated the length of the next field. • Modulus: This is the modulus field of the public key. Its length is specified in the field above.
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with `errorCode.getErrorString()`

21.21.1.115 **RETURN_CODE** IDTechSDK.IDT_VP8800.emv_retrieveCAPKList (ref byte[] *keys*, string *ident* = " ")

Retrieve the Certificate Authority Public Key list

Returns all the CAPK RID and Index installed on the terminal for EMV Kernel.

Parameters

<i>keys</i>	[key1][key2]...[keyn], each key 6 bytes where key = 5 bytes RID + 1 byte index
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with `errorCode.getErrorString()`

21.21.1.116 RETURN_CODE IDTechSDK.IDT_VP8800.emv_retrieveCRLList (ref byte[] list, string ident = " ")

Retrieve the Certificate Revocation List

Returns the CRL entries on the terminal.

Parameters

<i>list</i>	[CRL1][CRL2]...[CRLn], each CRL 9 bytes where CRL = 5 bytes RID + 1 byte index + 3 bytes serial number
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with `errorCode.getErrorString()`

21.21.1.117 RETURN_CODE IDTechSDK.IDT_VP8800.emv_retrieveExceptionList (ref byte[] list, string ident = " ")

Retrieve the EMV Exception List

Returns the EMV Exception entries on the terminal.

Parameters

<i>list</i>	[Exception1][Exception2]...[Exceptionn], where [Exception] is 12 bytes: [1 byte Len][10 bytes PAN][1 byte Sequence Number]
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with `errorCode.getErrorString()`

21.21.1.118 RETURN_CODE IDTechSDK.IDT_VP8800.emv_retrieveTerminalData (ref byte[] tlv, string ident = " ")

Retrieve Terminal Data

Retrieves the Terminal Data for EMV Kernel.

Parameters

<i>tlv</i>	Response returned as a TLV
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Parameters

<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.
--------------	--

Returns

RETURN_CODE: Values can be parsed with `errorCode.getErrorString()`

21.21.1.121 RETURN_CODE IDTechSDK.IDT_VP8800.emv_setCAPK (byte[] key, string ident = " ")

Set Certificate Authority Public Key

Sets the CAPK for EMV Kernel as specified by the CAKey structure

Parameters

<i>key</i>	CAKey format: [5 bytes RID][1 byte Index][1 byte Hash Algorithm][1 byte Encryption Algorithm][20 bytes HashValue][4 bytes Public Key Exponent][2 bytes Modulus Length][Variable bytes Modulus] Where: <ul style="list-style-type: none"> • Hash Algorithm: The only algorithm supported is SHA-1. The value is set to 0x01 • Encryption Algorithm: The encryption algorithm in which this key is used. Currently support only one type: RSA. The value is set to 0x01. • HashValue: Which is calculated using SHA-1 over the following fields: RID & Index & Modulus & Exponent • Public Key Exponent: Actually, the real length of the exponent is either one byte or 3 bytes. It can have two values: 3 (Format is 0x00 00 00 03), or 65537 (Format is 0x00 01 00 01, string ident = "") • Modulus Length: LenL LenH Indicated the length of the next field. • Modulus: This is the modulus field of the public key. Its length is specified in the field above.
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with `errorCode.getErrorString()`

21.21.1.122 RETURN_CODE IDTechSDK.IDT_VP8800.emv_setCRL (byte[] crl, string ident = " ")

Set Certificate Revocation List

Sets the CRL

Parameters

<i>crl</i>	CRL Entries containing the RID, Index, and serial numbers to set where [CRL] is 9 bytes: [5 bytes RID][1 byte CAPK Index][3 bytes serial number]
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with `errorCode.getErrorString()`

21.21.1.123 RETURN_CODE IDTechSDK.IDT_VP8800.emv_setException (*byte[] exception*, *string ident* = " ")

Set EMV Exception

Adds an entry to the EMV Exception List

Parameters

<i>exception</i>	EMV Exception entry containing the PAN and Sequence Number where [Exception] is 12 bytes: [1 byte Len][10 bytes PAN][1 byte Sequence Number] PAN, in compressed numeric, padded with F if required (example 0x5413339000001596FFFF, string <i>ident</i> = "")
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with `errorCode.getErrorString()`

21.21.1.124 RETURN_CODE IDTechSDK.IDT_VP8800.emv_setTerminalDataVP8800 (*byte[] tlv*, *int config*, *string ident* = " ")

Set Terminal Data

Sets the Terminal Data

Parameters

<i>tlv</i>	TerminalData TLV Data
<i>config</i>	Config verification, valid values 1-4
<i>ident</i>	Optional identifier

Return values

<i>RETURN_CODE</i>	Return codes listed as typedef enum in IDTCommon:RETURN_CODE. Values can be parsed with <code>IDT_BTPay::device_getResponseCodeString()</code>
--------------------	--

21.21.1.125 RETURN_CODE IDTechSDK.IDT_VP8800.emv_startTransaction (*double amount*, *double amtOther*, *int exponent*, *int type*, *int timeout*, *byte[] tags*, *bool forceOnline*, *string ident* = " ")

Start EMV Transaction Request

Authorizes the EMV transaction for an ICC card

The tags will be returned in the callback routine.

Parameters

<i>amount</i>	Transaction amount value (tag value 9F02, string <i>ident</i> = "")
<i>amtOther</i>	Other amount value, if any (tag value 9F03, string <i>ident</i> = "")
<i>exponent</i>	Number of characters after decimile point
<i>type</i>	Transaction type (tag value 9C).

Parameters

<i>timeout</i>	Timeout value in seconds.
<i>tags</i>	Any other tags to be included in the request. Passed as TLV Data. Example, tag 9F02 with amount 0x000000000100 would be 0x9F0206000000000100 If tags 9F02 (amount), 9F03 (other amount), or 9C (transaction type) are included, they will take priority over these values supplied as individual parameters to this method.
<i>forceOnline</i>	TRUE = do not allow offline approval, FALSE = allow ICC to approve offline if terminal capable Note: To request tags to be included in default response, use tag DFEE1A, and specify tag list. Example four tags 9F02, 9F36, 95, 9F37 to be included in response = DFEE1A079F029F369F37 Use tag DFEF1F to determine if the Authentication step must be executed. DFEF1F 02 XX YY, where XX = 00 forces Authentication step, and YY = RFU. DFEF1F020000 will force authentication.
<i>isFastEMV</i>	If TRUE, it will populate the IDTTransactionData.fastEMV with ASCII data similar to IDTech FastEMV KB output, after performing an auto-authenticate and auto-complete with ResultCode = Could Not Contact Host
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString

21.21.1.126 RETURN_CODE IDTechSDK.IDT_VP8800.emv_trySetTerminalData (byte[] *tlv*, ref byte[] *rejectedTLV*, ref byte[] *convertedTLV*, bool *overwrite* = false, string *ident* = " ")

Try to Set Terminal Data

Attempts to set the Terminal Data. Use this functions to attempt to set the terminal data to the device. This function will allow the cross-device tag compatible writing of IDTech Legacy/New tag definitions

Parameters

<i>tlv</i>	TerminalData TLV Data
<i>rejectedTLV</i>	Contains the tags (if any) that were rejected by the firmware
<i>convertedTLV</i>	Contains the tags (if any) that were converted and accepted by the firmware
<i>overwrite</i>	TRUE = add TLV to existing tags, FALSE = replace existing tags with TLV
<i>ident</i>	Optional identifier

Return values

<i>RETURN_CODE</i>	Return codes listed as typedef enum in IDTCommon:RETURN_CODE. Values can be parsed with IDT_BTPay::device_getResponseCodeString:()
--------------------	--

21.21.1.127 RETURN_CODE IDTechSDK.IDT_VP8800.felica_authentication (byte[] *key*, string *ident* = " ")

FeliCa Authentication

Provides a key to be used in a follow up FeliCa Read with MAC (3 blocks max) or Write with MAC (1 block max). This command must be executed before each Read w/MAC or Write w/MAC command

NOTE: The reader must be in Pass Through Mode for FeliCa commands to work.

Parameters

<i>key</i>	16 byte key used for MAC generation of Read or Write with MAC
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with `errorCode.getErrorString(string ident = "")`

21.21.1.128 RETURN_CODE IDTechSDK.IDT_VP8800.felica_read (*byte[] serviceCode*, *int numBlocks*, *byte[] blockList*, *ref byte[] blocks*, *string ident = " "*)

FeliCa Read

Reads up to 4 blocks.

NOTE: The reader must be in Pass Through Mode for FeliCa commands to work.

Parameters

<i>serviceCode</i>	Service Code List. Each service code in Service Code List = 2 bytes of data
<i>numBlocks</i>	Number of blocks
<i>blockList</i>	Blocks to read. Maximum 4 block requests
<i>blocks</i>	Blocks read. Each block 16 bytes.
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with `errorCode.getErrorString(string ident = "")`

21.21.1.129 RETURN_CODE IDTechSDK.IDT_VP8800.felica_readWithMac (*int numBlocks*, *byte[] blockList*, *ref byte[] blocks*, *string ident = " "*)

FeliCa Read with MAC Generation

Reads up to 3 blocks with MAC Generation. FeliCa Authentication must be performed first

NOTE: The reader must be in Pass Through Mode for FeliCa commands to work.

Parameters

<i>numBlocks</i>	Number of blocks
<i>blockList</i>	Block to read. Each block in blockList Maximum 3 block requests
<i>blocks</i>	Blocks read. Each block 16 bytes.
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with `errorCode.getErrorString(string ident = "")`

21.21.1.130 RETURN_CODE IDTechSDK.IDT_VP8800.felica_requestService (byte[] *nodeCode*, ref byte[] *response*, string *ident* = " ")

FeliCa Request Service

Perform functions a Felica Request Service

NOTE: The reader must be in Pass Through Mode for FeliCa commands to work.

Parameters

<i>nodeCode</i>	Node Code
<i>response</i>	Response as explained in FeliCA Lite-S User's Manual
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with `errorCode.getErrorString(string ident = "")`

21.21.1.131 RETURN_CODE IDTechSDK.IDT_VP8800.felica_SendCommand (byte[] *command*, ref byte[] *response*, string *ident* = " ")

FeliCa Send Command

Send a Felica Command

Parameters

<i>command</i>	Command data from settlement center to be sent to felica card
<i>response</i>	Response data from felica card.

Returns

RETURN_CODE: Values can be parsed with `device_getIDGStatusCodeString(string ident = "")`

21.21.1.132 RETURN_CODE IDTechSDK.IDT_VP8800.felica_write (byte[] *serviceCode*, int *blockCount*, byte[] *blockList*, byte[] *data*, ref byte[] *statusFlag*, string *ident* = " ")

FeliCa Write

Writes a block

NOTE: The reader must be in Pass Through Mode for FeliCa commands to work.

Parameters

<i>serviceCode</i>	Service Code list. Each service code must be be 2 bytes
<i>blockCount</i>	Block Count
<i>blockList</i>	Block list.
<i>data</i>	Block to write. Must be 16 bytes.
<i>statusFlag</i>	Status flag response as explained in FeliCA Lite-S User's Manual, Section 4.5
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with `errorCode.getErrorString(string ident = "")`

21.21.1.133 RETURN_CODE IDTechSDK.IDT_VP8800.felica_writeWithMac (int *blockNumber*, byte[] *data*, string *ident* = " ")

FeliCa Write with MAC Generation

Writes a block with MAC Generation. FeliCa Authentication must be performed first

NOTE: The reader must be in Pass Through Mode for FeliCa commands to work.

Parameters

<i>blockNumber</i>	Number of block
<i>data</i>	Block to write. Must be 16 bytes.
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with `errorCode.getErrorString(string ident = "")`

21.21.1.134 static int IDTechSDK.IDT_VP8800.getCommandTimeout (string *ident* = " ") [static]

Get Command Timeout

Gets the default timeout (in milliseconds) waiting for a blocking command response

Return values

<i>time</i>	Time
-------------	------

21.21.1.135 RETURN_CODE IDTechSDK.IDT_VP8800.icc_exchangeAPDU (string *c_APDU*, ref byte[] *response*, string *ident* = " ")

Exchange APDU

Sends an APDU packet to the ICC. If successful, response is returned in APDUResult class instance in response parameter.

Parameters

<i>c_APDU</i>	APDU data packet
<i>response</i>	Unencrypted/encrypted parsed APDU response
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with `errorCode.getErrorString(string ident = "")`

21.21.1.136 RETURN_CODE IDTechSDK.IDT_VP8800.icc_powerOffICC (string *ident* = " ")

Power Off ICC

Powers down the ICC

Parameters

<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.
--------------	--

Returns

RETURN_CODE: Values can be parsed with `errorCode.getErrorString(string ident = "")`

If Success, empty If Failure, ASCII encoded data of error string

21.21.1.137 RETURN_CODE IDTechSDK.IDT_VP8800.icc_powerOnICC (ref byte[] *ATR*, byte *interfaces*, string *ident* = " ")

Power On ICC

Power up the currently selected microprocessor card in the ICC reader

Parameters

<i>interfaces</i>	For NEO2 devices allowed interfaces for which to get the ATR. 0x20h = retrieve last ATR received from PICC 0x21h = SAM1 (SRED version only, string ident = "") 0x22h = SAM2 (SRED version only, string ident = "") For other devices interfaces euquals to 0s
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with `errorCode.getErrorString(string ident = "")`

21.21.1.138 RETURN_CODE IDTechSDK.IDT_VP8800.lcd_captureSignature (int *timeout*, int *format*, string *ident* = " ")

Enables Signature Capture

This command executes the signature capture screen. Once a signature is captured, it is sent to the callback with `DeviceState.Signature`, and the data will contain a .png of the signature

Parameters

<i>timeout</i>	Timeout waiting for the signature capture in 60 seconds
<i>format</i>	1 = Windows BMP, 2 = png, 3 = raw
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with `errorCode.getErrorString(string ident = "")`

21.21.1.139 static RETURN_CODE IDTechSDK.IDT_VP8800.lcd_clearDisplay (string *ident* = " ") [static]

Clear Display

Command to clear the display screen on the reader.It returns the display to the currently defined background color and terminates all events

Parameters

<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.
--------------	--

Returns

RETURN_CODE: Values can be parsed with `errorCode.getErrorString(string ident = "")`

21.21.1.140 RETURN_CODE IDTechSDK.IDT_VP8800.lcd_clearInputEvents (string *ident* = " ")

Clear Input Events

Clears the input event buffer.

Parameters

<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.
--------------	--

Returns

RETURN_CODE: Values can be parsed with `errorCode.getErrorString(string ident = "")`

21.21.1.141 RETURN_CODE IDTechSDK.IDT_VP8800.lcd_customDisplayMode (bool *enable*, string *ident* = " ")

Custom Display Mode

Controls the LCD display mode to custom display. Keyboard entry is limited to the Cancel, Clear, Enter and the function keys, if present. PIN entry is not permitted while the reader is in Custom Display Mode

Parameters

<i>enable</i>	TRUE = enabled, FALSE = disabled
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with `errorCode.getErrorString(string ident = "")`

21.21.1.142 RETURN_CODE IDTechSDK.IDT_VP8800.lcd_displayButton (int *xCord*, int *yCord*, int *xWidth*, int *yWidth*, int *font*, int *style*, int *displayProp*, string *text*, ref byte[] *identifier*, byte[] *foreRGB* = null, byte[] *backRGB* = null, string *ident* = " ")

Display Button

Displays a button on the screen. To remove buttons, use [lcd_clearDisplay\(\)](#). The button becomes sensitive after activating `lcd_getInputEvent(string ident = "")` and remains active until the display is cleared.

While in Secure Mode (Pin Pad active), the text must match one of the following messages:

DONE, CREDIT, DEBIT, CANCEL, CLEAR, OK, YES, NO, LOYALTY, EBT, GIFT CARD, CASH, TOUCH, ACCEPT, ENTER, DOES NOT AGREE, BACK, NEXT, PREVIOUS, HOME, DISAGREE, REJECT, CONTINUE, USE, DON'T USE, SELECT, NO THANK YOU, PAGE UP,

Parameters

<i>xCord</i>	The x Coordinate for the text
<i>yCord</i>	The y Coordinate for the text
<i>xWidth</i>	Width of text field
<i>yWidth</i>	Height of text field
<i>font</i>	Font. Default font = 1
<i>style</i>	Font style: <ul style="list-style-type: none"> • 1 = 13px Regular • 2 = 17px Regular • 3 = 17px Bold • 4 = 22px Regular • 5 = 20px Regular • 6 = 20px Bold • 7 = 29px Regular • 8 = 38px Regular • 9 = 38px Bold • 10 = 58px Regular
<i>displayProp</i>	<ul style="list-style-type: none"> • 0 = Center on line Y, do not clear screen, word wrap disabled • 1 = Center on line Y, clear screen before displaying button, word wrap disabled • 2 = Display button at (X,Y) specified, do not clear screen, word wrap disabled • 3 = Display button at (X,Y) specified, clear screen before displaying button, word wrap disabled • 4 = Center button on screen, do not clear screen, word wrap disabled • 5 = Center button on screen, clear screen before displaying button, word wrap disabled. • 64 = Center on line Y, do not clear screen, word wrap enabled • 65 = Center on line Y, clear screen before displaying button, word wrap enabled • 66 = Display button at (X,Y) specified, do not clear screen, word wrap enabled • 67 = Display button at (X,Y) specified, clear screen before displaying button, word wrap enabled • 68 = Center button on screen, do not clear screen, word wrap enabled • 69 = Center button on screen, clear screen before displaying button, word wrap enabled
<i>text</i>	Text to display. When in Secure Mode, must match secure messages exactly. When in custom mode, it can be any message.
<i>identifier</i>	The 4 byte Graphic ID assigned to the element once created
<i>foreRGB</i>	Text Color RGB. 000000 = black, FF0000 = red, 00FF00 = green, 0000FF = blue, FFFFFFFF = white. Pass NULL for default color
<i>backRGB</i>	Button Color RGB. 000000 = black, FF0000 = red, 00FF00 = green, 0000FF = blue, FFFFFFFF = white. Pass NULL for default color
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with `errorCode.getErrorString(string ident = "")`

21.21.1.143 RETURN_CODE IDTechSDK.IDT_VP8800.lcd_displayText (int *xCord*, int *yCord*, int *xWidth*, int *yWidth*, int *font*, int *style*, int *displayProp*, string *text*, ref byte[] *identifier*, string *ident* = " ")

Display Text

Displays text on the screen. While in Secure Mode (Pin Pad active), the text must match one of the following messages: Approved, approved avail, approved available, authorized, authorizing please wait, available, balance, call your bank, cancel, cancelled, cancel to reject, card read ok, card read ok remove card, cash, cash back?, Choose transaction type, clear, confirm amount, connecting online, convert to credit?, Copyright, credit, debit, declined, done, end of key life, enter configuration id, enter date and time, enter force transaction online, fail, fare, fatal error, fee, initializing, input date of birth and press enter, input joint applicant date of birth and press enter, input joint applicant social security number and press enter, input social security number and press enter, insert or swipe card, international card, international card please insert, international card please swipe, invalid entry, is amount ok?, Keys not found, no card, not accepted, not authorized, not connected, offline, offline available fund, other, other amount, out of order, payment, pin try limit exceeded, please enter amount, please enter cash back amount, please enter phone, please enter tip, please enter tip amount using keypad, please enter tip option using keypad, please enter zip code, please insert card, please insert or swipe card, please present card, please present one card only, please press enter on keypad to continue, please press enter to continue, please push enter, please re-enter phone number, please re-enter zip code, please remove card, please select 1 card, please select option, please sign on the screen, please sign the receipt, please swipe card, please tap card, please tap or swipe card, please use chip reader, please use keypad to confirm, please use keypad to confirm or cancel, please use keypad to select account, please use keypad to select option, please use other visa card, please wait, please wait..., present card, present one card only, press cancel to reject, press enter to accept, processing, processing..., push enter, receipt?, Remove card please wait, signature required, signature required transaction not completed, subtotal, swipe again, swipe card, tap again, tap card, tap or swipe card, thank you, timeout, tip, tip amount, total, total charged to card, transaction complete, transaction completed, transaction not completed, unit disabled, vivotech, inc., voucher, welcome, would you like a receipt?

Parameters

<i>xCord</i>	The x Coordinate for the text
<i>yCord</i>	The y Coordinate for the text
<i>xWidth</i>	The display width - Enter 0 for default display dimensions
<i>yWidth</i>	The display height - Enter 0 for default display dimensions
<i>font</i>	Font. Default font = 1
<i>style</i>	Font style: <ul style="list-style-type: none"> • 1 = 13px Regular • 2 = 17px Regular • 3 = 17px Bold • 4 = 22px Regular • 5 = 20px Regular • 6 = 20px Bold • 7 = 29px Regular • 8 = 38px Regular • 9 = 38px Bold • 10 = 58px Regular

Parameters

<i>displayProp</i>	<ul style="list-style-type: none"> • 0 = Center on line Y, do not clear screen • 1 = Center on line Y, clear screen before displaying button • 2 = Display text at (X,Y) specified, do not clear screen • 3 = Display text at (X,Y) specified, clear screen before displaying button • 4 = Center text on screen, do not clear screen • 5 = Center text on screen, clear screen before displaying button • 6 = right justified, do not clear screen before display amount • 7 = right justified, clear screen before displaying amount
<i>text</i>	Text to display. When in Secure Mode, must match secure messages exactly. When in custom mode, it can be any message.
<i>identifier</i>	The 4 byte Graphic ID assigned to the element once created
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with `errorCode.getErrorString(string ident = "")`

21.21.1.144 RETURN_CODE IDTechSDK.IDT_VP8800.lcd_getInputEvent (byte timeout, string ident = " ")

Get Input Event

Returns the event in the input buffer. If no events in the input buffer, with will wait the specified timeout for an event to occur. It will continue to listen until an event is received, timeout, or `lcd_ClearScreen` is executed. Events returned to MessageCallback: DeviceState.InputEvent Data Format: {Event Type 1 byte}{Event Graphic ID 4 bytes}{Event Specific data variable bytes}

- Event Type 0 = Button Event
- Event Type 0 Data = {01h = Pressed}{Null Terminated Caption}
- Event Type 1 = Checkbox Event
- Event Type 1 Data = {01h = Checked, 00h = Unchecked}
- Event Type 2 = Line Item Event
- Event Type 2 Data = {01h = Selected}{Null Terminated Caption}
- Event Type 3 = Keypad Event
- Event Type 3 Data = {01h = Pressed, 02h = Released}{0030h = KEYPAD_0, 0031h = KEYPAD_1, 0032h = KEYPAD_2, 0033h = KEYPAD_3, 0034h = KEYPAD_4, 0035h = KEYPAD_5, 0036h = KEYPAD_6, 0037h = KEYPAD_7, 0038h = KEYPAD_8, 0039h = KEYPAD_9, 000Dh = KEYPAD_ENTER, 0008h = KEYPAD_CLEAR, 001Bh = KEYPAD_CANCEL }
- Event Type 4 = Touchscreen Event
- Event Type 4 Data = {00h = RFU}{Null Terminated Image Name}
- Event Type 5 = Slideshow Event

- Event Type 5 Data = {00h = RFU}
- Event Type 6 = Transaction Event
- Event Type 6 Data = {00h = RFU}{3 bytes card type}{00000000h = Success, 00000008h = Timeout, 0000000Ah = Failed}
- Event Type 7 = Radio Button Event
- Event Type 7 Data = {00h = RFU}{31 bytes null terminated group name}{31 bytes radio button caption null terminated}

Parameters

<i>timeout</i>	Timeout to wait for event.
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with `errorCode.getErrorString(string ident = "")`

21.21.1.145 `RETURN_CODE IDTechSDK.IDT_VP8800.lcd_resetInitialState (string ident = " ")`

Reset to Initial State

This command places the reader UI into the idle state and displays the appropriate idle display.

Parameters

<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.
--------------	--

Returns

RETURN_CODE: Values can be parsed with `errorCode.getErrorString(string ident = "")`

21.21.1.146 `static void IDTechSDK.IDT_VP8800.lcd_retrieveMessage (DisplayMessages.DISPLAY_MESSAGE_LANGUAGE lang, DisplayMessages.DISPLAY_MESSAGE_IDENTIFIER id, ref string line1, ref string line2) [static]`

Retrieve LCD Message

Returns the string value for a message ID returned for LCD messaging

Parameters

<i>lang</i>	Language.
<i>id</i>	Message ID
<i>line1</i>	Line 1 string value
<i>line2</i>	Line 2 string value
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with `device_getResponseCodeString`

21.21.1.147 **RETURN_CODE** IDTechSDK.IDT_VP8800.lcd_setBackgroundImage (*string file*, *bool enable*, *string ident* = " ")

Set Background Image

You must send images to the reader's memory and send a Start Custom Mode command to the reader before it will respond to Image commands. Image files must be in .bmp or .png format.

Parameters

<i>file</i>	Complete path and file name of the file you want to use. Example "file.png" will put in root directory, while "ss/file.png" will put in ss directory (which must exist, string ident = "")
<i>enable</i>	TRUE = Use Background Image, FALSE = Use Background Color
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with `errorCode.getErrorString(string ident = "")`

21.21.1.148 **RETURN_CODE** IDTechSDK.IDT_VP8800.lcd_setDisplayImage (*string file*, *int posX*, *int posY*, *int posMode*, *bool touchEnable*, *bool clearScreen*, *string ident* = " ")

Set Display Image

You must send images to the reader's memory and send a Start Custom Mode command to the reader before it will respond to Image commands. Image files must be in .bmp or .png format.

Parameters

<i>files</i>	Complete path and file name of the file you want to use. Example "file.png" will put in root directory, while "ss/file.png" will put in ss directory (which must exist, string ident = "")
<i>posX</i>	X coordinate in pixels, Range 0-271
<i>posY</i>	Y coordinate in pixels, Range 0-479
<i>posMode</i>	Position Mode <ul style="list-style-type: none"> • 0 = Center on Line Y • 1 = Display at (X,Y, string ident = "") • 2 - Center on screen
<i>touchEnable</i>	TRUE = Image is touch sensitive
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with `errorCode.getErrorString(string ident = "")`

21.21.1.149 **RETURN_CODE** IDTechSDK.IDT_VP8800.lcd_setForeBackColor (*byte[] foreRGB*, *byte[] backRGB*, *string ident* = " ")

Set Foreground and Background Color

This command sets the foreground and background colors of the LCD.

Parameters

<i>foreRGB</i>	Foreground RGB. 000000 = black, FF0000 = red, 00FF00 = green, 0000FF = blue, FFFFFFFF = white
<i>backRGB</i>	Background RGB. 000000 = black, FF0000 = red, 00FF00 = green, 0000FF = blue, FFFFFFFF = white
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with `errorCode.getErrorString(string ident = "")`

21.21.1.150 RETURN_CODE IDTechSDK.IDT_VP8800.lcd_startSlideShow (*string files*, *int posX*, *int posY*, *int posMode*, *bool touchEnable*, *bool recursion*, *bool touchTerminate*, *int delay*, *int loops*, *bool clearScreen*, *string ident = " "*)

Start slide show

You must send images to the reader's memory and send a Start Custom Mode command to the reader before it will respond to this commands. Image files must be in .bmp or .png format.

Parameters

<i>files</i>	Complete paths and file names of the files you want to use, separated by commas. If a directory is specified, all files in the directory are displayed
<i>posX</i>	X coordinate in pixels, Range 0-271
<i>posY</i>	Y coordinate in pixels, Range 0-479
<i>posMode</i>	Position Mode <ul style="list-style-type: none"> • 0 = Center on Line Y • 1 = Display at (X,Y, string ident = "") • 2 - Center on screen
<i>touchEnable</i>	TRUE = Image is touch sensitive
<i>recursion</i>	TRUE = Recursively follow directories in list
<i>touchTerminate</i>	TRUE = Terminate slideshow on touch (if touch enabled, string ident = "")
<i>delay</i>	Number of seconds between image displays
<i>loops</i>	Number of display loops. A zero indicates continuous display.
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with `errorCode.getErrorString(string ident = "")`

21.21.1.151 RETURN_CODE IDTechSDK.IDT_VP8800.msr_cancelMSRSwipe (*string ident = " "*)

Disable MSR Swipe Cancels MSR swipe request.

Parameters

<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.
--------------	--

Returns

RETURN_CODE: Values can be parsed with `errorCode.getErrorString(string ident = "")`

21.21.1.152 RETURN_CODE IDTechSDK.IDT_VP8800.msr_flushTrackData (string *ident* = " ")

Flush Track Data

Clears any track data being retained in memory by future PIN Block request.

Parameters

<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.
--------------	--

Returns

RETURN_CODE: Values can be parsed with `errorCode.getErrorString(string ident = "")`

21.21.1.153 RETURN_CODE IDTechSDK.IDT_VP8800.msr_startMSRSwipe (int *timeout*, string *ident* = " ")

Enable MSR Swipe

Enables MSR, waiting for swipe to occur.

Parameters

<i>timeout</i>	Swipe Timeout Value
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with `errorCode.getErrorString(string ident = "")`

21.21.1.154 RETURN_CODE IDTechSDK.IDT_VP8800.pin_getEncryptedOnlinePIN (int *keyType*, int *timeout*, string *ident* = " ")

Get Encrypted DUKPT PIN

Requests PIN Entry for online authorization. PIN block and KSN returned in callback function `DeviceState`.↔
TransactionData with `cardData.pin_pinblock`. A swipe must be captured first before this function can execute

Parameters

<i>keyType</i>	PIN block key type. Valid values 0,3 for TDES, 4 for AES
<i>timeout</i>	PIN entry timeout
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with `errorCode.getErrorString(string ident = "")`

21.21.1.155 `RETURN_CODE IDTechSDK.IDT_VP8800.pin_getPAN (bool getCSC, int timeout, string ident = " ")`

Get PAN

Requests PAN Entry on pinpad

Parameters

<i>getCSC</i>	Include Customer Service Code (also known as CVV, CVC, string <i>ident</i> = "")
<i>timeout</i>	PAN entry timeout
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with `errorCode.getErrorString(string ident = "")`

21.21.1.156 `RETURN_CODE IDTechSDK.IDT_VP8800.pin_promptCreditDebit (object currencySymbol, string displayAmount, int timeout, string ident = " ")`

Prompt for Credit or Debit

Requests prompt for Credit or Debit. Response returned in callback function as `DeviceState.MenuItem` with data `MENU_SELECTION_CREDIT = 0`, `MENU_SELECTION_DEBIT = 1`

Parameters

<i>currencySymbol</i>	Allowed values are \$ (0x24), ¥ (0xA5), £ (0xA3), € (0xA4), or NULL
<i>displayAmount</i>	Amount to display (can be NULL, string <i>ident</i> = "")
<i>timeout</i>	Menu entry timeout. Valid values 2-20 seconds
<i>ident</i>	Device ID to send command to. If not specified, current SDK default device will be used.

Returns

RETURN_CODE: Values can be parsed with `errorCode.getErrorString(string ident = "")`

21.21.1.157 `static String IDTechSDK.IDT_VP8800.SDK_Version () [static]`

SDK Version

- All Devices

Returns the current version of SDK

Returns

Framework version

21.21.1.158 `static void IDTechSDK.IDT_VP8800.setCallback (Callback my_Callback) [static]`

Set Callback

Sets the class callback

21.21.1.159 `static void IDTechSDK.IDT_VP8800.setCallback (IntPtr my_Callback, SynchronizationContext context)`
`[static]`

Set Callback

Sets the class callback

Parameters

<i>my_Callback</i>	The callback function to receive the response message from device. defined as follows. public unsafe delegate void MFCCallBack(Parameters parameters, ident);
<i>context</i>	The context of the UI thread

21.21.1.160 `static void IDTechSDK.IDT_VP8800.setCommandTimeout (int milliseconds, string ident = " ")` `[static]`

Set Command Timeout

Sets the default timeout (in milliseconds) waiting for a blocking command response

Parameters

<i>milliseconds</i>	Time
---------------------	------

21.21.1.161 `static bool IDTechSDK.IDT_VP8800.useSerialPort (int port)` `[static]`

Use Serial Port Interface

Instructs SDK to attempt to use the Serial Port for communication with VP8800 using default baud rate

Parameters

<i>port</i>	Serial Port to use. Example COM1 = 1.
-------------	---------------------------------------

Returns

bool TRUE=successful, FALSE=failure

21.21.1.162 `static bool IDTechSDK.IDT_VP8800.useSerialPort (int port, int baud, string ident = " ")` `[static]`

Use Serial Port Interface with baud rate

Instructs SDK to attempt to use the Serial Port for communication with VP8800

Parameters

<i>port</i>	Serial Port to use. Example COM1 = 1.
<i>baud</i>	Baud rate to override default. Example 115200;

Returns

bool TRUE=successful, FALSE=failure

21.21.1.163 `static bool IDTechSDK.IDT_VP8800.useSerialPortLinux (string path) [static]`

Use Serial Port Interface on Linux

Instructs SDK to attempt to use the Serial Port for communication with BTMag using default baud rate on Linux implementations

Parameters

<i>path</i>	Path to use. Example /dev/ttyUSB*
-------------	-----------------------------------

Returns

bool TRUE=successful, FALSE=failure

21.21.1.164 `static bool IDTechSDK.IDT_VP8800.useSerialPortLinux (string path, int baud) [static]`

Use Serial Port Interface on Linux with baud rate

Instructs SDK to attempt to use the Serial Port for communication with BTPay on Linux implementations

Parameters

<i>path</i>	Path to use. Example /dev/ttyUSB*
<i>baud</i>	Baud rate to override default. Example 115200;

Returns

bool TRUE=successful, FALSE=failure

21.21.2 Property Documentation

21.21.2.1 `IDT_VP8800 IDTechSDK.IDT_VP8800.SharedController [static],[get]`

Singleton Instance

Establishes an singleton instance of [IDT_VP8800](#) class.

Returns

Instance of [IDT_VP8800](#)

The documentation for this class was generated from the following file:

- /Users/randy/Repo/universal-sdk/IDTechSDK_STD/DeviceHeaders/IDT_VP8800.cs

Index

adf_ApplicationControl
 IDTechSDK::IDT_NEO2, [390](#)

adf_eraseFlash
 IDTechSDK::IDT_NEO2, [391](#)

adf_getADFMode
 IDTechSDK::IDT_NEO2, [391](#)

adf_getModuleBytes
 IDTechSDK::IDT_NEO2, [391](#)

adf_getModuleInfo
 IDTechSDK::IDT_NEO2, [391](#)

adf_setADFMode
 IDTechSDK::IDT_NEO2, [392](#)

adf_setJTAG
 IDTechSDK::IDT_NEO2, [392](#)

closeSerialPort
 IDTechSDK::IDT_K100, [235](#)
 IDTechSDK::IDT_L100, [289](#)
 IDTechSDK::IDT_L80, [320](#)

closeSocket
 IDTechSDK::IDT_NEO2, [392](#)

closeUSB
 IDTechSDK::IDT_K100, [235](#)
 IDTechSDK::IDT_L100, [289](#)
 IDTechSDK::IDT_L80, [320](#)

config_checkDUKPTKey
 IDTechSDK::IDT_KioskIII, [250](#)
 IDTechSDK::IDT_NEO2, [393](#)

config_getBaudRate
 IDTechSDK::IDT_K100, [235](#)
 IDTechSDK::IDT_L100, [289](#)
 IDTechSDK::IDT_L80, [320](#)

config_getBeeperController
 IDTechSDK::IDT_Augusta, [84](#)
 IDTechSDK::IDT_BTPay, [177](#)

config_getDEKVariantType
 IDTechSDK::IDT_KioskIII, [250](#)
 IDTechSDK::IDT_NEO2, [393](#)

config_getDUKPT_DEK_Attribution
 IDTechSDK::IDT_KioskIII, [251](#)
 IDTechSDK::IDT_NEO2, [394](#)

config_getDUKPT_KSN
 IDTechSDK::IDT_KioskIII, [251](#)
 IDTechSDK::IDT_NEO2, [394](#)

config_getDUKPTEncryptionType
 IDTechSDK::IDT_KioskIII, [251](#)
 IDTechSDK::IDT_NEO2, [394](#)

config_getDataKeySlot
 IDTechSDK::IDT_VP8800, [879](#)

config_getDate
 IDTechSDK::IDT_VP3300, [817](#)

config_getEncryptionControl
 IDTechSDK::IDT_Augusta, [84](#)
 IDTechSDK::IDT_BTMag, [142](#)
 IDTechSDK::IDT_BTPay, [177](#)
 IDTechSDK::IDT_NEO2, [395](#)
 IDTechSDK::IDT_UniPayI_V, [716](#)
 IDTechSDK::IDT_VP3300, [818](#)

config_getEthernetMACAddress
 IDTechSDK::IDT_L100, [290](#)
 IDTechSDK::IDT_NEO2, [395](#)
 IDTechSDK::IDT_VP8800, [880](#)

config_getKeySlotInfo
 IDTechSDK::IDT_NEO2, [396](#)
 IDTechSDK::IDT_VP8800, [880](#)

config_getKeyslot_PEK_DEK
 IDTechSDK::IDT_KioskIII, [252](#)
 IDTechSDK::IDT_NEO2, [396](#)

config_getLEDController
 IDTechSDK::IDT_Augusta, [85](#)
 IDTechSDK::IDT_BTMag, [143](#)
 IDTechSDK::IDT_BTPay, [178](#)

config_getMasking
 IDTechSDK::IDT_NEO2, [396](#)

config_getModelNumber
 IDTechSDK::IDT_Augusta, [85](#)
 IDTechSDK::IDT_BTPay, [178](#)
 IDTechSDK::IDT_K100, [236](#)
 IDTechSDK::IDT_L100, [290](#)
 IDTechSDK::IDT_L80, [321](#)
 IDTechSDK::IDT_MiniSmartII, [342](#)
 IDTechSDK::IDT_SREDKey2, [661](#)
 IDTechSDK::IDT_SecureKey, [565](#)
 IDTechSDK::IDT_SecureMag, [588](#)
 IDTechSDK::IDT_SpectrumPro, [615](#)
 IDTechSDK::IDT_UniPay, [697](#)

config_getNetworkConfiguration
 IDTechSDK::IDT_L100, [290](#)
 IDTechSDK::IDT_NEO2, [397](#)
 IDTechSDK::IDT_VP8800, [880](#)

config_getPINKeySlot
 IDTechSDK::IDT_VP8800, [881](#)

config_getSSLServerEthernet
 IDTechSDK::IDT_NEO2, [398](#)

config_getSalt_KCV
 IDTechSDK::IDT_KioskIII, [252](#)
 IDTechSDK::IDT_NEO2, [397](#)

config_getSerialNumber
 IDTechSDK::IDT_Augusta, [86](#)

- IDTechSDK::IDT_BTMag, [143](#)
- IDTechSDK::IDT_BTPay, [178](#)
- IDTechSDK::IDT_KioskIII, [252](#)
- IDTechSDK::IDT_L100, [291](#)
- IDTechSDK::IDT_L80, [321](#)
- IDTechSDK::IDT_MiniSmartII, [342](#)
- IDTechSDK::IDT_NEO2, [398](#)
- IDTechSDK::IDT_PIP, [538](#)
- IDTechSDK::IDT_SREDKey2, [661](#)
- IDTechSDK::IDT_SecureKey, [565](#)
- IDTechSDK::IDT_SecureMag, [588](#)
- IDTechSDK::IDT_SpectrumPro, [615](#)
- IDTechSDK::IDT_UniPay, [697](#)
- IDTechSDK::IDT_UniPayI_V, [717](#)
- IDTechSDK::IDT_VP3300, [818](#)
- IDTechSDK::IDT_VP8800, [881](#)
- IDTechSDK::IDT_VendIII, [787](#)
- IDTechSDK::IDT_Vendi, [759](#)
- config_getStatusKeySlots
 - IDTechSDK::IDT_SREDKey2, [661](#)
 - IDTechSDK::IDT_VP8800, [881](#)
- config_getSwipeandDone
 - IDTechSDK::IDT_NEO2, [398](#)
- config_getTime
 - IDTechSDK::IDT_VP3300, [819](#)
- config_getTrackFormat
 - IDTechSDK::IDT_NEO2, [399](#)
- config_getWhiteList
 - IDTechSDK::IDT_NEO2, [399](#)
- config_getWifiConfig
 - IDTechSDK::IDT_NEO2, [399](#)
- config_getWirelessWorkMode
 - IDTechSDK::IDT_NEO2, [400](#)
- config_sendSSLRequestWiFi
 - IDTechSDK::IDT_NEO2, [400](#)
- config_setBaudRate
 - IDTechSDK::IDT_K100, [236](#)
 - IDTechSDK::IDT_KioskIII, [253](#)
 - IDTechSDK::IDT_L100, [291](#)
 - IDTechSDK::IDT_L80, [321](#)
 - IDTechSDK::IDT_NEO2, [400](#)
 - IDTechSDK::IDT_VendIII, [788](#)
 - IDTechSDK::IDT_Vendi, [759](#)
- config_setBeeperController
 - IDTechSDK::IDT_Augusta, [86](#)
 - IDTechSDK::IDT_BTPay, [179](#)
- config_setBluetoothParameters
 - IDTechSDK::IDT_NEO2, [401](#)
- config_setCmdTimeOutDuration
 - IDTechSDK::IDT_Augusta, [86](#)
 - IDTechSDK::IDT_BTMag, [143](#)
 - IDTechSDK::IDT_BTPay, [179](#)
 - IDTechSDK::IDT_K100, [236](#)
 - IDTechSDK::IDT_L100, [291](#)
 - IDTechSDK::IDT_L80, [322](#)
 - IDTechSDK::IDT_MiniSmartII, [342](#)
 - IDTechSDK::IDT_UniPay, [697](#)
- config_setDEKVariantType
 - IDTechSDK::IDT_KioskIII, [253](#)
 - IDTechSDK::IDT_NEO2, [401](#)
- config_setDUKPT_DEK_Attribution_AES
 - IDTechSDK::IDT_KioskIII, [254](#)
 - IDTechSDK::IDT_NEO2, [401](#)
- config_setDUKPT_DEK_Attribution_TDES
 - IDTechSDK::IDT_KioskIII, [254](#)
 - IDTechSDK::IDT_NEO2, [402](#)
- config_setDUKPTEncryptionType
 - IDTechSDK::IDT_KioskIII, [255](#)
 - IDTechSDK::IDT_NEO2, [402](#)
- config_setDataKeySlot
 - IDTechSDK::IDT_VP8800, [882](#)
- config_setDate
 - IDTechSDK::IDT_VP3300, [819](#)
- config_setEncryptionControl
 - IDTechSDK::IDT_Augusta, [86](#)
 - IDTechSDK::IDT_BTMag, [144](#)
 - IDTechSDK::IDT_BTPay, [179](#)
 - IDTechSDK::IDT_NEO2, [403](#)
 - IDTechSDK::IDT_UniPayI_V, [717](#)
 - IDTechSDK::IDT_VP3300, [819](#)
- config_setEthernetMACAddress
 - IDTechSDK::IDT_L100, [292](#)
 - IDTechSDK::IDT_VP8800, [882](#)
- config_setKeyslot_PEK_DEK
 - IDTechSDK::IDT_KioskIII, [255](#)
 - IDTechSDK::IDT_NEO2, [403](#)
- config_setLEDController
 - IDTechSDK::IDT_Augusta, [87](#)
 - IDTechSDK::IDT_BTMag, [144](#)
 - IDTechSDK::IDT_BTPay, [180](#)
- config_setMasking
 - IDTechSDK::IDT_NEO2, [404](#)
- config_setNetworkConfiguration
 - IDTechSDK::IDT_L100, [292](#)
 - IDTechSDK::IDT_NEO2, [404](#)
 - IDTechSDK::IDT_VP8800, [882](#)
- config_setPINKeySlot
 - IDTechSDK::IDT_VP8800, [883](#)
- config_setRKLLKeys
 - IDTechSDK::IDT_KioskIII, [255](#)
 - IDTechSDK::IDT_NEO2, [404](#)
- config_setSSLServerEthernet
 - IDTechSDK::IDT_NEO2, [405](#)
- config_setSwipeandDone
 - IDTechSDK::IDT_NEO2, [405](#)
- config_setTrackFormat
 - IDTechSDK::IDT_NEO2, [406](#)
- config_setWhiteList
 - IDTechSDK::IDT_NEO2, [406](#)
- config_setWifiConfig
 - IDTechSDK::IDT_NEO2, [406](#)
- config_setWirelessWorkMode
 - IDTechSDK::IDT_NEO2, [407](#)
- createFastEMVData
 - IDTechSDK::IDT_Augusta, [87](#)
 - IDTechSDK::IDT_MiniSmartII, [342](#)

- IDTechSDK::IDT_NEO2, [407](#)
- IDTechSDK::IDT_SpectrumPro, [615](#)
- IDTechSDK::IDT_VP3300, [820](#)
- IDTechSDK::IDT_VP8800, [883](#)
- ctls_activateTransaction
 - IDTechSDK::IDT_KioskIII, [256](#)
 - IDTechSDK::IDT_NEO2, [407](#)
 - IDTechSDK::IDT_PIP, [539](#)
 - IDTechSDK::IDT_VP3300, [820](#)
 - IDTechSDK::IDT_VP8800, [883](#)
 - IDTechSDK::IDT_VendIII, [788](#)
 - IDTechSDK::IDT_Vendi, [760](#)
- ctls_cancelTransaction
 - IDTechSDK::IDT_CM100, [226](#)
 - IDTechSDK::IDT_KioskIII, [257](#)
 - IDTechSDK::IDT_NEO2, [409](#)
 - IDTechSDK::IDT_PIP, [540](#)
 - IDTechSDK::IDT_VP3300, [822](#)
 - IDTechSDK::IDT_VP8800, [884](#)
 - IDTechSDK::IDT_VendIII, [789](#)
 - IDTechSDK::IDT_Vendi, [761](#)
- ctls_displayOnlineAuthResult
 - IDTechSDK::IDT_VP8800, [885](#)
- ctls_displayOnlineAuthResult_ext
 - IDTechSDK::IDT_VP8800, [885](#)
- ctls_enableL2Application
 - IDTechSDK::IDT_CM100, [227](#)
- ctls_getAllConfigurationGroups
 - IDTechSDK::IDT_KioskIII, [257](#)
 - IDTechSDK::IDT_NEO2, [409](#)
 - IDTechSDK::IDT_PIP, [540](#)
 - IDTechSDK::IDT_VP3300, [822](#)
 - IDTechSDK::IDT_VendIII, [790](#)
 - IDTechSDK::IDT_Vendi, [761](#)
- ctls_getConfigurationGroup
 - IDTechSDK::IDT_KioskIII, [258](#)
 - IDTechSDK::IDT_NEO2, [409](#)
 - IDTechSDK::IDT_PIP, [540](#)
 - IDTechSDK::IDT_VP3300, [822](#)
 - IDTechSDK::IDT_VP8800, [885](#)
 - IDTechSDK::IDT_VendIII, [790](#)
 - IDTechSDK::IDT_Vendi, [762](#)
- ctls_nfcCommand
 - IDTechSDK::IDT_KioskIII, [258](#)
 - IDTechSDK::IDT_NEO2, [410](#)
 - IDTechSDK::IDT_PIP, [541](#)
 - IDTechSDK::IDT_VP8800, [886](#)
- ctls_removeAICAPK
 - IDTechSDK::IDT_KioskIII, [259](#)
 - IDTechSDK::IDT_NEO2, [411](#)
 - IDTechSDK::IDT_VP3300, [823](#)
 - IDTechSDK::IDT_VP8800, [887](#)
 - IDTechSDK::IDT_VendIII, [790](#)
 - IDTechSDK::IDT_Vendi, [762](#)
- ctls_removeApplicationData
 - IDTechSDK::IDT_KioskIII, [260](#)
 - IDTechSDK::IDT_NEO2, [411](#)
 - IDTechSDK::IDT_PIP, [542](#)
- IDTechSDK::IDT_VP3300, [823](#)
- IDTechSDK::IDT_VP8800, [888](#)
- IDTechSDK::IDT_VendIII, [791](#)
- IDTechSDK::IDT_Vendi, [762](#)
- ctls_removeCAPK
 - IDTechSDK::IDT_KioskIII, [260](#)
 - IDTechSDK::IDT_NEO2, [411](#)
 - IDTechSDK::IDT_VP3300, [823](#)
 - IDTechSDK::IDT_VP8800, [888](#)
 - IDTechSDK::IDT_VendIII, [791](#)
 - IDTechSDK::IDT_Vendi, [762](#)
- ctls_removeConfigurationGroup
 - IDTechSDK::IDT_KioskIII, [260](#)
 - IDTechSDK::IDT_NEO2, [411](#)
 - IDTechSDK::IDT_PIP, [543](#)
 - IDTechSDK::IDT_VP3300, [823](#)
 - IDTechSDK::IDT_VP8800, [888](#)
 - IDTechSDK::IDT_VendIII, [791](#)
 - IDTechSDK::IDT_Vendi, [763](#)
- ctls_resetConfigurationGroup
 - IDTechSDK::IDT_NEO2, [412](#)
 - IDTechSDK::IDT_VP8800, [888](#)
- ctls_retrieveAIDList
 - IDTechSDK::IDT_KioskIII, [260](#)
 - IDTechSDK::IDT_NEO2, [412](#)
 - IDTechSDK::IDT_PIP, [543](#)
 - IDTechSDK::IDT_VP3300, [824](#)
 - IDTechSDK::IDT_VP8800, [889](#)
 - IDTechSDK::IDT_VendIII, [791](#)
 - IDTechSDK::IDT_Vendi, [763](#)
- ctls_retrieveApplicationData
 - IDTechSDK::IDT_KioskIII, [261](#)
 - IDTechSDK::IDT_NEO2, [412](#)
 - IDTechSDK::IDT_PIP, [543](#)
 - IDTechSDK::IDT_VP3300, [824](#)
 - IDTechSDK::IDT_VP8800, [889](#)
 - IDTechSDK::IDT_VendIII, [792](#)
 - IDTechSDK::IDT_Vendi, [763](#)
- ctls_retrieveCAPKList
 - IDTechSDK::IDT_KioskIII, [262](#)
 - IDTechSDK::IDT_NEO2, [413](#)
 - IDTechSDK::IDT_VP3300, [825](#)
 - IDTechSDK::IDT_VP8800, [890](#)
 - IDTechSDK::IDT_VendIII, [793](#)
 - IDTechSDK::IDT_Vendi, [764](#)
- ctls_retrieveCAPK
 - IDTechSDK::IDT_KioskIII, [261](#)
 - IDTechSDK::IDT_NEO2, [413](#)
 - IDTechSDK::IDT_VP3300, [824](#)
 - IDTechSDK::IDT_VP8800, [889](#)
 - IDTechSDK::IDT_VendIII, [792](#)
 - IDTechSDK::IDT_Vendi, [764](#)
- ctls_retrieveTerminalData
 - IDTechSDK::IDT_KioskIII, [262](#)
 - IDTechSDK::IDT_NEO2, [413](#)
 - IDTechSDK::IDT_PIP, [544](#)
 - IDTechSDK::IDT_VP3300, [825](#)
 - IDTechSDK::IDT_VP8800, [890](#)

- IDTechSDK::IDT_VendIII, 793
- IDTechSDK::IDT_Vendi, 764
- ctls_setApplicationData
 - IDTechSDK::IDT_KioskIII, 263
 - IDTechSDK::IDT_NEO2, 414
 - IDTechSDK::IDT_PIP, 544
 - IDTechSDK::IDT_VP3300, 826
 - IDTechSDK::IDT_VP8800, 891
 - IDTechSDK::IDT_VendIII, 794
 - IDTechSDK::IDT_Vendi, 765
- ctls_setCAPK
 - IDTechSDK::IDT_KioskIII, 263
 - IDTechSDK::IDT_NEO2, 414
 - IDTechSDK::IDT_VP3300, 826
 - IDTechSDK::IDT_VP8800, 891
 - IDTechSDK::IDT_VendIII, 794
 - IDTechSDK::IDT_Vendi, 765
- ctls_setConfigurationGroup
 - IDTechSDK::IDT_KioskIII, 263
 - IDTechSDK::IDT_NEO2, 415
 - IDTechSDK::IDT_PIP, 544
 - IDTechSDK::IDT_VP3300, 827
 - IDTechSDK::IDT_VP8800, 892
 - IDTechSDK::IDT_VendIII, 795
 - IDTechSDK::IDT_Vendi, 766
- ctls_setDefaultConfiguration
 - IDTechSDK::IDT_KioskIII, 264
 - IDTechSDK::IDT_NEO2, 415
 - IDTechSDK::IDT_PIP, 545
 - IDTechSDK::IDT_VP3300, 827
 - IDTechSDK::IDT_VP8800, 892
 - IDTechSDK::IDT_VendIII, 795
 - IDTechSDK::IDT_Vendi, 766
- ctls_setTerminalData
 - IDTechSDK::IDT_KioskIII, 264
 - IDTechSDK::IDT_NEO2, 416
 - IDTechSDK::IDT_PIP, 545
 - IDTechSDK::IDT_VP3300, 827
 - IDTechSDK::IDT_VP8800, 893
 - IDTechSDK::IDT_VendIII, 795
 - IDTechSDK::IDT_Vendi, 767
- ctls_startTransaction
 - IDTechSDK::IDT_CM100, 227
 - IDTechSDK::IDT_KioskIII, 264
 - IDTechSDK::IDT_NEO2, 416
 - IDTechSDK::IDT_PIP, 545
 - IDTechSDK::IDT_VP3300, 827
 - IDTechSDK::IDT_VP8800, 893
 - IDTechSDK::IDT_VendIII, 795
 - IDTechSDK::IDT_Vendi, 767
- ctls_trySetTerminalData
 - IDTechSDK::IDT_KioskIII, 266
 - IDTechSDK::IDT_NEO2, 417
 - IDTechSDK::IDT_PIP, 546
 - IDTechSDK::IDT_VP3300, 829
 - IDTechSDK::IDT_VP8800, 894
 - IDTechSDK::IDT_VendIII, 797
 - IDTechSDK::IDT_Vendi, 768
- ctls_updateBalance
 - IDTechSDK::IDT_KioskIII, 266
 - IDTechSDK::IDT_NEO2, 418
 - IDTechSDK::IDT_PIP, 547
- device_PKI_RKI
 - IDTechSDK::IDT_L80, 324
- device_RemoteKeyInjection
 - IDTechSDK::IDT_Augusta, 105
 - IDTechSDK::IDT_BTMag, 159
 - IDTechSDK::IDT_BTPay, 195
 - IDTechSDK::IDT_CM100, 229
 - IDTechSDK::IDT_K100, 240
 - IDTechSDK::IDT_KioskIII, 273
 - IDTechSDK::IDT_L100, 296
 - IDTechSDK::IDT_L80, 325
 - IDTechSDK::IDT_MiniSmartII, 357
 - IDTechSDK::IDT_NEO2, 451
 - IDTechSDK::IDT_PIP, 553
 - IDTechSDK::IDT_SREDKey2, 675
 - IDTechSDK::IDT_SecureKey, 578
 - IDTechSDK::IDT_SecureMag, 600
 - IDTechSDK::IDT_SpectrumPro, 631
 - IDTechSDK::IDT_UniPay, 701
 - IDTechSDK::IDT_UniPayI_V, 732
 - IDTechSDK::IDT_VP3300, 850
 - IDTechSDK::IDT_VP8800, 913
 - IDTechSDK::IDT_VendIII, 801
 - IDTechSDK::IDT_Vendi, 774
- device_StartRKI
 - IDTechSDK::IDT_Augusta, 110
 - IDTechSDK::IDT_K100, 242
 - IDTechSDK::IDT_KioskIII, 279
 - IDTechSDK::IDT_L100, 300
 - IDTechSDK::IDT_MiniSmartII, 360
 - IDTechSDK::IDT_NEO2, 463
 - IDTechSDK::IDT_SecureKey, 583
 - IDTechSDK::IDT_SecureMag, 603, 604
 - IDTechSDK::IDT_SpectrumPro, 636
 - IDTechSDK::IDT_UniPayI_V, 738
 - IDTechSDK::IDT_VP3300, 856
 - IDTechSDK::IDT_VP8800, 920
- device_activateTransaction
 - IDTechSDK::IDT_Augusta, 88
 - IDTechSDK::IDT_KioskIII, 267
 - IDTechSDK::IDT_MiniSmartII, 343
 - IDTechSDK::IDT_NEO2, 418
 - IDTechSDK::IDT_PIP, 547
 - IDTechSDK::IDT_SpectrumPro, 616
 - IDTechSDK::IDT_UniPayI_V, 717
 - IDTechSDK::IDT_VP3300, 829
 - IDTechSDK::IDT_VP8800, 895
 - IDTechSDK::IDT_VendIII, 797
 - IDTechSDK::IDT_Vendi, 769
- device_buzzer
 - IDTechSDK::IDT_NEO2, 419
 - IDTechSDK::IDT_VP3300, 830
 - IDTechSDK::IDT_VP8800, 896
- device_buzzerOnOff

- IDTechSDK::IDT_NEO2, [420](#)
- device_calibrateParameters
 - IDTechSDK::IDT_VP8800, [896](#)
- device_cancelTransaction
 - IDTechSDK::IDT_KioskIII, [268](#)
 - IDTechSDK::IDT_NEO2, [420](#)
 - IDTechSDK::IDT_VP3300, [831](#)
 - IDTechSDK::IDT_VP8800, [896](#)
- device_capturePINFromLast12
 - IDTechSDK::IDT_SecureKey, [565](#)
- device_cardNotification
 - IDTechSDK::IDT_SpectrumPro, [616](#)
- device_certificateType
 - IDTechSDK::IDT_NEO2, [420](#)
 - IDTechSDK::IDT_SREDKey2, [662](#)
- device_controlBeep
 - IDTechSDK::IDT_Augusta, [88](#)
 - IDTechSDK::IDT_BTMag, [145](#)
 - IDTechSDK::IDT_BTPay, [180](#)
- device_controlIndicator
 - IDTechSDK::IDT_VP8800, [896](#)
- device_controlLED_ICC
 - IDTechSDK::IDT_Augusta, [89](#)
 - IDTechSDK::IDT_BTMag, [146](#)
 - IDTechSDK::IDT_BTPay, [181](#)
- device_controlLED_MSR
 - IDTechSDK::IDT_Augusta, [89](#)
 - IDTechSDK::IDT_BTMag, [146](#)
 - IDTechSDK::IDT_BTPay, [182](#)
- device_controlLED
 - IDTechSDK::IDT_Augusta, [88](#)
 - IDTechSDK::IDT_BTMag, [145](#)
 - IDTechSDK::IDT_BTPay, [181](#)
 - IDTechSDK::IDT_KioskIII, [268](#)
 - IDTechSDK::IDT_NEO2, [420](#)
 - IDTechSDK::IDT_VP3300, [831](#)
 - IDTechSDK::IDT_VP8800, [897](#)
 - IDTechSDK::IDT_VendIII, [798](#)
 - IDTechSDK::IDT_Vendi, [770](#)
- device_controlUserInterface
 - IDTechSDK::IDT_KioskIII, [269](#)
 - IDTechSDK::IDT_NEO2, [421](#)
 - IDTechSDK::IDT_PIP, [548](#)
 - IDTechSDK::IDT_SpectrumPro, [616](#)
 - IDTechSDK::IDT_VP3300, [832](#)
 - IDTechSDK::IDT_VP8800, [897](#)
 - IDTechSDK::IDT_VendIII, [798](#)
 - IDTechSDK::IDT_Vendi, [770](#)
- device_createDirectory
 - IDTechSDK::IDT_VP8800, [898](#)
- device_deleteDirectory
 - IDTechSDK::IDT_NEO2, [422](#)
 - IDTechSDK::IDT_VP8800, [899](#)
- device_deleteFile
 - IDTechSDK::IDT_NEO2, [423](#)
 - IDTechSDK::IDT_VP8800, [899](#)
- device_disBlueLED
 - IDTechSDK::IDT_NEO2, [423](#)
- device_disableAugustaLED
 - IDTechSDK::IDT_Augusta, [90](#)
- device_enaBlueLED
 - IDTechSDK::IDT_NEO2, [424](#)
- device_enableAdminKey
 - IDTechSDK::IDT_SREDKey2, [662](#)
- device_enableL100PassThrough
 - IDTechSDK::IDT_NEO2, [424](#)
- device_enableL80PassThrough
 - IDTechSDK::IDT_NEO2, [424](#)
- device_enablePassThrough
 - IDTechSDK::IDT_KioskIII, [269](#)
 - IDTechSDK::IDT_NEO2, [424](#)
 - IDTechSDK::IDT_PIP, [549](#)
 - IDTechSDK::IDT_UniPayI_V, [719](#)
 - IDTechSDK::IDT_VP3300, [833](#)
 - IDTechSDK::IDT_VP8800, [899](#)
- device_enableSecureHeadForMSII
 - IDTechSDK::IDT_MiniSmartII, [343](#)
- device_enterStandbyMode
 - IDTechSDK::IDT_NEO2, [425](#)
- device_enterStopMode
 - IDTechSDK::IDT_L100, [292](#)
 - IDTechSDK::IDT_L80, [322](#)
- device_extendedErrorCondition
 - IDTechSDK::IDT_NEO2, [425](#)
- device_get1050BootloaderVersion
 - IDTechSDK::IDT_NEO2, [425](#)
- device_get1050FuseStatus
 - IDTechSDK::IDT_NEO2, [426](#)
- device_getAnyRKIStatus
 - IDTechSDK::IDT_Augusta, [90](#)
 - IDTechSDK::IDT_BTMag, [147](#)
 - IDTechSDK::IDT_BTPay, [182](#)
 - IDTechSDK::IDT_CM100, [227](#)
 - IDTechSDK::IDT_K100, [237](#)
 - IDTechSDK::IDT_KioskIII, [270](#)
 - IDTechSDK::IDT_L100, [293](#)
 - IDTechSDK::IDT_L80, [322](#)
 - IDTechSDK::IDT_MiniSmartII, [344](#)
 - IDTechSDK::IDT_NEO2, [426](#)
 - IDTechSDK::IDT_PIP, [550](#)
 - IDTechSDK::IDT_SREDKey2, [662](#)
 - IDTechSDK::IDT_SecureKey, [565](#)
 - IDTechSDK::IDT_SecureMag, [588](#)
 - IDTechSDK::IDT_SpectrumPro, [617](#)
 - IDTechSDK::IDT_UniPay, [697](#)
 - IDTechSDK::IDT_UniPayI_V, [719](#)
 - IDTechSDK::IDT_VP3300, [834](#)
 - IDTechSDK::IDT_VendIII, [799](#)
 - IDTechSDK::IDT_Vendi, [771](#)
- device_getAudioVolume
 - IDTechSDK::IDT_NEO2, [426](#)
- device_getBLEName
 - IDTechSDK::IDT_NEO2, [427](#)
- device_getBatteryVoltage
 - IDTechSDK::IDT_NEO2, [427](#)
 - IDTechSDK::IDT_UniPay, [698](#)

- device_getBootloaderVersion
 - IDTechSDK::IDT_NEO2, [428](#)
 - IDTechSDK::IDT_UniPay, [698](#)
- device_getCashTranRiskPara
 - IDTechSDK::IDT_NEO2, [428](#)
 - IDTechSDK::IDT_VP3300, [834](#)
- device_getConfigurationFromMemory
 - IDTechSDK::IDT_Augusta, [91](#)
 - IDTechSDK::IDT_BTMag, [147](#)
 - IDTechSDK::IDT_BTPay, [182](#)
 - IDTechSDK::IDT_CM100, [228](#)
 - IDTechSDK::IDT_K100, [237](#)
 - IDTechSDK::IDT_KioskIII, [270](#)
 - IDTechSDK::IDT_L100, [293](#)
 - IDTechSDK::IDT_L80, [322](#)
 - IDTechSDK::IDT_MiniSmartII, [344](#)
 - IDTechSDK::IDT_NEO2, [428](#)
 - IDTechSDK::IDT_PIP, [550](#)
 - IDTechSDK::IDT_SREDKey2, [663](#)
 - IDTechSDK::IDT_SecureKey, [566](#)
 - IDTechSDK::IDT_SecureMag, [588](#)
 - IDTechSDK::IDT_SpectrumPro, [618](#)
 - IDTechSDK::IDT_UniPay, [699](#)
 - IDTechSDK::IDT_UniPayI_V, [719](#)
 - IDTechSDK::IDT_VP3300, [834](#)
 - IDTechSDK::IDT_VP8800, [900](#)
 - IDTechSDK::IDT_VendIII, [799](#)
 - IDTechSDK::IDT_Vendi, [771](#)
- device_getConfigurationGroup
 - IDTechSDK::IDT_VP8800, [900](#)
- device_getDRS
 - IDTechSDK::IDT_Augusta, [91](#)
 - IDTechSDK::IDT_NEO2, [429](#)
 - IDTechSDK::IDT_UniPayI_V, [720](#)
- device_getDateTime
 - IDTechSDK::IDT_Augusta, [91](#)
 - IDTechSDK::IDT_L100, [293](#)
 - IDTechSDK::IDT_L80, [323](#)
 - IDTechSDK::IDT_MiniSmartII, [344](#)
 - IDTechSDK::IDT_VP3300, [835](#)
- device_getDateTimeString
 - IDTechSDK::IDT_K100, [237](#)
- device_getDeviceTime
 - IDTechSDK::IDT_NEO2, [428](#)
- device_getDeviceTreeVersion
 - IDTechSDK::IDT_NEO2, [429](#)
- device_getDriveFreeSpace
 - IDTechSDK::IDT_VP8800, [900](#)
- device_getDriReaderRiskPara
 - IDTechSDK::IDT_VP3300, [835](#)
- device_getFirmwareVersion
 - IDTechSDK::IDT_Augusta, [92](#)
 - IDTechSDK::IDT_BTMag, [148](#)
 - IDTechSDK::IDT_BTPay, [183](#)
 - IDTechSDK::IDT_CM100, [228](#)
 - IDTechSDK::IDT_K100, [238](#)
 - IDTechSDK::IDT_KioskIII, [270](#)
 - IDTechSDK::IDT_L100, [294](#)
 - IDTechSDK::IDT_L80, [323](#)
 - IDTechSDK::IDT_MiniSmartII, [345](#)
 - IDTechSDK::IDT_NEO2, [429](#)
 - IDTechSDK::IDT_PIP, [550](#)
 - IDTechSDK::IDT_SREDKey2, [663](#)
 - IDTechSDK::IDT_SecureKey, [566](#)
 - IDTechSDK::IDT_SecureMag, [589](#)
 - IDTechSDK::IDT_SpectrumPro, [618](#)
 - IDTechSDK::IDT_UniPay, [699](#)
 - IDTechSDK::IDT_UniPayI_V, [720](#)
 - IDTechSDK::IDT_VP3300, [835](#)
 - IDTechSDK::IDT_VP8800, [901](#)
 - IDTechSDK::IDT_VendIII, [799](#)
 - IDTechSDK::IDT_Vendi, [772](#)
- device_getHardwareInfor
 - IDTechSDK::IDT_NEO2, [430](#)
 - IDTechSDK::IDT_VP3300, [836](#)
- device_getKeyFormatForICCDUKPT
 - IDTechSDK::IDT_Augusta, [92](#)
- device_getKeyStatus
 - IDTechSDK::IDT_Augusta, [92](#)
 - IDTechSDK::IDT_BTMag, [148](#)
 - IDTechSDK::IDT_BTPay, [183](#)
 - IDTechSDK::IDT_K100, [238](#)
 - IDTechSDK::IDT_L100, [294](#)
 - IDTechSDK::IDT_L80, [323](#)
- device_getKeyTypeForICCDUKPT
 - IDTechSDK::IDT_Augusta, [93](#)
 - IDTechSDK::IDT_MiniSmartII, [345](#)
- device_getLightSensorVal
 - IDTechSDK::IDT_NEO2, [430](#)
- device_getMerchantRecord
 - IDTechSDK::IDT_KioskIII, [270](#)
 - IDTechSDK::IDT_NEO2, [430](#)
 - IDTechSDK::IDT_PIP, [550](#)
 - IDTechSDK::IDT_VP3300, [836](#)
 - IDTechSDK::IDT_VP8800, [901](#)
 - IDTechSDK::IDT_VendIII, [800](#)
 - IDTechSDK::IDT_Vendi, [772](#)
- device_getModuleVer
 - IDTechSDK::IDT_NEO2, [431](#)
 - IDTechSDK::IDT_VP3300, [836](#)
- device_getMsrSecurePar
 - IDTechSDK::IDT_NEO2, [431](#)
- device_getNonce
 - IDTechSDK::IDT_SpectrumPro, [618](#)
- device_getOutputType
 - IDTechSDK::IDT_SREDKey2, [663](#)
 - IDTechSDK::IDT_SecureKey, [566](#)
- device_getPIPMODE
 - IDTechSDK::IDT_PIP, [551](#)
- device_getPollMode
 - IDTechSDK::IDT_VP3300, [837](#)
- device_getProcessorType
 - IDTechSDK::IDT_NEO2, [431](#)
- device_getProductType
 - IDTechSDK::IDT_NEO2, [432](#)
 - IDTechSDK::IDT_VP3300, [837](#)

- device_getQuickChipMode
 - IDTechSDK::IDT_Augusta, [93](#)
- device_getRKIStatus
 - IDTechSDK::IDT_Augusta, [103](#)
 - IDTechSDK::IDT_BTMag, [158](#)
 - IDTechSDK::IDT_BTPay, [193](#)
 - IDTechSDK::IDT_CM100, [228](#)
 - IDTechSDK::IDT_K100, [238](#)
 - IDTechSDK::IDT_KioskIII, [271](#)
 - IDTechSDK::IDT_L100, [294](#)
 - IDTechSDK::IDT_L80, [324](#)
 - IDTechSDK::IDT_MiniSmartII, [355](#)
 - IDTechSDK::IDT_NEO2, [442](#)
 - IDTechSDK::IDT_PIP, [551](#)
 - IDTechSDK::IDT_SREDKey2, [673](#)
 - IDTechSDK::IDT_SecureKey, [577](#)
 - IDTechSDK::IDT_SecureMag, [599](#)
 - IDTechSDK::IDT_SpectrumPro, [628](#)
 - IDTechSDK::IDT_UniPay, [699](#)
 - IDTechSDK::IDT_UniPayI_V, [730](#)
 - IDTechSDK::IDT_VP3300, [847](#)
 - IDTechSDK::IDT_VendIII, [800](#)
 - IDTechSDK::IDT_Vendi, [772](#)
- device_getRT1050FirmwareVersion
 - IDTechSDK::IDT_NEO2, [443](#)
- device_getRemoteKeyInjectionTO
 - IDTechSDK::IDT_NEO2, [432](#)
- device_getResponseCodeString
 - IDTechSDK::IDT_Augusta, [93](#)
 - IDTechSDK::IDT_BTMag, [148](#)
 - IDTechSDK::IDT_BTPay, [184](#)
 - IDTechSDK::IDT_MiniSmartII, [345](#)
 - IDTechSDK::IDT_NEO2, [432](#)
 - IDTechSDK::IDT_SREDKey2, [664](#)
 - IDTechSDK::IDT_SecureKey, [567](#)
 - IDTechSDK::IDT_SecureMag, [589](#)
 - IDTechSDK::IDT_SpectrumPro, [618](#)
 - IDTechSDK::IDT_UniPayI_V, [721](#)
 - IDTechSDK::IDT_VP3300, [838](#)
 - IDTechSDK::IDT_VP8800, [901](#)
- device_getSelfCheckTime
 - IDTechSDK::IDT_NEO2, [443](#)
- device_getSpectrumProKSN
 - IDTechSDK::IDT_SpectrumPro, [629](#)
- device_getTransArmorID
 - IDTechSDK::IDT_Augusta, [104](#)
 - IDTechSDK::IDT_NEO2, [443](#)
 - IDTechSDK::IDT_SREDKey2, [674](#)
- device_getTransactionResults
 - IDTechSDK::IDT_KioskIII, [271](#)
 - IDTechSDK::IDT_NEO2, [443](#)
 - IDTechSDK::IDT_PIP, [551](#)
 - IDTechSDK::IDT_VP3300, [848](#)
 - IDTechSDK::IDT_VP8800, [911](#)
 - IDTechSDK::IDT_VendIII, [800](#)
 - IDTechSDK::IDT_Vendi, [773](#)
- device_getUIDofMCU
 - IDTechSDK::IDT_NEO2, [444](#)
 - IDTechSDK::IDT_VP3300, [848](#)
- device_getUsbBootLoader
 - IDTechSDK::IDT_NEO2, [444](#)
 - IDTechSDK::IDT_VP3300, [848](#)
- device_getVersion
 - IDTechSDK::IDT_CM100, [228](#)
- device_getVersions
 - IDTechSDK::IDT_SpectrumPro, [629](#)
- device_isSRED
 - IDTechSDK::IDT_Augusta, [104](#)
- device_listDirectory
 - IDTechSDK::IDT_NEO2, [444](#)
 - IDTechSDK::IDT_VP8800, [911](#)
- device_listenForNotifications
 - IDTechSDK::IDT_NEO2, [445](#)
- device_loadCertCA
 - IDTechSDK::IDT_NEO2, [445](#)
- device_logClear
 - IDTechSDK::IDT_NEO2, [445](#)
- device_logEnable
 - IDTechSDK::IDT_NEO2, [446](#)
- device_logRead
 - IDTechSDK::IDT_NEO2, [446](#)
- device_lowPowerMode
 - IDTechSDK::IDT_NEO2, [446](#)
- device_offYellowLED
 - IDTechSDK::IDT_NEO2, [447](#)
- device_onYellowLED
 - IDTechSDK::IDT_NEO2, [447](#)
- device_pingDevice
 - IDTechSDK::IDT_KioskIII, [271](#)
 - IDTechSDK::IDT_NEO2, [447](#)
 - IDTechSDK::IDT_PIP, [552](#)
 - IDTechSDK::IDT_UniPayI_V, [731](#)
 - IDTechSDK::IDT_VP3300, [849](#)
 - IDTechSDK::IDT_VP8800, [912](#)
 - IDTechSDK::IDT_VendIII, [801](#)
 - IDTechSDK::IDT_Vendi, [773](#)
- device_playAudio
 - IDTechSDK::IDT_NEO2, [447](#)
- device_pollCardReader
 - IDTechSDK::IDT_SpectrumPro, [629](#)
- device_pollForToken
 - IDTechSDK::IDT_KioskIII, [272](#)
 - IDTechSDK::IDT_NEO2, [448](#)
 - IDTechSDK::IDT_PIP, [552](#)
- device_queryFile
 - IDTechSDK::IDT_NEO2, [448](#)
- device_readConfigurationToMemory
 - IDTechSDK::IDT_Augusta, [104](#)
 - IDTechSDK::IDT_BTMag, [159](#)
 - IDTechSDK::IDT_BTPay, [194](#)
 - IDTechSDK::IDT_CM100, [229](#)
 - IDTechSDK::IDT_K100, [239](#)
 - IDTechSDK::IDT_KioskIII, [272](#)
 - IDTechSDK::IDT_L100, [295](#)
 - IDTechSDK::IDT_L80, [324](#)
 - IDTechSDK::IDT_MiniSmartII, [356](#)

- IDTechSDK::IDT_NEO2, [449](#)
- IDTechSDK::IDT_PIP, [553](#)
- IDTechSDK::IDT_SREDKey2, [674](#)
- IDTechSDK::IDT_SecureKey, [577](#)
- IDTechSDK::IDT_SecureMag, [599](#)
- IDTechSDK::IDT_SpectrumPro, [630](#)
- IDTechSDK::IDT_UniPay, [700](#)
- IDTechSDK::IDT_UniPayI_V, [731](#)
- IDTechSDK::IDT_VP3300, [849](#)
- IDTechSDK::IDT_VP8800, [912](#)
- IDTechSDK::IDT_VendIII, [801](#)
- IDTechSDK::IDT_Vendi, [773](#)
- device_readFileFromSD
 - IDTechSDK::IDT_NEO2, [450](#)
- device_rebootDevice
 - IDTechSDK::IDT_Augusta, [105](#)
 - IDTechSDK::IDT_BTMag, [159](#)
 - IDTechSDK::IDT_BTPay, [194](#)
 - IDTechSDK::IDT_K100, [239](#)
 - IDTechSDK::IDT_KioskIII, [273](#)
 - IDTechSDK::IDT_L100, [295](#)
 - IDTechSDK::IDT_L80, [325](#)
 - IDTechSDK::IDT_MiniSmartII, [356](#)
 - IDTechSDK::IDT_NEO2, [450](#)
 - IDTechSDK::IDT_SREDKey2, [675](#)
 - IDTechSDK::IDT_SpectrumPro, [631](#)
 - IDTechSDK::IDT_UniPay, [700](#)
 - IDTechSDK::IDT_VP3300, [849](#)
- device_removeConfigurationGroup
 - IDTechSDK::IDT_VP8800, [913](#)
- device_resetConfigurationGroup
 - IDTechSDK::IDT_NEO2, [451](#)
 - IDTechSDK::IDT_VP8800, [913](#)
- device_resetNVM
 - IDTechSDK::IDT_NEO2, [451](#)
- device_resetTransaction
 - IDTechSDK::IDT_NEO2, [451](#)
- device_retrieveAIDList
 - IDTechSDK::IDT_KioskIII, [274](#)
 - IDTechSDK::IDT_NEO2, [452](#)
 - IDTechSDK::IDT_PIP, [554](#)
 - IDTechSDK::IDT_VP8800, [914](#)
 - IDTechSDK::IDT_VendIII, [802](#)
 - IDTechSDK::IDT_Vendi, [774](#)
- device_retrieveTerminalData
 - IDTechSDK::IDT_NEO2, [452](#)
 - IDTechSDK::IDT_UniPayI_V, [732](#)
- device_rrcConnect
 - IDTechSDK::IDT_NEO2, [452](#)
- device_rrcDisconnect
 - IDTechSDK::IDT_NEO2, [453](#)
- device_rrcDownloadApp
 - IDTechSDK::IDT_NEO2, [453](#)
- device_rrcInstallApp
 - IDTechSDK::IDT_NEO2, [453](#)
- device_rrcRunApp
 - IDTechSDK::IDT_NEO2, [454](#)
- device_rrcUninstallApp
 - IDTechSDK::IDT_NEO2, [454](#)
- device_selfCheck
 - IDTechSDK::IDT_Augusta, [105](#)
 - IDTechSDK::IDT_UniPayI_V, [732](#)
- device_sendBeep
 - IDTechSDK::IDT_CM100, [230](#)
- device_sendConfiguration
 - IDTechSDK::IDT_Augusta, [106](#)
 - IDTechSDK::IDT_BTMag, [160](#)
 - IDTechSDK::IDT_BTPay, [195](#)
 - IDTechSDK::IDT_CM100, [230](#)
 - IDTechSDK::IDT_K100, [240](#)
 - IDTechSDK::IDT_KioskIII, [274](#)
 - IDTechSDK::IDT_L100, [296](#)
 - IDTechSDK::IDT_L80, [326](#)
 - IDTechSDK::IDT_MiniSmartII, [357](#)
 - IDTechSDK::IDT_NEO2, [454](#)
 - IDTechSDK::IDT_PIP, [554](#)
 - IDTechSDK::IDT_SREDKey2, [675](#)
 - IDTechSDK::IDT_SecureKey, [578](#)
 - IDTechSDK::IDT_SecureMag, [600](#)
 - IDTechSDK::IDT_SpectrumPro, [632](#)
 - IDTechSDK::IDT_UniPay, [701](#)
 - IDTechSDK::IDT_UniPayI_V, [732](#)
 - IDTechSDK::IDT_VP3300, [850](#)
 - IDTechSDK::IDT_VP8800, [914](#)
 - IDTechSDK::IDT_VendIII, [802](#)
 - IDTechSDK::IDT_Vendi, [774](#)
- device_sendConfigurationFromZip
 - IDTechSDK::IDT_Augusta, [107](#)
 - IDTechSDK::IDT_BTMag, [161](#)
 - IDTechSDK::IDT_BTPay, [196](#)
 - IDTechSDK::IDT_CM100, [231](#)
 - IDTechSDK::IDT_K100, [241](#)
 - IDTechSDK::IDT_KioskIII, [275](#)
 - IDTechSDK::IDT_L100, [297](#)
 - IDTechSDK::IDT_L80, [327](#)
 - IDTechSDK::IDT_MiniSmartII, [358](#)
 - IDTechSDK::IDT_NEO2, [455](#)
 - IDTechSDK::IDT_PIP, [555](#)
 - IDTechSDK::IDT_SREDKey2, [676](#)
 - IDTechSDK::IDT_SecureKey, [579](#)
 - IDTechSDK::IDT_SecureMag, [601](#)
 - IDTechSDK::IDT_SpectrumPro, [633](#)
 - IDTechSDK::IDT_UniPay, [702](#)
 - IDTechSDK::IDT_UniPayI_V, [733](#)
 - IDTechSDK::IDT_VP3300, [851](#)
 - IDTechSDK::IDT_VP8800, [915](#)
 - IDTechSDK::IDT_VendIII, [803](#)
 - IDTechSDK::IDT_Vendi, [775](#)
- device_sendDataCommand
 - IDTechSDK::IDT_Augusta, [107](#)
 - IDTechSDK::IDT_BTMag, [161](#)
 - IDTechSDK::IDT_BTPay, [197](#)
 - IDTechSDK::IDT_CM100, [232](#)
 - IDTechSDK::IDT_K100, [242](#)
 - IDTechSDK::IDT_KioskIII, [276](#)
 - IDTechSDK::IDT_L100, [298](#)

- IDTechSDK::IDT_L80, [327](#)
- IDTechSDK::IDT_MiniSmartII, [359](#)
- IDTechSDK::IDT_NEO2, [456](#)
- IDTechSDK::IDT_PIP, [556](#)
- IDTechSDK::IDT_SREDKey2, [677](#)
- IDTechSDK::IDT_SecureKey, [580](#)
- IDTechSDK::IDT_SecureMag, [602](#)
- IDTechSDK::IDT_SpectrumPro, [633](#)
- IDTechSDK::IDT_UniPay, [703](#)
- IDTechSDK::IDT_UniPayI_V, [734](#)
- IDTechSDK::IDT_VP3300, [852](#)
- IDTechSDK::IDT_VP8800, [916](#)
- IDTechSDK::IDT_VendIII, [804](#)
- IDTechSDK::IDT_Vendi, [776](#)
- device_sendDataCommand_ext
 - IDTechSDK::IDT_Augusta, [108](#)
 - IDTechSDK::IDT_K100, [242](#)
 - IDTechSDK::IDT_KioskIII, [276](#)
 - IDTechSDK::IDT_L100, [298](#)
 - IDTechSDK::IDT_L80, [328](#)
 - IDTechSDK::IDT_MiniSmartII, [359](#)
 - IDTechSDK::IDT_NEO2, [457](#)
 - IDTechSDK::IDT_PIP, [556](#)
 - IDTechSDK::IDT_SREDKey2, [677](#)
 - IDTechSDK::IDT_SecureKey, [580](#)
 - IDTechSDK::IDT_SecureMag, [602](#)
 - IDTechSDK::IDT_SpectrumPro, [634](#)
 - IDTechSDK::IDT_UniPayI_V, [735](#)
 - IDTechSDK::IDT_VP3300, [852](#)
 - IDTechSDK::IDT_VP8800, [916](#)
 - IDTechSDK::IDT_VendIII, [804](#)
 - IDTechSDK::IDT_Vendi, [777](#)
- device_sendMacDataCommand
 - IDTechSDK::IDT_SpectrumPro, [634](#)
- device_sendPAE
 - IDTechSDK::IDT_Augusta, [108](#)
 - IDTechSDK::IDT_BTMag, [162](#)
 - IDTechSDK::IDT_BTPay, [197](#)
 - IDTechSDK::IDT_CM100, [232](#)
 - IDTechSDK::IDT_KioskIII, [276](#)
 - IDTechSDK::IDT_L100, [299](#)
 - IDTechSDK::IDT_L80, [328](#)
 - IDTechSDK::IDT_MiniSmartII, [359](#)
 - IDTechSDK::IDT_NEO2, [457](#)
 - IDTechSDK::IDT_SecureKey, [580](#)
 - IDTechSDK::IDT_SecureMag, [603](#)
 - IDTechSDK::IDT_SpectrumPro, [635](#)
 - IDTechSDK::IDT_UniPay, [703](#)
 - IDTechSDK::IDT_UniPayI_V, [735](#)
 - IDTechSDK::IDT_VP3300, [852](#)
 - IDTechSDK::IDT_VP8800, [916](#)
 - IDTechSDK::IDT_VendIII, [805](#)
 - IDTechSDK::IDT_Vendi, [777](#)
- device_sendVivoCommandP2
 - IDTechSDK::IDT_KioskIII, [277](#)
 - IDTechSDK::IDT_NEO2, [457](#)
 - IDTechSDK::IDT_PIP, [557](#)
 - IDTechSDK::IDT_SREDKey2, [678](#)
- IDTechSDK::IDT_SecureKey, [581](#)
- IDTechSDK::IDT_UniPayI_V, [735](#)
- IDTechSDK::IDT_VP3300, [853](#)
- IDTechSDK::IDT_VP8800, [917](#)
- IDTechSDK::IDT_VendIII, [805](#)
- IDTechSDK::IDT_Vendi, [777](#)
- device_sendVivoCommandP2_ext
 - IDTechSDK::IDT_KioskIII, [277](#)
 - IDTechSDK::IDT_NEO2, [458](#)
 - IDTechSDK::IDT_PIP, [557](#)
 - IDTechSDK::IDT_SREDKey2, [678](#)
 - IDTechSDK::IDT_SecureKey, [581](#)
 - IDTechSDK::IDT_UniPayI_V, [736](#)
 - IDTechSDK::IDT_VP3300, [853](#)
 - IDTechSDK::IDT_VP8800, [917](#)
 - IDTechSDK::IDT_VendIII, [805](#)
 - IDTechSDK::IDT_Vendi, [778](#)
- device_sendVivoCommandP3
 - IDTechSDK::IDT_NEO2, [458](#)
 - IDTechSDK::IDT_SREDKey2, [679](#)
 - IDTechSDK::IDT_SecureKey, [582](#)
 - IDTechSDK::IDT_VP8800, [917](#)
- device_sendVivoCommandP3_ext
 - IDTechSDK::IDT_NEO2, [459](#)
 - IDTechSDK::IDT_SREDKey2, [679](#)
 - IDTechSDK::IDT_SecureKey, [582](#)
 - IDTechSDK::IDT_VP8800, [918](#)
- device_sendVivoCommandP4
 - IDTechSDK::IDT_NEO2, [459](#)
 - IDTechSDK::IDT_SREDKey2, [679](#)
- device_sendVivoCommandP4_ext
 - IDTechSDK::IDT_NEO2, [459](#)
 - IDTechSDK::IDT_SREDKey2, [680](#)
- device_setAudioVolume
 - IDTechSDK::IDT_NEO2, [460](#)
- device_setBurstMode
 - IDTechSDK::IDT_KioskIII, [278](#)
 - IDTechSDK::IDT_NEO2, [460](#)
 - IDTechSDK::IDT_PIP, [557](#)
 - IDTechSDK::IDT_UniPayI_V, [736](#)
 - IDTechSDK::IDT_VP3300, [854](#)
 - IDTechSDK::IDT_VP8800, [918](#)
 - IDTechSDK::IDT_VendIII, [806](#)
 - IDTechSDK::IDT_Vendi, [778](#)
- device_setBuzzerLED
 - IDTechSDK::IDT_VP8800, [919](#)
- device_setDateTime
 - IDTechSDK::IDT_Augusta, [109](#)
 - IDTechSDK::IDT_BTMag, [162](#)
 - IDTechSDK::IDT_BTPay, [197](#)
 - IDTechSDK::IDT_CM100, [233](#)
 - IDTechSDK::IDT_L100, [299](#)
 - IDTechSDK::IDT_L80, [329](#)
 - IDTechSDK::IDT_MiniSmartII, [360](#)
 - IDTechSDK::IDT_VP3300, [854](#)
- device_setLED
 - IDTechSDK::IDT_NEO2, [460](#)
 - IDTechSDK::IDT_VP3300, [854](#)

- device_setLanguage
 - IDTechSDK::IDT_SREDKey2, [680](#)
- device_setMerchantRecord
 - IDTechSDK::IDT_KioskIII, [278](#)
 - IDTechSDK::IDT_NEO2, [461](#)
 - IDTechSDK::IDT_PIP, [558](#)
 - IDTechSDK::IDT_VP3300, [854](#)
 - IDTechSDK::IDT_VP8800, [919](#)
 - IDTechSDK::IDT_VendIII, [806](#)
 - IDTechSDK::IDT_Vendi, [779](#)
- device_setOutputType
 - IDTechSDK::IDT_SREDKey2, [680](#)
 - IDTechSDK::IDT_SecureKey, [582](#)
- device_setPIPMODE
 - IDTechSDK::IDT_PIP, [558](#)
- device_setPollMode
 - IDTechSDK::IDT_KioskIII, [278](#)
 - IDTechSDK::IDT_NEO2, [461](#)
 - IDTechSDK::IDT_PIP, [558](#)
 - IDTechSDK::IDT_SREDKey2, [681](#)
 - IDTechSDK::IDT_UniPayI_V, [737](#)
 - IDTechSDK::IDT_VP3300, [855](#)
 - IDTechSDK::IDT_VP8800, [920](#)
 - IDTechSDK::IDT_VendIII, [806](#)
 - IDTechSDK::IDT_Vendi, [779](#)
- device_setQuickChipHIDMode
 - IDTechSDK::IDT_VP3300, [855](#)
- device_setQuickChipMode
 - IDTechSDK::IDT_Augusta, [109](#)
- device_setSelfCheckTime
 - IDTechSDK::IDT_NEO2, [461](#)
- device_setSleepModeTime
 - IDTechSDK::IDT_L100, [299](#)
 - IDTechSDK::IDT_L80, [329](#)
- device_setSpectrumProBDK
 - IDTechSDK::IDT_SpectrumPro, [635](#)
- device_setTerminalData
 - IDTechSDK::IDT_NEO2, [462](#)
 - IDTechSDK::IDT_UniPayI_V, [737](#)
- device_setTransArmorEncryption
 - IDTechSDK::IDT_Augusta, [109](#)
 - IDTechSDK::IDT_NEO2, [462](#)
 - IDTechSDK::IDT_SREDKey2, [681](#)
- device_setTransArmorID
 - IDTechSDK::IDT_Augusta, [109](#)
 - IDTechSDK::IDT_NEO2, [462](#)
 - IDTechSDK::IDT_SREDKey2, [681](#)
- device_setUID
 - IDTechSDK::IDT_SpectrumPro, [635](#)
- device_startRKI
 - IDTechSDK::IDT_Augusta, [110](#)
 - IDTechSDK::IDT_BTPay, [199](#)
 - IDTechSDK::IDT_K100, [243](#)
 - IDTechSDK::IDT_KioskIII, [279](#)
 - IDTechSDK::IDT_L100, [300](#)
 - IDTechSDK::IDT_MiniSmartII, [360](#)
 - IDTechSDK::IDT_NEO2, [463](#)
 - IDTechSDK::IDT_SREDKey2, [682](#)
 - IDTechSDK::IDT_SpectrumPro, [636](#)
 - IDTechSDK::IDT_UniPay, [703](#)
 - IDTechSDK::IDT_UniPayI_V, [737](#)
 - IDTechSDK::IDT_VP3300, [856](#)
 - IDTechSDK::IDT_VP8800, [920](#)
 - IDTechSDK::IDT_VendIII, [807](#)
 - IDTechSDK::IDT_Vendi, [779](#)
- device_startTransaction
 - IDTechSDK::IDT_Augusta, [110](#)
 - IDTechSDK::IDT_KioskIII, [279](#)
 - IDTechSDK::IDT_MiniSmartII, [361](#)
 - IDTechSDK::IDT_NEO2, [463](#)
 - IDTechSDK::IDT_PIP, [559](#)
 - IDTechSDK::IDT_SpectrumPro, [637](#)
 - IDTechSDK::IDT_UniPayI_V, [738](#)
 - IDTechSDK::IDT_VP3300, [856](#)
 - IDTechSDK::IDT_VP8800, [921](#)
 - IDTechSDK::IDT_VendIII, [807](#)
 - IDTechSDK::IDT_Vendi, [779](#)
- device_startTransactionCB
 - IDTechSDK::IDT_NEO2, [465](#)
- device_stopAudio
 - IDTechSDK::IDT_NEO2, [466](#)
- device_terminalInfo
 - IDTechSDK::IDT_NEO2, [466](#)
- device_transferFile
 - IDTechSDK::IDT_NEO2, [467](#)
 - IDTechSDK::IDT_VP8800, [922](#)
- device_updateDeviceFirmware
 - IDTechSDK::IDT_Augusta, [111](#)
 - IDTechSDK::IDT_KioskIII, [280](#)
 - IDTechSDK::IDT_L100, [300](#)
 - IDTechSDK::IDT_L80, [329](#)
 - IDTechSDK::IDT_MiniSmartII, [361](#)
 - IDTechSDK::IDT_NEO2, [467](#)
 - IDTechSDK::IDT_PIP, [560](#)
 - IDTechSDK::IDT_SREDKey2, [682](#)
 - IDTechSDK::IDT_UniPayI_V, [739](#)
 - IDTechSDK::IDT_VP3300, [857](#)
 - IDTechSDK::IDT_VP8800, [922](#)
 - IDTechSDK::IDT_VendIII, [807](#)
 - IDTechSDK::IDT_Vendi, [781](#)
- device_updateDeviceFromManifest
 - IDTechSDK::IDT_NEO2, [469](#)
 - IDTechSDK::IDT_SREDKey2, [683](#)
- device_updateFirmware
 - IDTechSDK::IDT_SpectrumPro, [637](#)
- device_updateFirmwareFromZip
 - IDTechSDK::IDT_Augusta, [112](#)
 - IDTechSDK::IDT_BTMag, [162](#)
 - IDTechSDK::IDT_K100, [243](#)
 - IDTechSDK::IDT_KioskIII, [282](#)
 - IDTechSDK::IDT_L100, [301](#)
 - IDTechSDK::IDT_L80, [330](#)
 - IDTechSDK::IDT_MiniSmartII, [362](#)
 - IDTechSDK::IDT_NEO2, [469](#)
 - IDTechSDK::IDT_PIP, [561](#)
 - IDTechSDK::IDT_SREDKey2, [684](#)

- IDTechSDK::IDT_SecureKey, [583](#)
- IDTechSDK::IDT_SecureMag, [604](#)
- IDTechSDK::IDT_UniPay, [705](#)
- IDTechSDK::IDT_UniPayI_V, [740](#)
- IDTechSDK::IDT_VP3300, [859](#)
- IDTechSDK::IDT_VP8800, [924](#)
- IDTechSDK::IDT_VendIII, [809](#)
- IDTechSDK::IDT_Vendi, [782](#)
- device_updateFirmwareIP
 - IDTechSDK::IDT_NEO2, [469](#)
- device_updateFirmwareKernels
 - IDTechSDK::IDT_NEO2, [471](#)
- device_updateFirmwareType
 - IDTechSDK::IDT_NEO2, [472](#), [473](#)
 - IDTechSDK::IDT_SREDKey2, [684](#)
- device_verifyBackdoorKey
 - IDTechSDK::IDT_Augusta, [113](#)
- device_wakeDevice
 - IDTechSDK::IDT_NEO2, [475](#)
- emv_activateTransaction
 - IDTechSDK::IDT_Augusta, [113](#)
 - IDTechSDK::IDT_MiniSmartII, [363](#)
 - IDTechSDK::IDT_NEO2, [476](#)
 - IDTechSDK::IDT_SpectrumPro, [638](#)
 - IDTechSDK::IDT_UniPayI_V, [740](#)
 - IDTechSDK::IDT_VP3300, [859](#)
 - IDTechSDK::IDT_VP8800, [924](#)
 - IDTechSDK::IDT_VendIII, [809](#)
- emv_addTerminalData
 - IDTechSDK::IDT_Augusta, [113](#)
 - IDTechSDK::IDT_BTPay, [199](#)
 - IDTechSDK::IDT_MiniSmartII, [363](#)
 - IDTechSDK::IDT_NEO2, [476](#)
 - IDTechSDK::IDT_SpectrumPro, [638](#)
 - IDTechSDK::IDT_UniPayI_V, [741](#)
 - IDTechSDK::IDT_VP3300, [860](#)
 - IDTechSDK::IDT_VP8800, [925](#)
- emv_allowFallback
 - IDTechSDK::IDT_Augusta, [114](#)
 - IDTechSDK::IDT_BTPay, [199](#)
 - IDTechSDK::IDT_MiniSmartII, [364](#)
 - IDTechSDK::IDT_NEO2, [476](#)
 - IDTechSDK::IDT_SpectrumPro, [639](#)
 - IDTechSDK::IDT_UniPayI_V, [741](#)
 - IDTechSDK::IDT_VP3300, [860](#)
 - IDTechSDK::IDT_VP8800, [925](#)
- emv_authenticateTransaction
 - IDTechSDK::IDT_Augusta, [114](#)
 - IDTechSDK::IDT_BTPay, [199](#)
 - IDTechSDK::IDT_MiniSmartII, [364](#)
 - IDTechSDK::IDT_NEO2, [477](#)
 - IDTechSDK::IDT_SpectrumPro, [639](#)
 - IDTechSDK::IDT_UniPayI_V, [741](#)
 - IDTechSDK::IDT_VP3300, [860](#)
- emv_authenticateTransactionCB
 - IDTechSDK::IDT_NEO2, [477](#)
- emv_autoAuthenticate
 - IDTechSDK::IDT_Augusta, [114](#)
- IDTechSDK::IDT_BTPay, [200](#)
- IDTechSDK::IDT_MiniSmartII, [364](#)
- IDTechSDK::IDT_NEO2, [477](#)
- IDTechSDK::IDT_SpectrumPro, [639](#)
- IDTechSDK::IDT_UniPayI_V, [742](#)
- IDTechSDK::IDT_VP3300, [860](#)
- IDTechSDK::IDT_VP8800, [925](#)
- emv_autoAuthenticateTags
 - IDTechSDK::IDT_Augusta, [115](#)
 - IDTechSDK::IDT_BTPay, [200](#)
 - IDTechSDK::IDT_MiniSmartII, [365](#)
 - IDTechSDK::IDT_NEO2, [478](#)
 - IDTechSDK::IDT_SpectrumPro, [640](#)
 - IDTechSDK::IDT_UniPayI_V, [742](#)
 - IDTechSDK::IDT_VP3300, [861](#)
 - IDTechSDK::IDT_VP8800, [925](#)
- emv_callbackResponseKSN
 - IDTechSDK::IDT_NEO2, [478](#)
- emv_callbackResponseLCD
 - IDTechSDK::IDT_Augusta, [115](#)
 - IDTechSDK::IDT_BTPay, [200](#)
 - IDTechSDK::IDT_MiniSmartII, [365](#)
 - IDTechSDK::IDT_NEO2, [478](#)
 - IDTechSDK::IDT_SpectrumPro, [640](#)
 - IDTechSDK::IDT_UniPayI_V, [742](#)
 - IDTechSDK::IDT_VP3300, [861](#)
 - IDTechSDK::IDT_VP8800, [925](#)
- emv_callbackResponseMSR
 - IDTechSDK::IDT_Augusta, [115](#)
 - IDTechSDK::IDT_BTPay, [201](#)
 - IDTechSDK::IDT_MiniSmartII, [365](#)
 - IDTechSDK::IDT_NEO2, [479](#)
 - IDTechSDK::IDT_SpectrumPro, [640](#)
 - IDTechSDK::IDT_UniPayI_V, [743](#)
 - IDTechSDK::IDT_VP3300, [861](#)
 - IDTechSDK::IDT_VP8800, [926](#)
- emv_callbackResponsePIN_ETC
 - IDTechSDK::IDT_NEO2, [479](#)
- emv_callbackResponsePIN
 - IDTechSDK::IDT_Augusta, [115](#)
 - IDTechSDK::IDT_BTPay, [201](#)
 - IDTechSDK::IDT_MiniSmartII, [365](#)
 - IDTechSDK::IDT_NEO2, [479](#)
 - IDTechSDK::IDT_SpectrumPro, [641](#)
 - IDTechSDK::IDT_UniPayI_V, [743](#)
 - IDTechSDK::IDT_VP3300, [862](#)
 - IDTechSDK::IDT_VP8800, [926](#)
- emv_cancelTransaction
 - IDTechSDK::IDT_Augusta, [116](#)
 - IDTechSDK::IDT_BTPay, [202](#)
 - IDTechSDK::IDT_MiniSmartII, [366](#)
 - IDTechSDK::IDT_NEO2, [480](#)
 - IDTechSDK::IDT_SpectrumPro, [641](#)
 - IDTechSDK::IDT_UniPayI_V, [743](#)
 - IDTechSDK::IDT_VP3300, [862](#)
 - IDTechSDK::IDT_VP8800, [927](#)
- emv_clearTransactionLog
 - IDTechSDK::IDT_SpectrumPro, [641](#)

- IDTechSDK::IDT_VP8800, [927](#)
- emv_completeTransaction
 - IDTechSDK::IDT_Augusta, [116](#)
 - IDTechSDK::IDT_BTPay, [202](#)
 - IDTechSDK::IDT_MiniSmartII, [366](#)
 - IDTechSDK::IDT_NEO2, [480](#)
 - IDTechSDK::IDT_SpectrumPro, [642](#)
 - IDTechSDK::IDT_UniPayI_V, [744](#)
 - IDTechSDK::IDT_VP3300, [862](#)
 - IDTechSDK::IDT_VP8800, [927](#)
 - IDTechSDK::IDT_VendIII, [810](#)
- emv_completeTransactionCB
 - IDTechSDK::IDT_NEO2, [481](#)
- emv_continueTransactionForCV
 - IDTechSDK::IDT_VP8800, [928](#)
- emv_exchangeCerts
 - IDTechSDK::IDT_NEO2, [481](#)
- emv_generateDUKPT
 - IDTechSDK::IDT_NEO2, [482](#)
- emv_getCRLStatus
 - IDTechSDK::IDT_VP8800, [929](#)
- emv_getConfigurationGroup
 - IDTechSDK::IDT_VP8800, [928](#)
- emv_getEMVConfigurationCheckValue
 - IDTechSDK::IDT_Augusta, [117](#)
 - IDTechSDK::IDT_BTPay, [202](#)
 - IDTechSDK::IDT_MiniSmartII, [367](#)
 - IDTechSDK::IDT_NEO2, [482](#)
 - IDTechSDK::IDT_SpectrumPro, [642](#)
 - IDTechSDK::IDT_UniPayI_V, [744](#)
 - IDTechSDK::IDT_VP3300, [863](#)
 - IDTechSDK::IDT_VP8800, [929](#)
- emv_getEMVKernelCheckValue
 - IDTechSDK::IDT_Augusta, [117](#)
 - IDTechSDK::IDT_BTPay, [203](#)
 - IDTechSDK::IDT_MiniSmartII, [367](#)
 - IDTechSDK::IDT_NEO2, [482](#)
 - IDTechSDK::IDT_SpectrumPro, [642](#)
 - IDTechSDK::IDT_UniPayI_V, [744](#)
 - IDTechSDK::IDT_VP3300, [863](#)
 - IDTechSDK::IDT_VP8800, [929](#)
- emv_getEMVKernelVersion
 - IDTechSDK::IDT_Augusta, [117](#)
 - IDTechSDK::IDT_BTPay, [203](#)
 - IDTechSDK::IDT_MiniSmartII, [367](#)
 - IDTechSDK::IDT_NEO2, [482](#)
 - IDTechSDK::IDT_SpectrumPro, [643](#)
 - IDTechSDK::IDT_UniPayI_V, [745](#)
 - IDTechSDK::IDT_VP3300, [863](#)
 - IDTechSDK::IDT_VP8800, [929](#)
- emv_getEMVKernelVersionExt
 - IDTechSDK::IDT_NEO2, [483](#)
 - IDTechSDK::IDT_UniPayI_V, [745](#)
 - IDTechSDK::IDT_VP3300, [864](#)
 - IDTechSDK::IDT_VP8800, [930](#)
- emv_getExemptionLogStatus
 - IDTechSDK::IDT_VP8800, [930](#)
- emv_getTerminalID
 - IDTechSDK::IDT_SpectrumPro, [643](#)
- emv_getTerminalMajorConfiguration
 - IDTechSDK::IDT_NEO2, [483](#)
 - IDTechSDK::IDT_SpectrumPro, [643](#)
 - IDTechSDK::IDT_VP3300, [864](#)
- emv_getTransactionLogRecord
 - IDTechSDK::IDT_VP8800, [930](#)
- emv_getTransactionLogStatus
 - IDTechSDK::IDT_VP8800, [932](#)
- emv_removeAllApplicationData
 - IDTechSDK::IDT_Augusta, [118](#)
 - IDTechSDK::IDT_BTPay, [203](#)
 - IDTechSDK::IDT_MiniSmartII, [368](#)
 - IDTechSDK::IDT_NEO2, [483](#)
 - IDTechSDK::IDT_SpectrumPro, [644](#)
 - IDTechSDK::IDT_UniPayI_V, [745](#)
 - IDTechSDK::IDT_VP3300, [864](#)
- emv_removeAllCAPK
 - IDTechSDK::IDT_Augusta, [118](#)
 - IDTechSDK::IDT_BTPay, [204](#)
 - IDTechSDK::IDT_MiniSmartII, [368](#)
 - IDTechSDK::IDT_NEO2, [484](#)
 - IDTechSDK::IDT_SpectrumPro, [644](#)
 - IDTechSDK::IDT_UniPayI_V, [745](#)
 - IDTechSDK::IDT_VP3300, [865](#)
 - IDTechSDK::IDT_VP8800, [932](#)
- emv_removeAllCRL
 - IDTechSDK::IDT_Augusta, [118](#)
 - IDTechSDK::IDT_BTPay, [204](#)
 - IDTechSDK::IDT_MiniSmartII, [368](#)
 - IDTechSDK::IDT_NEO2, [484](#)
 - IDTechSDK::IDT_SpectrumPro, [644](#)
 - IDTechSDK::IDT_UniPayI_V, [746](#)
 - IDTechSDK::IDT_VP3300, [865](#)
 - IDTechSDK::IDT_VP8800, [932](#)
- emv_removeAllExceptions
 - IDTechSDK::IDT_VP8800, [932](#)
- emv_removeApplicationData
 - IDTechSDK::IDT_Augusta, [118](#)
 - IDTechSDK::IDT_BTPay, [204](#)
 - IDTechSDK::IDT_MiniSmartII, [368](#)
 - IDTechSDK::IDT_NEO2, [484](#)
 - IDTechSDK::IDT_SpectrumPro, [645](#)
 - IDTechSDK::IDT_UniPayI_V, [746](#)
 - IDTechSDK::IDT_VP3300, [865](#)
 - IDTechSDK::IDT_VP8800, [933](#)
- emv_removeCAPK
 - IDTechSDK::IDT_Augusta, [119](#)
 - IDTechSDK::IDT_BTPay, [204](#)
 - IDTechSDK::IDT_MiniSmartII, [369](#)
 - IDTechSDK::IDT_NEO2, [484](#)
 - IDTechSDK::IDT_SpectrumPro, [645](#)
 - IDTechSDK::IDT_UniPayI_V, [746](#)
 - IDTechSDK::IDT_VP3300, [865](#)
 - IDTechSDK::IDT_VP8800, [933](#)
- emv_removeCRL
 - IDTechSDK::IDT_Augusta, [119](#)
 - IDTechSDK::IDT_BTPay, [205](#)

- IDTechSDK::IDT_MiniSmartII, [369](#)
- IDTechSDK::IDT_NEO2, [485](#)
- IDTechSDK::IDT_SpectrumPro, [645](#)
- IDTechSDK::IDT_UniPayI_V, [747](#)
- IDTechSDK::IDT_VP3300, [866](#)
- IDTechSDK::IDT_VP8800, [934](#)
- emv_removeConfigurationGroup
 - IDTechSDK::IDT_VP8800, [933](#)
- emv_removeException
 - IDTechSDK::IDT_VP8800, [934](#)
- emv_removeTerminalData
 - IDTechSDK::IDT_Augusta, [119](#)
 - IDTechSDK::IDT_BTPay, [205](#)
 - IDTechSDK::IDT_MiniSmartII, [369](#)
 - IDTechSDK::IDT_NEO2, [485](#)
 - IDTechSDK::IDT_SpectrumPro, [646](#)
 - IDTechSDK::IDT_UniPayI_V, [747](#)
 - IDTechSDK::IDT_VP3300, [866](#)
 - IDTechSDK::IDT_VP8800, [934](#)
- emv_removeTransactionAmountLog
 - IDTechSDK::IDT_VP8800, [935](#)
- emv_resetConfigurationGroup
 - IDTechSDK::IDT_NEO2, [485](#)
 - IDTechSDK::IDT_VP8800, [935](#)
- emv_retrieveAIDList
 - IDTechSDK::IDT_Augusta, [120](#)
 - IDTechSDK::IDT_BTPay, [205](#)
 - IDTechSDK::IDT_MiniSmartII, [370](#)
 - IDTechSDK::IDT_NEO2, [486](#)
 - IDTechSDK::IDT_SpectrumPro, [646](#)
 - IDTechSDK::IDT_UniPayI_V, [747](#)
 - IDTechSDK::IDT_VP3300, [866](#)
 - IDTechSDK::IDT_VP8800, [935](#)
- emv_retrieveApplicationData
 - IDTechSDK::IDT_Augusta, [120](#)
 - IDTechSDK::IDT_BTPay, [206](#)
 - IDTechSDK::IDT_MiniSmartII, [370](#)
 - IDTechSDK::IDT_NEO2, [486](#)
 - IDTechSDK::IDT_SpectrumPro, [646](#)
 - IDTechSDK::IDT_UniPayI_V, [747](#)
 - IDTechSDK::IDT_VP3300, [867](#)
 - IDTechSDK::IDT_VP8800, [935](#)
- emv_retrieveCAPKList
 - IDTechSDK::IDT_Augusta, [121](#)
 - IDTechSDK::IDT_BTPay, [206](#)
 - IDTechSDK::IDT_MiniSmartII, [371](#)
 - IDTechSDK::IDT_NEO2, [487](#)
 - IDTechSDK::IDT_SpectrumPro, [647](#)
 - IDTechSDK::IDT_UniPayI_V, [748](#)
 - IDTechSDK::IDT_VP3300, [867](#)
 - IDTechSDK::IDT_VP8800, [936](#)
- emv_retrieveCAPK
 - IDTechSDK::IDT_Augusta, [120](#)
 - IDTechSDK::IDT_BTPay, [206](#)
 - IDTechSDK::IDT_MiniSmartII, [370](#)
 - IDTechSDK::IDT_NEO2, [486](#)
 - IDTechSDK::IDT_SpectrumPro, [646](#)
 - IDTechSDK::IDT_UniPayI_V, [748](#)
- IDTechSDK::IDT_VP3300, [867](#)
- IDTechSDK::IDT_VP8800, [936](#)
- emv_retrieveCRLList
 - IDTechSDK::IDT_Augusta, [121](#)
 - IDTechSDK::IDT_BTPay, [207](#)
 - IDTechSDK::IDT_MiniSmartII, [371](#)
 - IDTechSDK::IDT_NEO2, [487](#)
 - IDTechSDK::IDT_SpectrumPro, [647](#)
 - IDTechSDK::IDT_UniPayI_V, [749](#)
 - IDTechSDK::IDT_VP3300, [868](#)
 - IDTechSDK::IDT_VP8800, [937](#)
- emv_retrieveExceptionList
 - IDTechSDK::IDT_VP8800, [937](#)
- emv_retrieveTerminalData
 - IDTechSDK::IDT_Augusta, [121](#)
 - IDTechSDK::IDT_BTPay, [207](#)
 - IDTechSDK::IDT_MiniSmartII, [372](#)
 - IDTechSDK::IDT_NEO2, [488](#)
 - IDTechSDK::IDT_SpectrumPro, [648](#)
 - IDTechSDK::IDT_UniPayI_V, [749](#)
 - IDTechSDK::IDT_VP3300, [868](#)
 - IDTechSDK::IDT_VP8800, [937](#)
- emv_retrieveTransactionResult
 - IDTechSDK::IDT_Augusta, [122](#)
 - IDTechSDK::IDT_BTPay, [207](#)
 - IDTechSDK::IDT_MiniSmartII, [372](#)
 - IDTechSDK::IDT_NEO2, [488](#)
 - IDTechSDK::IDT_SpectrumPro, [648](#)
 - IDTechSDK::IDT_UniPayI_V, [749](#)
 - IDTechSDK::IDT_VP3300, [868](#)
 - IDTechSDK::IDT_VP8800, [938](#)
- emv_setApplicationData
 - IDTechSDK::IDT_Augusta, [122](#)
 - IDTechSDK::IDT_BTPay, [208](#)
 - IDTechSDK::IDT_MiniSmartII, [372](#)
 - IDTechSDK::IDT_NEO2, [488, 489](#)
 - IDTechSDK::IDT_SpectrumPro, [648](#)
 - IDTechSDK::IDT_UniPayI_V, [749](#)
 - IDTechSDK::IDT_VP3300, [869](#)
 - IDTechSDK::IDT_VP8800, [938](#)
- emv_setCAPK
 - IDTechSDK::IDT_Augusta, [122](#)
 - IDTechSDK::IDT_BTPay, [208](#)
 - IDTechSDK::IDT_MiniSmartII, [372](#)
 - IDTechSDK::IDT_NEO2, [489](#)
 - IDTechSDK::IDT_SpectrumPro, [648](#)
 - IDTechSDK::IDT_UniPayI_V, [750](#)
 - IDTechSDK::IDT_VP3300, [869](#)
 - IDTechSDK::IDT_VP8800, [939](#)
- emv_setCRL
 - IDTechSDK::IDT_Augusta, [123](#)
 - IDTechSDK::IDT_BTPay, [209](#)
 - IDTechSDK::IDT_MiniSmartII, [373](#)
 - IDTechSDK::IDT_NEO2, [490](#)
 - IDTechSDK::IDT_SpectrumPro, [649](#)
 - IDTechSDK::IDT_UniPayI_V, [750](#)
 - IDTechSDK::IDT_VP3300, [870](#)
 - IDTechSDK::IDT_VP8800, [939](#)

- emv_setException
 - IDTechSDK::IDT_VP8800, [940](#)
- emv_setTerminalData
 - IDTechSDK::IDT_Augusta, [123](#)
 - IDTechSDK::IDT_BTPay, [209](#)
 - IDTechSDK::IDT_MiniSmartII, [373](#)
 - IDTechSDK::IDT_NEO2, [490](#)
 - IDTechSDK::IDT_SpectrumPro, [649](#)
 - IDTechSDK::IDT_UniPayI_V, [751](#)
 - IDTechSDK::IDT_VP3300, [870](#)
- emv_setTerminalDataVP8800
 - IDTechSDK::IDT_VP8800, [940](#)
- emv_setTerminalID
 - IDTechSDK::IDT_SpectrumPro, [650](#)
- emv_setTerminalMajorConfiguration
 - IDTechSDK::IDT_Augusta, [123](#)
 - IDTechSDK::IDT_BTPay, [209](#)
 - IDTechSDK::IDT_MiniSmartII, [374](#)
 - IDTechSDK::IDT_NEO2, [491](#)
 - IDTechSDK::IDT_SpectrumPro, [650](#)
 - IDTechSDK::IDT_UniPayI_V, [751](#)
 - IDTechSDK::IDT_VP3300, [870](#)
- emv_startTransaction
 - IDTechSDK::IDT_Augusta, [124](#)
 - IDTechSDK::IDT_BTPay, [209](#)
 - IDTechSDK::IDT_MiniSmartII, [374](#)
 - IDTechSDK::IDT_NEO2, [491](#)
 - IDTechSDK::IDT_SpectrumPro, [650](#)
 - IDTechSDK::IDT_UniPayI_V, [751](#)
 - IDTechSDK::IDT_VP3300, [870](#)
 - IDTechSDK::IDT_VP8800, [940](#)
 - IDTechSDK::IDT_VendIII, [810](#)
- emv_startTransactionCB
 - IDTechSDK::IDT_NEO2, [491](#)
- emv_trySetTerminalData
 - IDTechSDK::IDT_Augusta, [124](#)
 - IDTechSDK::IDT_BTPay, [210](#)
 - IDTechSDK::IDT_MiniSmartII, [375](#)
 - IDTechSDK::IDT_NEO2, [492](#)
 - IDTechSDK::IDT_SpectrumPro, [651](#)
 - IDTechSDK::IDT_UniPayI_V, [752](#)
 - IDTechSDK::IDT_VP3300, [871](#)
 - IDTechSDK::IDT_VP8800, [941](#)
- emv_verifyDUKPTLoaded
 - IDTechSDK::IDT_NEO2, [493](#)
- felica_SendCommand
 - IDTechSDK::IDT_KioskIII, [284](#)
 - IDTechSDK::IDT_NEO2, [494](#)
 - IDTechSDK::IDT_VP8800, [943](#)
- felica_authentication
 - IDTechSDK::IDT_KioskIII, [282](#)
 - IDTechSDK::IDT_NEO2, [493](#)
 - IDTechSDK::IDT_VP8800, [941](#)
- felica_read
 - IDTechSDK::IDT_KioskIII, [282](#)
 - IDTechSDK::IDT_NEO2, [493](#)
 - IDTechSDK::IDT_VP8800, [942](#)
- felica_readWithMac
 - IDTechSDK::IDT_KioskIII, [283](#)
 - IDTechSDK::IDT_NEO2, [494](#)
 - IDTechSDK::IDT_VP8800, [942](#)
- felica_requestService
 - IDTechSDK::IDT_KioskIII, [283](#)
 - IDTechSDK::IDT_NEO2, [494](#)
 - IDTechSDK::IDT_PIP, [562](#)
 - IDTechSDK::IDT_VP8800, [943](#)
- felica_write
 - IDTechSDK::IDT_KioskIII, [284](#)
 - IDTechSDK::IDT_NEO2, [495](#)
 - IDTechSDK::IDT_VP8800, [943](#)
- felica_writeWithMac
 - IDTechSDK::IDT_KioskIII, [284](#)
 - IDTechSDK::IDT_NEO2, [495](#)
 - IDTechSDK::IDT_VP8800, [944](#)
- getCommandTimeout
 - IDTechSDK::IDT_Augusta, [125](#)
 - IDTechSDK::IDT_BTMag, [163](#)
 - IDTechSDK::IDT_BTPay, [210](#)
 - IDTechSDK::IDT_CM100, [233](#)
 - IDTechSDK::IDT_K100, [243](#)
 - IDTechSDK::IDT_KioskIII, [285](#)
 - IDTechSDK::IDT_L100, [303](#)
 - IDTechSDK::IDT_L80, [331](#)
 - IDTechSDK::IDT_MiniSmartII, [375](#)
 - IDTechSDK::IDT_NEO2, [495](#)
 - IDTechSDK::IDT_PIP, [562](#)
 - IDTechSDK::IDT_SREDKey2, [686](#)
 - IDTechSDK::IDT_SecureKey, [584](#)
 - IDTechSDK::IDT_SecureMag, [604](#)
 - IDTechSDK::IDT_SpectrumPro, [651](#)
 - IDTechSDK::IDT_UniPay, [705](#)
 - IDTechSDK::IDT_UniPayI_V, [752](#)
 - IDTechSDK::IDT_VP3300, [871](#)
 - IDTechSDK::IDT_VP8800, [944](#)
 - IDTechSDK::IDT_Vendi, [782](#)
- getLastErrorMessage
 - IDTechSDK::IDT_NEO2, [496](#)
- IDTechSDK.IDT_Augusta, [81](#)
- IDTechSDK.IDT_BTMag, [141](#)
- IDTechSDK.IDT_BTPay, [174](#)
- IDTechSDK.IDT_CM100, [226](#)
- IDTechSDK.IDT_K100, [234](#)
- IDTechSDK.IDT_KioskIII, [247](#)
- IDTechSDK.IDT_L100, [287](#)
- IDTechSDK.IDT_L80, [318](#)
- IDTechSDK.IDT_MiniSmartII, [339](#)
- IDTechSDK.IDT_NEO2, [382](#)
- IDTechSDK.IDT_PIP, [537](#)
- IDTechSDK.IDT_SREDKey2, [659](#)
- IDTechSDK.IDT_SecureKey, [563](#)
- IDTechSDK.IDT_SecureMag, [586](#)
- IDTechSDK.IDT_SpectrumPro, [612](#)
- IDTechSDK.IDT_UniPay, [695](#)
- IDTechSDK.IDT_UniPayI_V, [714](#)
- IDTechSDK.IDT_VP3300, [814](#)

- IDTechSDK.IDT_VP8800, [876](#)
- IDTechSDK.IDT_VendIII, [786](#)
- IDTechSDK.IDT_Vendi, [757](#)
- IDTechSDK::IDT_Augusta
 - config_getBeeperController, [84](#)
 - config_getEncryptionControl, [84](#)
 - config_getLEDController, [85](#)
 - config_getModelNumber, [85](#)
 - config_getSerialNumber, [86](#)
 - config_setBeeperController, [86](#)
 - config_setCmdTimeOutDuration, [86](#)
 - config_setEncryptionControl, [86](#)
 - config_setLEDController, [87](#)
 - createFastEMVData, [87](#)
 - device_RemoteKeyInjection, [105](#)
 - device_StartRKI, [110](#)
 - device_activateTransaction, [88](#)
 - device_controlBeep, [88](#)
 - device_controlLED_ICC, [89](#)
 - device_controlLED_MSR, [89](#)
 - device_controlLED, [88](#)
 - device_disableAugustaLED, [90](#)
 - device_getAnyRKIStatus, [90](#)
 - device_getConfigurationFromMemory, [91](#)
 - device_getDRS, [91](#)
 - device_getDateTime, [91](#)
 - device_getFirmwareVersion, [92](#)
 - device_getKeyFormatForICCDUKPT, [92](#)
 - device_getKeyStatus, [92](#)
 - device_getKeyTypeForICCDUKPT, [93](#)
 - device_getQuickChipMode, [93](#)
 - device_getRKIStatus, [103](#)
 - device_getResponseCodeString, [93](#)
 - device_getTransArmorID, [104](#)
 - device_isSRED, [104](#)
 - device_readConfigurationToMemory, [104](#)
 - device_rebootDevice, [105](#)
 - device_selfCheck, [105](#)
 - device_sendConfiguration, [106](#)
 - device_sendConfigurationFromZip, [107](#)
 - device_sendDataCommand, [107](#)
 - device_sendDataCommand_ext, [108](#)
 - device_sendPAE, [108](#)
 - device_setDateTime, [109](#)
 - device_setQuickChipMode, [109](#)
 - device_setTransArmorEncryption, [109](#)
 - device_setTransArmorID, [109](#)
 - device_startRKI, [110](#)
 - device_startTransaction, [110](#)
 - device_updateDeviceFirmware, [111](#)
 - device_updateFirmwareFromZip, [112](#)
 - device_verifyBackdoorKey, [113](#)
 - emv_activateTransaction, [113](#)
 - emv_addTerminalData, [113](#)
 - emv_allowFallback, [114](#)
 - emv_authenticateTransaction, [114](#)
 - emv_autoAuthenticate, [114](#)
 - emv_autoAuthenticateTags, [115](#)
 - emv_callbackResponseLCD, [115](#)
 - emv_callbackResponseMSR, [115](#)
 - emv_callbackResponsePIN, [115](#)
 - emv_cancelTransaction, [116](#)
 - emv_completeTransaction, [116](#)
 - emv_getEMVConfigurationCheckValue, [117](#)
 - emv_getEMVKernelCheckValue, [117](#)
 - emv_getEMVKernelVersion, [117](#)
 - emv_removeAllApplicationData, [118](#)
 - emv_removeAllCAPK, [118](#)
 - emv_removeAllCRL, [118](#)
 - emv_removeApplicationData, [118](#)
 - emv_removeCAPK, [119](#)
 - emv_removeCRL, [119](#)
 - emv_removeTerminalData, [119](#)
 - emv_retrieveAIDList, [120](#)
 - emv_retrieveApplicationData, [120](#)
 - emv_retrieveCAPKList, [121](#)
 - emv_retrieveCAPK, [120](#)
 - emv_retrieveCRLList, [121](#)
 - emv_retrieveTerminalData, [121](#)
 - emv_retrieveTransactionResult, [122](#)
 - emv_setApplicationData, [122](#)
 - emv_setCAPK, [122](#)
 - emv_setCRL, [123](#)
 - emv_setTerminalData, [123](#)
 - emv_setTerminalMajorConfiguration, [123](#)
 - emv_startTransaction, [124](#)
 - emv_trySetTerminalData, [124](#)
 - getCommandTimeout, [125](#)
 - icc_disable, [125](#)
 - icc_enable, [125](#)
 - icc_exchangeAPDU, [126](#)
 - icc_exchangeEncryptedAPDU, [126](#)
 - icc_getAPDU_KSN, [127](#)
 - icc_getFunctionStatus, [127](#)
 - icc_getICCReaderStatus, [128](#)
 - icc_getKeyFormatForICCDUKPT, [128](#)
 - icc_getKeyTypeForICCDUKPT, [128](#)
 - icc_getSetting, [129](#)
 - icc_getSettings, [129](#)
 - icc_powerOffICC, [129](#)
 - icc_powerOnICC, [129](#)
 - icc_setKeyFormatForICCDUKPT, [130](#)
 - icc_setKeyTypeForICCDUKPT, [130](#)
 - icc_setSetting, [130, 131](#)
 - lcd_retrieveMessage, [131](#)
 - msr_RetrieveWhiteList, [135](#)
 - msr_cancelMSRSwipe, [131](#)
 - msr_captureMode, [132](#)
 - msr_disable, [132](#)
 - msr_getClearPANID, [132](#)
 - msr_getExpirationMask, [133](#)
 - msr_getFunctionStatus, [133](#)
 - msr_getSetting, [133](#)
 - msr_getSettings, [134](#)
 - msr_getSwipeEncryption, [134](#)
 - msr_getSwipeForcedEncryptionOption, [134](#)

- msr_getSwipeMaskOption, 135
- msr_setClearPANID, 135
- msr_setExpirationMask, 136
- msr_setSetting, 136
- msr_setSettings, 136
- msr_setSwipeEncryption, 137
- msr_setSwipeForcedEncryptionOption, 137
- msr_setSwipeMaskOption, 137
- msr_setWhiteList, 138
- msr_setWhiteListFromBDK, 138
- msr_startMSRSwipe, 138
- msr_switchUSBInterfaceMode, 139
- SDK_Version, 139
- setCallback, 140
- setCommandTimeout, 140
- SharedController, 140
- IDTechSDK::IDT_BTMag
 - config_getEncryptionControl, 142
 - config_getLEDController, 143
 - config_getSerialNumber, 143
 - config_setCmdTimeOutDuration, 143
 - config_setEncryptionControl, 144
 - config_setLEDController, 144
 - device_RemoteKeyInjection, 159
 - device_controlBeep, 145
 - device_controlLED_ICC, 146
 - device_controlLED_MSR, 146
 - device_controlLED, 145
 - device_getAnyRKIStatus, 147
 - device_getConfigurationFromMemory, 147
 - device_getFirmwareVersion, 148
 - device_getKeyStatus, 148
 - device_getRKIStatus, 158
 - device_getResponseCodeString, 148
 - device_readConfigurationToMemory, 159
 - device_rebootDevice, 159
 - device_sendConfiguration, 160
 - device_sendConfigurationFromZip, 161
 - device_sendDataCommand, 161
 - device_sendPAE, 162
 - device_setDateTime, 162
 - device_updateFirmwareFromZip, 162
 - getCommandTimeout, 163
 - msr_RetrieveWhiteList, 168
 - msr_cancelMSRSwipe, 163
 - msr_captureMode, 163
 - msr_disable, 165
 - msr_getClearPANID, 165
 - msr_getExpirationMask, 165
 - msr_getFunctionStatus, 166
 - msr_getSetting, 166
 - msr_getSettings, 166
 - msr_getSwipeEncryption, 167
 - msr_getSwipeForcedEncryptionOption, 167
 - msr_getSwipeMaskOption, 167
 - msr_setClearPANID, 168
 - msr_setExpirationMask, 168
 - msr_setSetting, 169
 - msr_setSettings, 169
 - msr_setSwipeEncryption, 169
 - msr_setSwipeForcedEncryptionOption, 170
 - msr_setSwipeMaskOption, 170
 - msr_setWhiteList, 170
 - msr_startMSRSwipe, 171
 - msr_switchUSBInterfaceMode, 171
 - SDK_Version, 172
 - setCallback, 172
 - setCommandTimeout, 172
 - SharedController, 174
 - useSerialPort, 173
 - useSerialPortLinux, 173, 174
- IDTechSDK::IDT_BTPay
 - config_getBeeperController, 177
 - config_getEncryptionControl, 177
 - config_getLEDController, 178
 - config_getModelNumber, 178
 - config_getSerialNumber, 178
 - config_setBeeperController, 179
 - config_setCmdTimeOutDuration, 179
 - config_setEncryptionControl, 179
 - config_setLEDController, 180
 - device_RemoteKeyInjection, 195
 - device_controlBeep, 180
 - device_controlLED_ICC, 181
 - device_controlLED_MSR, 182
 - device_controlLED, 181
 - device_getAnyRKIStatus, 182
 - device_getConfigurationFromMemory, 182
 - device_getFirmwareVersion, 183
 - device_getKeyStatus, 183
 - device_getRKIStatus, 193
 - device_getResponseCodeString, 184
 - device_readConfigurationToMemory, 194
 - device_rebootDevice, 194
 - device_sendConfiguration, 195
 - device_sendConfigurationFromZip, 196
 - device_sendDataCommand, 197
 - device_sendPAE, 197
 - device_setDateTime, 197
 - device_startRKI, 199
 - emv_addTerminalData, 199
 - emv_allowFallback, 199
 - emv_authenticateTransaction, 199
 - emv_autoAuthenticate, 200
 - emv_autoAuthenticateTags, 200
 - emv_callbackResponseLCD, 200
 - emv_callbackResponseMSR, 201
 - emv_callbackResponsePIN, 201
 - emv_cancelTransaction, 202
 - emv_completeTransaction, 202
 - emv_getEMVConfigurationCheckValue, 202
 - emv_getEMVKernelCheckValue, 203
 - emv_getEMVKernelVersion, 203
 - emv_removeAllApplicationData, 203
 - emv_removeAllCAPK, 204
 - emv_removeAllCRL, 204

- emv_removeApplicationData, 204
- emv_removeCAPK, 204
- emv_removeCRL, 205
- emv_removeTerminalData, 205
- emv_retrieveAIDList, 205
- emv_retrieveApplicationData, 206
- emv_retrieveCAPKList, 206
- emv_retrieveCAPK, 206
- emv_retrieveCRLList, 207
- emv_retrieveTerminalData, 207
- emv_retrieveTransactionResult, 207
- emv_setApplicationData, 208
- emv_setCAPK, 208
- emv_setCRL, 209
- emv_setTerminalData, 209
- emv_setTerminalMajorConfiguration, 209
- emv_startTransaction, 209
- emv_trySetTerminalData, 210
- getCommandTimeout, 210
- icc_disable, 211
- icc_enable, 211
- icc_exchangeAPDU, 211
- icc_exchangeEncryptedAPDU, 212
- icc_getAPDU_KSN, 212
- icc_getFunctionStatus, 213
- icc_getICCReaderStatus, 213
- icc_getKeyFormatForICCDUKPT, 213
- icc_getKeyTypeForICCDUKPT, 214
- icc_powerOffICC, 214
- icc_powerOnICC, 214
- icc_setKeyFormatForICCDUKPT, 215
- icc_setKeyTypeForICCDUKPT, 215
- msr_RetrieveWhiteList, 219
- msr_cancelMSRSwipe, 215
- msr_captureMode, 216
- msr_disable, 216
- msr_getClearPANID, 216
- msr_getExpirationMask, 217
- msr_getFunctionStatus, 217
- msr_getSetting, 218
- msr_getSettings, 218
- msr_getSwipeEncryption, 218
- msr_getSwipeForcedEncryptionOption, 218
- msr_getSwipeMaskOption, 219
- msr_setClearPANID, 220
- msr_setExpirationMask, 220
- msr_setSetting, 220
- msr_setSettings, 220
- msr_setSwipeEncryption, 221
- msr_setSwipeForcedEncryptionOption, 221
- msr_setSwipeMaskOption, 221
- msr_setWhiteList, 222
- msr_startMSRSwipe, 222
- msr_switchUSBInterfaceMode, 222, 223
- SDK_Version, 223
- setCallback, 223, 224
- setCommandTimeout, 224
- SharedController, 225
- useSerialPort, 224
- useSerialPortLinux, 225
- IDTechSDK::IDT_CM100
 - ctls_cancelTransaction, 226
 - ctls_enableL2Application, 227
 - ctls_startTransaction, 227
 - device_RemoteKeyInjection, 229
 - device_getAnyRKIStatus, 227
 - device_getConfigurationFromMemory, 228
 - device_getFirmwareVersion, 228
 - device_getRKIStatus, 228
 - device_getVersion, 228
 - device_readConfigurationToMemory, 229
 - device_sendBeep, 230
 - device_sendConfiguration, 230
 - device_sendConfigurationFromZip, 231
 - device_sendDataCommand, 232
 - device_sendPAE, 232
 - device_setDateTime, 233
 - getCommandTimeout, 233
 - SDK_Version, 233
 - setCallback, 233
 - setCommandTimeout, 234
 - SharedController, 234
- IDTechSDK::IDT_K100
 - closeSerialPort, 235
 - closeUSB, 235
 - config_getBaudRate, 235
 - config_getModelNumber, 236
 - config_setBaudRate, 236
 - config_setCmdTimeOutDuration, 236
 - device_RemoteKeyInjection, 240
 - device_StartRKI, 242
 - device_getAnyRKIStatus, 237
 - device_getConfigurationFromMemory, 237
 - device_getDateTimeString, 237
 - device_getFirmwareVersion, 238
 - device_getKeyStatus, 238
 - device_getRKIStatus, 238
 - device_readConfigurationToMemory, 239
 - device_rebootDevice, 239
 - device_sendConfiguration, 240
 - device_sendConfigurationFromZip, 241
 - device_sendDataCommand, 242
 - device_sendDataCommand_ext, 242
 - device_startRKI, 243
 - device_updateFirmwareFromZip, 243
 - getCommandTimeout, 243
 - pin_cancelPINEntry, 244
 - pin_enableKeypad, 244
 - pin_getEncryptedPIN, 244
 - pin_getFunctionKey, 244
 - pin_sendBeep, 245
 - SDK_Version, 245
 - setCallback, 245, 246
 - setCommandTimeout, 246
 - SharedController, 247
 - useSerialPort, 246

- useSerialPortLinux, [247](#)
- IDTechSDK::IDT_KioskIII
 - config_checkDUKPTKey, [250](#)
 - config_getDEKVariantType, [250](#)
 - config_getDUKPT_DEK_Attribution, [251](#)
 - config_getDUKPT_KSN, [251](#)
 - config_getDUKPTEncryptionType, [251](#)
 - config_getKeyslot_PEK_DEK, [252](#)
 - config_getSalt_KCV, [252](#)
 - config_getSerialNumber, [252](#)
 - config_setBaudRate, [253](#)
 - config_setDEKVariantType, [253](#)
 - config_setDUKPT_DEK_Attribution_AES, [254](#)
 - config_setDUKPT_DEK_Attribution_TDES, [254](#)
 - config_setDUKPTEncryptionType, [255](#)
 - config_setKeyslot_PEK_DEK, [255](#)
 - config_setRKLLKeys, [255](#)
 - ctls_activateTransaction, [256](#)
 - ctls_cancelTransaction, [257](#)
 - ctls_getAllConfigurationGroups, [257](#)
 - ctls_getConfigurationGroup, [258](#)
 - ctls_nfcCommand, [258](#)
 - ctls_removeAllCAPK, [259](#)
 - ctls_removeApplicationData, [260](#)
 - ctls_removeCAPK, [260](#)
 - ctls_removeConfigurationGroup, [260](#)
 - ctls_retrieveAIDList, [260](#)
 - ctls_retrieveApplicationData, [261](#)
 - ctls_retrieveCAPKList, [262](#)
 - ctls_retrieveCAPK, [261](#)
 - ctls_retrieveTerminalData, [262](#)
 - ctls_setApplicationData, [263](#)
 - ctls_setCAPK, [263](#)
 - ctls_setConfigurationGroup, [263](#)
 - ctls_setDefaultConfiguration, [264](#)
 - ctls_setTerminalData, [264](#)
 - ctls_startTransaction, [264](#)
 - ctls_trySetTerminalData, [266](#)
 - ctls_updateBalance, [266](#)
 - device_RemoteKeyInjection, [273](#)
 - device_StartRKI, [279](#)
 - device_activateTransaction, [267](#)
 - device_cancelTransaction, [268](#)
 - device_controlLED, [268](#)
 - device_controlUserInterface, [269](#)
 - device_enablePassThrough, [269](#)
 - device_getAnyRKIStatus, [270](#)
 - device_getConfigurationFromMemory, [270](#)
 - device_getFirmwareVersion, [270](#)
 - device_getMerchantRecord, [270](#)
 - device_getRKIStatus, [271](#)
 - device_getTransactionResults, [271](#)
 - device_pingDevice, [271](#)
 - device_pollForToken, [272](#)
 - device_readConfigurationToMemory, [272](#)
 - device_rebootDevice, [273](#)
 - device_retrieveAIDList, [274](#)
 - device_sendConfiguration, [274](#)
 - device_sendConfigurationFromZip, [275](#)
 - device_sendDataCommand, [276](#)
 - device_sendDataCommand_ext, [276](#)
 - device_sendPAE, [276](#)
 - device_sendVivoCommandP2, [277](#)
 - device_sendVivoCommandP2_ext, [277](#)
 - device_setBurstMode, [278](#)
 - device_setMerchantRecord, [278](#)
 - device_setPollMode, [278](#)
 - device_startRKI, [279](#)
 - device_startTransaction, [279](#)
 - device_updateDeviceFirmware, [280](#)
 - device_updateFirmwareFromZip, [282](#)
 - felica_SendCommand, [284](#)
 - felica_authentication, [282](#)
 - felica_read, [282](#)
 - felica_readWithMac, [283](#)
 - felica_requestService, [283](#)
 - felica_write, [284](#)
 - felica_writeWithMac, [284](#)
 - getCommandTimeout, [285](#)
 - lcd_retrieveMessage, [285](#)
 - SDK_Version, [285](#)
 - setCallback, [285](#)
 - setCommandTimeout, [286](#)
 - SharedController, [287](#)
 - useSerialPort, [286](#)
 - useSerialPortLinux, [286](#), [287](#)
- IDTechSDK::IDT_L100
 - closeSerialPort, [289](#)
 - closeUSB, [289](#)
 - config_getBaudRate, [289](#)
 - config_getEthernetMACAddress, [290](#)
 - config_getModelNumber, [290](#)
 - config_getNetworkConfiguration, [290](#)
 - config_getSerialNumber, [291](#)
 - config_setBaudRate, [291](#)
 - config_setCmdTimeOutDuration, [291](#)
 - config_setEthernetMACAddress, [292](#)
 - config_setNetworkConfiguration, [292](#)
 - device_RemoteKeyInjection, [296](#)
 - device_StartRKI, [300](#)
 - device_enterStopMode, [292](#)
 - device_getAnyRKIStatus, [293](#)
 - device_getConfigurationFromMemory, [293](#)
 - device_getDateTime, [293](#)
 - device_getFirmwareVersion, [294](#)
 - device_getKeyStatus, [294](#)
 - device_getRKIStatus, [294](#)
 - device_readConfigurationToMemory, [295](#)
 - device_rebootDevice, [295](#)
 - device_sendConfiguration, [296](#)
 - device_sendConfigurationFromZip, [297](#)
 - device_sendDataCommand, [298](#)
 - device_sendDataCommand_ext, [298](#)
 - device_sendPAE, [299](#)
 - device_setDateTime, [299](#)
 - device_setSleepModeTime, [299](#)

- device_startRKI, 300
- device_updateDeviceFirmware, 300
- device_updateFirmwareFromZip, 301
- getCommandTimeout, 303
- lcd_clearAllLines, 303
- lcd_clearDisplay, 303
- lcd_displayMessage, 303
- lcd_displayPrompt, 304
- lcd_enableBacklight, 304
- lcd_getBacklightStatus, 304
- lcd_retrieveMessage, 305
- lcd_savePrompt, 305
- pin_cancelPINEntry, 305
- pin_getEncryptedPIN, 306
- pin_getFunctionKey, 306
- pin_getManualPanEntry, 308
- pin_promptForAmountInput, 308
- pin_promptForKeyInput, 311, 313
- pin_sendBeep, 315
- pin_setKeypressCapture, 316
- SDK_Version, 316
- setCallback, 316
- setCommandTimeout, 317
- SharedController, 318
- useSerialPort, 317
- useSerialPortLinux, 317, 318
- IDTechSDK::IDT_L80
 - closeSerialPort, 320
 - closeUSB, 320
 - config_getBaudRate, 320
 - config_getModelNumber, 321
 - config_getSerialNumber, 321
 - config_setBaudRate, 321
 - config_setCmdTimeOutDuration, 322
 - device_PKI_RKI, 324
 - device_RemoteKeyInjection, 325
 - device_enterStopMode, 322
 - device_getAnyRKIStatus, 322
 - device_getConfigurationFromMemory, 322
 - device_getDateTime, 323
 - device_getFirmwareVersion, 323
 - device_getKeyStatus, 323
 - device_getRKIStatus, 324
 - device_readConfigurationToMemory, 324
 - device_rebootDevice, 325
 - device_sendConfiguration, 326
 - device_sendConfigurationFromZip, 327
 - device_sendDataCommand, 327
 - device_sendDataCommand_ext, 328
 - device_sendPAE, 328
 - device_setDateTime, 329
 - device_setSleepModeTime, 329
 - device_updateDeviceFirmware, 329
 - device_updateFirmwareFromZip, 330
 - getCommandTimeout, 331
 - lcd_clearAllLines, 331
 - lcd_clearDisplay, 331
 - lcd_displayMessage, 332
 - lcd_displayPrompt, 332
 - lcd_enableBacklight, 332
 - lcd_getBacklightStatus, 332
 - lcd_retrieveMessage, 333
 - lcd_savePrompt, 333
 - pin_cancelPINEntry, 333
 - pin_getEncryptedPIN, 334
 - pin_getFunctionKey, 334
 - pin_getManualPanEntry, 335
 - pin_promptForAmountInput, 335
 - pin_promptForKeyInput, 336
 - pin_sendBeep, 337
 - pin_setKeypressCapture, 337
 - SDK_Version, 337
 - setCallback, 337, 338
 - setCommandTimeout, 338
 - SharedController, 339
 - useSerialPort, 338
 - useSerialPortLinux, 339
- IDTechSDK::IDT_MiniSmartII
 - config_getModelNumber, 342
 - config_getSerialNumber, 342
 - config_setCmdTimeOutDuration, 342
 - createFastEMVData, 342
 - device_RemoteKeyInjection, 357
 - device_StartRKI, 360
 - device_activateTransaction, 343
 - device_enableSecureHeadForMSII, 343
 - device_getAnyRKIStatus, 344
 - device_getConfigurationFromMemory, 344
 - device_getDateTime, 344
 - device_getFirmwareVersion, 345
 - device_getKeyTypeForICCDUKPT, 345
 - device_getRKIStatus, 355
 - device_getResponseCodeString, 345
 - device_readConfigurationToMemory, 356
 - device_rebootDevice, 356
 - device_sendConfiguration, 357
 - device_sendConfigurationFromZip, 358
 - device_sendDataCommand, 359
 - device_sendDataCommand_ext, 359
 - device_sendPAE, 359
 - device_setDateTime, 360
 - device_startRKI, 360
 - device_startTransaction, 361
 - device_updateDeviceFirmware, 361
 - device_updateFirmwareFromZip, 362
 - emv_activateTransaction, 363
 - emv_addTerminalData, 363
 - emv_allowFallback, 364
 - emv_authenticateTransaction, 364
 - emv_autoAuthenticate, 364
 - emv_autoAuthenticateTags, 365
 - emv_callbackResponseLCD, 365
 - emv_callbackResponseMSR, 365
 - emv_callbackResponsePIN, 365
 - emv_cancelTransaction, 366
 - emv_completeTransaction, 366

- emv_getEMVConfigurationCheckValue, 367
- emv_getEMVKernelCheckValue, 367
- emv_getEMVKernelVersion, 367
- emv_removeAllApplicationData, 368
- emv_removeAllCAPK, 368
- emv_removeAllCRL, 368
- emv_removeApplicationData, 368
- emv_removeCAPK, 369
- emv_removeCRL, 369
- emv_removeTerminalData, 369
- emv_retrieveAIDList, 370
- emv_retrieveApplicationData, 370
- emv_retrieveCAPKList, 371
- emv_retrieveCAPK, 370
- emv_retrieveCRLList, 371
- emv_retrieveTerminalData, 372
- emv_retrieveTransactionResult, 372
- emv_setApplicationData, 372
- emv_setCAPK, 372
- emv_setCRL, 373
- emv_setTerminalData, 373
- emv_setTerminalMajorConfiguration, 374
- emv_startTransaction, 374
- emv_trySetTerminalData, 375
- getCommandTimeout, 375
- icc_exchangeAPDU, 375
- icc_getAPDU_KSN, 376
- icc_getClearPANID, 376
- icc_getICCRReaderStatus, 376
- icc_getKeyFormatForICCDUKPT, 376
- icc_getKeyTypeForICCDUKPT, 377
- icc_getReadingCharacteristics, 377
- icc_powerOffICC, 378
- icc_powerOnICC, 378
- icc_setClearPANID, 378
- icc_setKeyFormatForICCDUKPT, 379
- icc_setKeyTypeForICCDUKPT, 379
- icc_setReadingCharacteristics, 379
- lcd_retrieveMessage, 380
- SDK_Version, 380
- setCallback, 380, 381
- setCommandTimeout, 381
- SharedController, 382
- useSerialPort, 381
- useSerialPortLinux, 382
- IDTechSDK::IDT_NEO2
 - adf_ApplicationControl, 390
 - adf_eraseFlash, 391
 - adf_getADFMMode, 391
 - adf_getModuleBytes, 391
 - adf_getModuleInfo, 391
 - adf_setADFMMode, 392
 - adf_setJTAG, 392
 - closeSocket, 392
 - config_checkDUKPTKey, 393
 - config_getDEKVariantType, 393
 - config_getDUKPT_DEK_Attribution, 394
 - config_getDUKPT_KSN, 394
 - config_getDUKPTEncryptionType, 394
 - config_getEncryptionControl, 395
 - config_getEthernetMACAddress, 395
 - config_getKeySlotInfo, 396
 - config_getKeyslot_PEK_DEK, 396
 - config_getMasking, 396
 - config_getNetworkConfiguration, 397
 - config_getSSLServerEthernet, 398
 - config_getSalt_KCV, 397
 - config_getSerialNumber, 398
 - config_getSwipeandDone, 398
 - config_getTrackFormat, 399
 - config_getWhiteList, 399
 - config_getWifiConfig, 399
 - config_getWirelessWorkMode, 400
 - config_sendSSLRequestWiFi, 400
 - config_setBaudRate, 400
 - config_setBluetoothParameters, 401
 - config_setDEKVariantType, 401
 - config_setDUKPT_DEK_Attribution_AES, 401
 - config_setDUKPT_DEK_Attribution_TDES, 402
 - config_setDUKPTEncryptionType, 402
 - config_setEncryptionControl, 403
 - config_setKeyslot_PEK_DEK, 403
 - config_setMasking, 404
 - config_setNetworkConfiguration, 404
 - config_setRKLLKeys, 404
 - config_setSSLServerEthernet, 405
 - config_setSwipeandDone, 405
 - config_setTrackFormat, 406
 - config_setWhiteList, 406
 - config_setWifiConfig, 406
 - config_setWirelessWorkMode, 407
 - createFastEMVData, 407
 - ctls_activateTransaction, 407
 - ctls_cancelTransaction, 409
 - ctls_getAllConfigurationGroups, 409
 - ctls_getConfigurationGroup, 409
 - ctls_nfcCommand, 410
 - ctls_removeAllCAPK, 411
 - ctls_removeApplicationData, 411
 - ctls_removeCAPK, 411
 - ctls_removeConfigurationGroup, 411
 - ctls_resetConfigurationGroup, 412
 - ctls_retrieveAIDList, 412
 - ctls_retrieveApplicationData, 412
 - ctls_retrieveCAPKList, 413
 - ctls_retrieveCAPK, 413
 - ctls_retrieveTerminalData, 413
 - ctls_setApplicationData, 414
 - ctls_setCAPK, 414
 - ctls_setConfigurationGroup, 415
 - ctls_setDefaultConfiguration, 415
 - ctls_setTerminalData, 416
 - ctls_startTransaction, 416
 - ctls_trySetTerminalData, 417
 - ctls_updateBalance, 418
 - device_RemoteKeyInjection, 451

- device_StartRKI, [463](#)
- device_activateTransaction, [418](#)
- device_buzzer, [419](#)
- device_buzzerOnOff, [420](#)
- device_cancelTransaction, [420](#)
- device_certificateType, [420](#)
- device_controlLED, [420](#)
- device_controlUserInterface, [421](#)
- device_deleteDirectory, [422](#)
- device_deleteFile, [423](#)
- device_disBlueLED, [423](#)
- device_enaBlueLED, [424](#)
- device_enableL100PassThrough, [424](#)
- device_enableL80PassThrough, [424](#)
- device_enablePassThrough, [424](#)
- device_enterStandbyMode, [425](#)
- device_extendedErrorCondition, [425](#)
- device_get1050BootloaderVersion, [425](#)
- device_get1050FuseStatus, [426](#)
- device_getAnyRKIStatus, [426](#)
- device_getAudioVolume, [426](#)
- device_getBLEName, [427](#)
- device_getBatteryVoltage, [427](#)
- device_getBootloaderVersion, [428](#)
- device_getCashTranRiskPara, [428](#)
- device_getConfigurationFromMemory, [428](#)
- device_getDRS, [429](#)
- device_getDeviceTime, [428](#)
- device_getDeviceTreeVersion, [429](#)
- device_getFirmwareVersion, [429](#)
- device_getHardwareInfor, [430](#)
- device_getLightSensorVal, [430](#)
- device_getMerchantRecord, [430](#)
- device_getModuleVer, [431](#)
- device_getMsrSecurePar, [431](#)
- device_getProcessorType, [431](#)
- device_getProductType, [432](#)
- device_getRKIStatus, [442](#)
- device_getRT1050FirmwareVersion, [443](#)
- device_getRemoteKeyInjectionTO, [432](#)
- device_getResponseCodeString, [432](#)
- device_getSelfCheckTime, [443](#)
- device_getTransArmorID, [443](#)
- device_getTransactionResults, [443](#)
- device_getUIDofMCU, [444](#)
- device_getUsbBootLoader, [444](#)
- device_listDirectory, [444](#)
- device_listenForNotifications, [445](#)
- device_loadCertCA, [445](#)
- device_logClear, [445](#)
- device_logEnable, [446](#)
- device_logRead, [446](#)
- device_lowPowerMode, [446](#)
- device_offYellowLED, [447](#)
- device_onYellowLED, [447](#)
- device_pingDevice, [447](#)
- device_playAudio, [447](#)
- device_pollForToken, [448](#)
- device_queryFile, [448](#)
- device_readConfigurationToMemory, [449](#)
- device_readFileFromSD, [450](#)
- device_rebootDevice, [450](#)
- device_resetConfigurationGroup, [451](#)
- device_resetNVM, [451](#)
- device_resetTransaction, [451](#)
- device_retrieveAIDList, [452](#)
- device_retrieveTerminalData, [452](#)
- device_rrcConnect, [452](#)
- device_rrcDisconnect, [453](#)
- device_rrcDownloadApp, [453](#)
- device_rrcInstallApp, [453](#)
- device_rrcRunApp, [454](#)
- device_rrcUninstallApp, [454](#)
- device_sendConfiguration, [454](#)
- device_sendConfigurationFromZip, [455](#)
- device_sendDataCommand, [456](#)
- device_sendDataCommand_ext, [457](#)
- device_sendPAE, [457](#)
- device_sendVivoCommandP2, [457](#)
- device_sendVivoCommandP2_ext, [458](#)
- device_sendVivoCommandP3, [458](#)
- device_sendVivoCommandP3_ext, [459](#)
- device_sendVivoCommandP4, [459](#)
- device_sendVivoCommandP4_ext, [459](#)
- device_setAudioVolume, [460](#)
- device_setBurstMode, [460](#)
- device_setLED, [460](#)
- device_setMerchantRecord, [461](#)
- device_setPollMode, [461](#)
- device_setSelfCheckTime, [461](#)
- device_setTerminalData, [462](#)
- device_setTransArmorEncryption, [462](#)
- device_setTransArmorID, [462](#)
- device_startRKI, [463](#)
- device_startTransaction, [463](#)
- device_startTransactionCB, [465](#)
- device_stopAudio, [466](#)
- device_terminalInfo, [466](#)
- device_transferFile, [467](#)
- device_updateDeviceFirmware, [467](#)
- device_updateDeviceFromManifest, [469](#)
- device_updateFirmwareFromZip, [469](#)
- device_updateFirmwareIP, [469](#)
- device_updateFirmwareKernels, [471](#)
- device_updateFirmwareType, [472, 473](#)
- device_wakeDevice, [475](#)
- emv_activateTransaction, [476](#)
- emv_addTerminalData, [476](#)
- emv_allowFallback, [476](#)
- emv_authenticateTransaction, [477](#)
- emv_authenticateTransactionCB, [477](#)
- emv_autoAuthenticate, [477](#)
- emv_autoAuthenticateTags, [478](#)
- emv_callbackResponseKSN, [478](#)
- emv_callbackResponseLCD, [478](#)
- emv_callbackResponseMSR, [479](#)

- emv_callbackResponsePIN_ETC, 479
- emv_callbackResponsePIN, 479
- emv_cancelTransaction, 480
- emv_completeTransaction, 480
- emv_completeTransactionCB, 481
- emv_exchangeCerts, 481
- emv_generateDUKPT, 482
- emv_getEMVConfigurationCheckValue, 482
- emv_getEMVKernelCheckValue, 482
- emv_getEMVKernelVersion, 482
- emv_getEMVKernelVersionExt, 483
- emv_getTerminalMajorConfiguration, 483
- emv_removeAllApplicationData, 483
- emv_removeAllCAPK, 484
- emv_removeAllCRL, 484
- emv_removeApplicationData, 484
- emv_removeCAPK, 484
- emv_removeCRL, 485
- emv_removeTerminalData, 485
- emv_resetConfigurationGroup, 485
- emv_retrieveAIDList, 486
- emv_retrieveApplicationData, 486
- emv_retrieveCAPKList, 487
- emv_retrieveCAPK, 486
- emv_retrieveCRLList, 487
- emv_retrieveTerminalData, 488
- emv_retrieveTransactionResult, 488
- emv_setApplicationData, 488, 489
- emv_setCAPK, 489
- emv_setCRL, 490
- emv_setTerminalData, 490
- emv_setTerminalMajorConfiguration, 491
- emv_startTransaction, 491
- emv_startTransactionCB, 491
- emv_trySetTerminalData, 492
- emv_verifyDUKPTLoaded, 493
- felica_SendCommand, 494
- felica_authentication, 493
- felica_read, 493
- felica_readWithMac, 494
- felica_requestService, 494
- felica_write, 495
- felica_writeWithMac, 495
- getCommandTimeout, 495
- getLastErrorMessage, 496
- icc_exchangeAPDU, 496
- icc_getICCRReaderStatus, 496
- icc_powerOffICC, 496
- icc_powerOnICC, 497
- ip_autoConnectToSocket, 497
- ip_connectToSocket, 497
- ip_firstConnectToSocket, 498
- ip_getSocketList, 498
- ip_isConnected, 499
- ip_monitorSocketConnectionStatus, 499
- ip_switchToSocket, 500
- lcd_addButton, 500
- lcd_addEthernet, 501
- lcd_addImage, 501
- lcd_addLED, 502
- lcd_addText, 503
- lcd_clearAllLines, 505
- lcd_clearDisplay, 506
- lcd_clearScreenInfo, 506
- lcd_cloneScreen, 506
- lcd_createScreen, 506
- lcd_destroyScreen, 507
- lcd_displayMessage, 507
- lcd_getActiveScreen, 507
- lcd_getAllObjects, 508
- lcd_getAllScreens, 508
- lcd_getAudioVolume, 508
- lcd_getButtonEvent, 509
- lcd_linkUIWithTransactionMessageId, 509
- lcd_loadScreenInfo, 510
- lcd_playAudio, 510
- lcd_queryObjectbyID, 510
- lcd_queryObjectbyName, 511
- lcd_queryScreenbyID, 511
- lcd_queryScreenbyName, 511
- lcd_removeItem, 512
- lcd_retrieveMessage, 512
- lcd_setAudioVolume, 512
- lcd_setBacklight, 514
- lcd_setButtonCallback, 514
- lcd_setPinCancelPromptCallback, 514
- lcd_setPinFailureCallback, 515
- lcd_setPinInputCallback, 515
- lcd_setPinSwipeCallback, 515
- lcd_setPinTimeoutCallback, 515
- lcd_showScreen, 516
- lcd_startCameraCapture, 516
- lcd_startScanQR_ext, 517
- lcd_startScanQR, 516
- lcd_startScreenSaver, 517
- lcd_stopAudio, 517
- lcd_stopCameraCapture, 518
- lcd_stopScanQR, 518
- lcd_storeScreenInfo, 518
- lcd_updateColor, 519
- lcd_updateLabel, 519
- lcd_updatePosition, 520
- monitorNetworkForDevices, 520
- msr_cancelMSRSwipe, 521
- msr_getConfiguration, 521
- msr_getMSRTrack, 521
- msr_setConfiguration, 522
- msr_setMSRTrack, 522
- msr_startMSRSwipe, 523
- msr_startMSRSwipe_ext, 523
- pin_cancelPINEntry, 524
- pin_capturePin, 524
- pin_capturePin_ext, 525
- pin_capturePinExt, 526
- pin_getFunctionKey, 527
- pin_getFunctionKey_ext, 528

- pin_getPanEntry, 528
- pin_getPanEntry_ext, 529
- pin_promptForAmount, 530
- pin_promptForAmount_ext, 530
- pin_promptForInput, 531
- pin_promptForInput_ext, 531
- pin_promptForNumericKeyWithSwipe, 532
- pin_promptForNumericKeyWithSwipe_ext, 533
- pin_sendBeep, 534
- SDK_Version, 534
- setCallback, 534
- setCallbackIP, 534
- setCommandTimeout, 535
- setLongPressCallback, 535
- SharedController, 536
- useSerialPort, 535, 536
- useSerialPortLinux, 536
- IDTechSDK::IDT_PIP
 - config_getSerialNumber, 538
 - ctls_activateTransaction, 539
 - ctls_cancelTransaction, 540
 - ctls_getAllConfigurationGroups, 540
 - ctls_getConfigurationGroup, 540
 - ctls_nfcCommand, 541
 - ctls_removeApplicationData, 542
 - ctls_removeConfigurationGroup, 543
 - ctls_retrieveAIDList, 543
 - ctls_retrieveApplicationData, 543
 - ctls_retrieveTerminalData, 544
 - ctls_setApplicationData, 544
 - ctls_setConfigurationGroup, 544
 - ctls_setDefaultConfiguration, 545
 - ctls_setTerminalData, 545
 - ctls_startTransaction, 545
 - ctls_trySetTerminalData, 546
 - ctls_updateBalance, 547
 - device_RemoteKeyInjection, 553
 - device_activateTransaction, 547
 - device_controlUserInterface, 548
 - device_enablePassThrough, 549
 - device_getAnyRKIStatus, 550
 - device_getConfigurationFromMemory, 550
 - device_getFirmwareVersion, 550
 - device_getMerchantRecord, 550
 - device_getPIPMODE, 551
 - device_getRKIStatus, 551
 - device_getTransactionResults, 551
 - device_pingDevice, 552
 - device_pollForToken, 552
 - device_readConfigurationToMemory, 553
 - device_retrieveAIDList, 554
 - device_sendConfiguration, 554
 - device_sendConfigurationFromZip, 555
 - device_sendDataCommand, 556
 - device_sendDataCommand_ext, 556
 - device_sendVivoCommandP2, 557
 - device_sendVivoCommandP2_ext, 557
 - device_setBurstMode, 557
 - device_setMerchantRecord, 558
 - device_setPIPMODE, 558
 - device_setPollMode, 558
 - device_startTransaction, 559
 - device_updateDeviceFirmware, 560
 - device_updateFirmwareFromZip, 561
 - felica_requestService, 562
 - getCommandTimeout, 562
 - lcd_retrieveMessage, 562
 - SDK_Version, 562
 - setCallback, 563
 - setCommandTimeout, 563
 - SharedController, 563
- IDTechSDK::IDT_SREDKey2
 - config_getModelNumber, 661
 - config_getSerialNumber, 661
 - config_getStatusKeySlots, 661
 - device_RemoteKeyInjection, 675
 - device_certificateType, 662
 - device_enableAdminKey, 662
 - device_getAnyRKIStatus, 662
 - device_getConfigurationFromMemory, 663
 - device_getFirmwareVersion, 663
 - device_getOutputType, 663
 - device_getRKIStatus, 673
 - device_getResponseCodeString, 664
 - device_getTransArmorID, 674
 - device_readConfigurationToMemory, 674
 - device_rebootDevice, 675
 - device_sendConfiguration, 675
 - device_sendConfigurationFromZip, 676
 - device_sendDataCommand, 677
 - device_sendDataCommand_ext, 677
 - device_sendVivoCommandP2, 678
 - device_sendVivoCommandP2_ext, 678
 - device_sendVivoCommandP3, 679
 - device_sendVivoCommandP3_ext, 679
 - device_sendVivoCommandP4, 679
 - device_sendVivoCommandP4_ext, 680
 - device_setLanguage, 680
 - device_setOutputType, 680
 - device_setPollMode, 681
 - device_setTransArmorEncryption, 681
 - device_setTransArmorID, 681
 - device_startRKI, 682
 - device_updateDeviceFirmware, 682
 - device_updateDeviceFromManifest, 683
 - device_updateFirmwareFromZip, 684
 - device_updateFirmwareType, 684
 - getCommandTimeout, 686
 - lcd_retrieveMessage, 686
 - msr_cancelTransaction, 687
 - msr_disable, 687
 - msr_enable, 687
 - msr_getClearPANID, 688
 - msr_getConfiguration, 688
 - msr_getExpirationMask, 688
 - msr_getFunctionStatus, 689

- msr_getSwipeEncryption, 689
- msr_getSwipeForcedEncryptionOption, 689
- msr_getSwipeMaskOption, 690
- msr_getTerminalData, 690
- msr_resetTerminalData, 690
- msr_setClearPANID, 691
- msr_setConfiguration, 691
- msr_setExpirationMask, 691
- msr_setSwipeForcedEncryptionOption, 692
- msr_setSwipeMaskOption, 692
- msr_setTerminalData, 692
- msr_startKeyedTransaction, 693
- msr_startSwipeTransaction, 693
- msr_switchUSBInterfaceMode, 693, 694
- SDK_Version, 694
- setCallback, 694, 695
- setCommandTimeout, 695
- SharedController, 695
- IDTechSDK::IDT_SecureKey
 - config_getModelNumber, 565
 - config_getSerialNumber, 565
 - device_RemoteKeyInjection, 578
 - device_StartRKI, 583
 - device_capturePINFromLast12, 565
 - device_getAnyRKIStatus, 565
 - device_getConfigurationFromMemory, 566
 - device_getFirmwareVersion, 566
 - device_getOutputType, 566
 - device_getRKIStatus, 577
 - device_getResponseCodeString, 567
 - device_readConfigurationToMemory, 577
 - device_sendConfiguration, 578
 - device_sendConfigurationFromZip, 579
 - device_sendDataCommand, 580
 - device_sendDataCommand_ext, 580
 - device_sendPAE, 580
 - device_sendVivoCommandP2, 581
 - device_sendVivoCommandP2_ext, 581
 - device_sendVivoCommandP3, 582
 - device_sendVivoCommandP3_ext, 582
 - device_setOutputType, 582
 - device_updateFirmwareFromZip, 583
 - getCommandTimeout, 584
 - lcd_retrieveMessage, 584
 - msr_switchUSBInterfaceMode, 584
 - SDK_Version, 585
 - setCallback, 585
 - setCommandTimeout, 586
 - SharedController, 586
- IDTechSDK::IDT_SecureMag
 - config_getModelNumber, 588
 - config_getSerialNumber, 588
 - device_RemoteKeyInjection, 600
 - device_StartRKI, 603, 604
 - device_getAnyRKIStatus, 588
 - device_getConfigurationFromMemory, 588
 - device_getFirmwareVersion, 589
 - device_getRKIStatus, 599
 - device_getResponseCodeString, 589
 - device_readConfigurationToMemory, 599
 - device_sendConfiguration, 600
 - device_sendConfigurationFromZip, 601
 - device_sendDataCommand, 602
 - device_sendDataCommand_ext, 602
 - device_sendPAE, 603
 - device_updateFirmwareFromZip, 604
 - getCommandTimeout, 604
 - lcd_retrieveMessage, 605
 - msr_RetrieveWhiteList, 608
 - msr_cancelMSRSwipe, 605
 - msr_captureMode, 605
 - msr_disable, 606
 - msr_getClearPANID, 606
 - msr_getExpirationMask, 606
 - msr_getFunctionStatus, 606
 - msr_getSetting, 607
 - msr_getSettings, 607
 - msr_getSwipeForcedEncryptionOption, 608
 - msr_getSwipeMaskOption, 608
 - msr_setClearPANID, 609
 - msr_setExpirationMask, 609
 - msr_setSettings, 609
 - msr_setSwipeEncryption, 610
 - msr_setSwipeForcedEncryptionOption, 610
 - msr_setSwipeMaskOption, 610
 - msr_setWhiteList, 611
 - msr_startMSRSwipe, 611
 - SDK_Version, 611
 - setCallback, 611, 612
 - setCommandTimeout, 612
 - SharedController, 612
- IDTechSDK::IDT_SpectrumPro
 - config_getModelNumber, 615
 - config_getSerialNumber, 615
 - createFastEMVData, 615
 - device_RemoteKeyInjection, 631
 - device_StartRKI, 636
 - device_activateTransaction, 616
 - device_cardNotification, 616
 - device_controlUserInterface, 616
 - device_getAnyRKIStatus, 617
 - device_getConfigurationFromMemory, 618
 - device_getFirmwareVersion, 618
 - device_getNonce, 618
 - device_getRKIStatus, 628
 - device_getResponseCodeString, 618
 - device_getSpectrumProKSN, 629
 - device_getVersions, 629
 - device_pollCardReader, 629
 - device_readConfigurationToMemory, 630
 - device_rebootDevice, 631
 - device_sendConfiguration, 632
 - device_sendConfigurationFromZip, 633
 - device_sendDataCommand, 633
 - device_sendDataCommand_ext, 634
 - device_sendMacDataCommand, 634

- device_sendPAE, 635
- device_setSpectrumProBDK, 635
- device_setUID, 635
- device_startRKI, 636
- device_startTransaction, 637
- device_updateFirmware, 637
- emv_activateTransaction, 638
- emv_addTerminalData, 638
- emv_allowFallback, 639
- emv_authenticateTransaction, 639
- emv_autoAuthenticate, 639
- emv_autoAuthenticateTags, 640
- emv_callbackResponseLCD, 640
- emv_callbackResponseMSR, 640
- emv_callbackResponsePIN, 641
- emv_cancelTransaction, 641
- emv_clearTransactionLog, 641
- emv_completeTransaction, 642
- emv_getEMVConfigurationCheckValue, 642
- emv_getEMVKernelCheckValue, 642
- emv_getEMVKernelVersion, 643
- emv_getTerminalID, 643
- emv_getTerminalMajorConfiguration, 643
- emv_removeAllApplicationData, 644
- emv_removeAllCAPK, 644
- emv_removeAllCRL, 644
- emv_removeApplicationData, 645
- emv_removeCAPK, 645
- emv_removeCRL, 645
- emv_removeTerminalData, 646
- emv_retrieveAIDList, 646
- emv_retrieveApplicationData, 646
- emv_retrieveCAPKList, 647
- emv_retrieveCAPK, 646
- emv_retrieveCRLList, 647
- emv_retrieveTerminalData, 648
- emv_retrieveTransactionResult, 648
- emv_setApplicationData, 648
- emv_setCAPK, 648
- emv_setCRL, 649
- emv_setTerminalData, 649
- emv_setTerminalID, 650
- emv_setTerminalMajorConfiguration, 650
- emv_startTransaction, 650
- emv_trySetTerminalData, 651
- getCommandTimeout, 651
- icc_getICCReaderStatus, 652
- icc_getICCStatus, 652
- icc_powerOffICC, 653
- icc_powerOnICC, 653
- lcd_retrieveMessage, 654
- msr_cancelMSRSwipe, 654
- msr_clearMSRData, 654
- msr_getMSRData, 655
- msr_startMSRSwipe, 655
- pin_cancelPINEntry, 655
- pin_getPIN, 656
- pin_passThoughMode, 656
- SDK_Version, 656
- setCallback, 657
- setCardNotificationCallback, 657
- setCommandTimeout, 657
- SharedController, 659
- useSerialPort, 658
- useSerialPortLinux, 658
- IDTechSDK::IDT_UniPay
 - config_getModelNumber, 697
 - config_getSerialNumber, 697
 - config_setCmdTimeOutDuration, 697
 - device_RemoteKeyInjection, 701
 - device_getAnyRKIStatus, 697
 - device_getBatteryVoltage, 698
 - device_getBootloaderVersion, 698
 - device_getConfigurationFromMemory, 699
 - device_getFirmwareVersion, 699
 - device_getRKIStatus, 699
 - device_readConfigurationToMemory, 700
 - device_rebootDevice, 700
 - device_sendConfiguration, 701
 - device_sendConfigurationFromZip, 702
 - device_sendDataCommand, 703
 - device_sendPAE, 703
 - device_startRKI, 703
 - device_updateFirmwareFromZip, 705
 - getCommandTimeout, 705
 - icc_exchangeAPDU, 705
 - icc_getAPDU_KSN, 706
 - icc_getICCReaderStatus, 706
 - icc_getKeyFormatForICCDUKPT, 706
 - icc_getKeyTypeForICCDUKPT, 707
 - icc_powerOffICC, 707
 - icc_powerOnICC, 707
 - icc_setKeyFormatForICCDUKPT, 707
 - icc_setKeyTypeForICCDUKPT, 708
 - msr_cancelMSRSwipe, 708
 - msr_getClearPANID, 708
 - msr_getExpirationMask, 709
 - msr_getSetting, 709
 - msr_getSwipeEncryption, 709
 - msr_getSwipeForcedEncryptionOption, 710
 - msr_getSwipeMaskOption, 710
 - msr_setClearPANID, 710
 - msr_setExpirationMask, 711
 - msr_setSetting, 711
 - msr_setSwipeEncryption, 711
 - msr_setSwipeForcedEncryptionOption, 712
 - msr_setSwipeMaskOption, 712
 - msr_startMSRSwipe, 712
 - SDK_Version, 713
 - setCallback, 713
 - setCommandTimeout, 713
 - SharedController, 714
- IDTechSDK::IDT_UniPayI_V
 - config_getEncryptionControl, 716
 - config_getSerialNumber, 717
 - config_setEncryptionControl, 717

- device_RemoteKeyInjection, [732](#)
- device_StartRKI, [738](#)
- device_activateTransaction, [717](#)
- device_enablePassThrough, [719](#)
- device_getAnyRKIStatus, [719](#)
- device_getConfigurationFromMemory, [719](#)
- device_getDRS, [720](#)
- device_getFirmwareVersion, [720](#)
- device_getRKIStatus, [730](#)
- device_getResponseCodeString, [721](#)
- device_pingDevice, [731](#)
- device_readConfigurationToMemory, [731](#)
- device_retrieveTerminalData, [732](#)
- device_selfCheck, [732](#)
- device_sendConfiguration, [732](#)
- device_sendConfigurationFromZip, [733](#)
- device_sendDataCommand, [734](#)
- device_sendDataCommand_ext, [735](#)
- device_sendPAE, [735](#)
- device_sendVivoCommandP2, [735](#)
- device_sendVivoCommandP2_ext, [736](#)
- device_setBurstMode, [736](#)
- device_setPollMode, [737](#)
- device_setTerminalData, [737](#)
- device_startRKI, [737](#)
- device_startTransaction, [738](#)
- device_updateDeviceFirmware, [739](#)
- device_updateFirmwareFromZip, [740](#)
- emv_activateTransaction, [740](#)
- emv_addTerminalData, [741](#)
- emv_allowFallback, [741](#)
- emv_authenticateTransaction, [741](#)
- emv_autoAuthenticate, [742](#)
- emv_autoAuthenticateTags, [742](#)
- emv_callbackResponseLCD, [742](#)
- emv_callbackResponseMSR, [743](#)
- emv_callbackResponsePIN, [743](#)
- emv_cancelTransaction, [743](#)
- emv_completeTransaction, [744](#)
- emv_getEMVConfigurationCheckValue, [744](#)
- emv_getEMVKernelCheckValue, [744](#)
- emv_getEMVKernelVersion, [745](#)
- emv_getEMVKernelVersionExt, [745](#)
- emv_removeAllApplicationData, [745](#)
- emv_removeAllCAPK, [745](#)
- emv_removeAllCRL, [746](#)
- emv_removeApplicationData, [746](#)
- emv_removeCAPK, [746](#)
- emv_removeCRL, [747](#)
- emv_removeTerminalData, [747](#)
- emv_retrieveAIDList, [747](#)
- emv_retrieveApplicationData, [747](#)
- emv_retrieveCAPKList, [748](#)
- emv_retrieveCAPK, [748](#)
- emv_retrieveCRLList, [749](#)
- emv_retrieveTerminalData, [749](#)
- emv_retrieveTransactionResult, [749](#)
- emv_setApplicationData, [749](#)
- emv_setCAPK, [750](#)
- emv_setCRL, [750](#)
- emv_setTerminalData, [751](#)
- emv_setTerminalMajorConfiguration, [751](#)
- emv_startTransaction, [751](#)
- emv_trySetTerminalData, [752](#)
- getCommandTimeout, [752](#)
- icc_exchangeAPDU, [753](#)
- icc_getICCReaderStatus, [753](#)
- icc_powerOffICC, [753](#)
- icc_powerOnICC, [753](#)
- lcd_retrieveMessage, [754](#)
- msr_cancelMSRSwipe, [754](#)
- msr_startMSRSwipe, [754](#)
- msr_switchUSBInterfaceMode, [756](#)
- SDK_Version, [756](#)
- setCallback, [757](#)
- setCommandTimeout, [757](#)
- SharedController, [757](#)
- IDTechSDK::IDT_VP3300
 - config_getDate, [817](#)
 - config_getEncryptionControl, [818](#)
 - config_getSerialNumber, [818](#)
 - config_getTime, [819](#)
 - config_setDate, [819](#)
 - config_setEncryptionControl, [819](#)
 - createFastEMVData, [820](#)
 - ctls_activateTransaction, [820](#)
 - ctls_cancelTransaction, [822](#)
 - ctls_getAllConfigurationGroups, [822](#)
 - ctls_getConfigurationGroup, [822](#)
 - ctls_removeAllCAPK, [823](#)
 - ctls_removeApplicationData, [823](#)
 - ctls_removeCAPK, [823](#)
 - ctls_removeConfigurationGroup, [823](#)
 - ctls_retrieveAIDList, [824](#)
 - ctls_retrieveApplicationData, [824](#)
 - ctls_retrieveCAPKList, [825](#)
 - ctls_retrieveCAPK, [824](#)
 - ctls_retrieveTerminalData, [825](#)
 - ctls_setApplicationData, [826](#)
 - ctls_setCAPK, [826](#)
 - ctls_setConfigurationGroup, [827](#)
 - ctls_setDefaultConfiguration, [827](#)
 - ctls_setTerminalData, [827](#)
 - ctls_startTransaction, [827](#)
 - ctls_trySetTerminalData, [829](#)
 - device_RemoteKeyInjection, [850](#)
 - device_StartRKI, [856](#)
 - device_activateTransaction, [829](#)
 - device_buzzer, [830](#)
 - device_cancelTransaction, [831](#)
 - device_controlLED, [831](#)
 - device_controlUserInterface, [832](#)
 - device_enablePassThrough, [833](#)
 - device_getAnyRKIStatus, [834](#)
 - device_getCashTranRiskPara, [834](#)
 - device_getConfigurationFromMemory, [834](#)

- device_getDateTime, 835
- device_getDriReaderRiskPara, 835
- device_getFirmwareVersion, 835
- device_getHardwareInfor, 836
- device_getMerchantRecord, 836
- device_getModuleVer, 836
- device_getPollMode, 837
- device_getProductType, 837
- device_getRKIStatus, 847
- device_getResponseCodeString, 838
- device_getTransactionResults, 848
- device_getUIDofMCU, 848
- device_getUsbBootLoader, 848
- device_pingDevice, 849
- device_readConfigurationToMemory, 849
- device_rebootDevice, 849
- device_sendConfiguration, 850
- device_sendConfigurationFromZip, 851
- device_sendDataCommand, 852
- device_sendDataCommand_ext, 852
- device_sendPAE, 852
- device_sendVivoCommandP2, 853
- device_sendVivoCommandP2_ext, 853
- device_setBurstMode, 854
- device_setDateTime, 854
- device_setLED, 854
- device_setMerchantRecord, 854
- device_setPollMode, 855
- device_setQuickChipHIDMode, 855
- device_startRKI, 856
- device_startTransaction, 856
- device_updateDeviceFirmware, 857
- device_updateFirmwareFromZip, 859
- emv_activateTransaction, 859
- emv_addTerminalData, 860
- emv_allowFallback, 860
- emv_authenticateTransaction, 860
- emv_autoAuthenticate, 860
- emv_autoAuthenticateTags, 861
- emv_callbackResponseLCD, 861
- emv_callbackResponseMSR, 861
- emv_callbackResponsePIN, 862
- emv_cancelTransaction, 862
- emv_completeTransaction, 862
- emv_getEMVConfigurationCheckValue, 863
- emv_getEMVKernelCheckValue, 863
- emv_getEMVKernelVersion, 863
- emv_getEMVKernelVersionExt, 864
- emv_getTerminalMajorConfiguration, 864
- emv_removeAllApplicationData, 864
- emv_removeAllCAPK, 865
- emv_removeAllCRL, 865
- emv_removeApplicationData, 865
- emv_removeCAPK, 865
- emv_removeCRL, 866
- emv_removeTerminalData, 866
- emv_retrieveAIDList, 866
- emv_retrieveApplicationData, 867
- emv_retrieveCAPKList, 867
- emv_retrieveCAPK, 867
- emv_retrieveCRLList, 868
- emv_retrieveTerminalData, 868
- emv_retrieveTransactionResult, 868
- emv_setApplicationData, 869
- emv_setCAPK, 869
- emv_setCRL, 870
- emv_setTerminalData, 870
- emv_setTerminalMajorConfiguration, 870
- emv_startTransaction, 870
- emv_trySetTerminalData, 871
- getCommandTimeout, 871
- icc_exchangeAPDU, 872
- icc_getICCReaderStatus, 872
- icc_powerOffICC, 872
- icc_powerOnICC, 873
- lcd_retrieveMessage, 873
- msr_cancelMSRSwipe, 873
- msr_startMSRSwipe, 874
- SDK_Version, 874
- setCallback, 874
- setCommandTimeout, 874
- SharedController, 875
- useSerialPort, 875
- IDTechSDK::IDT_VP8800
 - config_getDataKeySlot, 879
 - config_getEthernetMACAddress, 880
 - config_getKeySlotInfo, 880
 - config_getNetworkConfiguration, 880
 - config_getPINKeySlot, 881
 - config_getSerialNumber, 881
 - config_getStatusKeySlots, 881
 - config_setDataKeySlot, 882
 - config_setEthernetMACAddress, 882
 - config_setNetworkConfiguration, 882
 - config_setPINKeySlot, 883
 - createFastEMVData, 883
 - ctls_activateTransaction, 883
 - ctls_cancelTransaction, 884
 - ctls_displayOnlineAuthResult, 885
 - ctls_displayOnlineAuthResult_ext, 885
 - ctls_getConfigurationGroup, 885
 - ctls_nfcCommand, 886
 - ctls_removeAllCAPK, 887
 - ctls_removeApplicationData, 888
 - ctls_removeCAPK, 888
 - ctls_removeConfigurationGroup, 888
 - ctls_resetConfigurationGroup, 888
 - ctls_retrieveAIDList, 889
 - ctls_retrieveApplicationData, 889
 - ctls_retrieveCAPKList, 890
 - ctls_retrieveCAPK, 889
 - ctls_retrieveTerminalData, 890
 - ctls_setApplicationData, 891
 - ctls_setCAPK, 891
 - ctls_setConfigurationGroup, 892
 - ctls_setDefaultConfiguration, 892

- ctls_setTerminalData, 893
- ctls_startTransaction, 893
- ctls_trySetTerminalData, 894
- device_RemoteKeyInjection, 913
- device_StartRKI, 920
- device_activateTransaction, 895
- device_buzzer, 896
- device_calibrateParameters, 896
- device_cancelTransaction, 896
- device_controlIndicator, 896
- device_controlLED, 897
- device_controlUserInterface, 897
- device_createDirectory, 898
- device_deleteDirectory, 899
- device_deleteFile, 899
- device_enablePassThrough, 899
- device_getConfigurationFromMemory, 900
- device_getConfigurationGroup, 900
- device_getDriveFreeSpace, 900
- device_getFirmwareVersion, 901
- device_getMerchantRecord, 901
- device_getResponseCodeString, 901
- device_getTransactionResults, 911
- device_listDirectory, 911
- device_pingDevice, 912
- device_readConfigurationToMemory, 912
- device_removeConfigurationGroup, 913
- device_resetConfigurationGroup, 913
- device_retrieveAIDList, 914
- device_sendConfiguration, 914
- device_sendConfigurationFromZip, 915
- device_sendDataCommand, 916
- device_sendDataCommand_ext, 916
- device_sendPAE, 916
- device_sendVivoCommandP2, 917
- device_sendVivoCommandP2_ext, 917
- device_sendVivoCommandP3, 917
- device_sendVivoCommandP3_ext, 918
- device_setBurstMode, 918
- device_setBuzzerLED, 919
- device_setMerchantRecord, 919
- device_setPollMode, 920
- device_startRKI, 920
- device_startTransaction, 921
- device_transferFile, 922
- device_updateDeviceFirmware, 922
- device_updateFirmwareFromZip, 924
- emv_activateTransaction, 924
- emv_addTerminalData, 925
- emv_allowFallback, 925
- emv_autoAuthenticate, 925
- emv_autoAuthenticateTags, 925
- emv_callbackResponseLCD, 925
- emv_callbackResponseMSR, 926
- emv_callbackResponsePIN, 926
- emv_cancelTransaction, 927
- emv_clearTransactionLog, 927
- emv_completeTransaction, 927
- emv_continueTransactionForCV, 928
- emv_getCRLStatus, 929
- emv_getConfigurationGroup, 928
- emv_getEMVConfigurationCheckValue, 929
- emv_getEMVKernelCheckValue, 929
- emv_getEMVKernelVersion, 929
- emv_getEMVKernelVersionExt, 930
- emv_getExemptionLogStatus, 930
- emv_getTransactionLogRecord, 930
- emv_getTransactionLogStatus, 932
- emv_removeAllCAPK, 932
- emv_removeAllCRL, 932
- emv_removeAllExceptions, 932
- emv_removeApplicationData, 933
- emv_removeCAPK, 933
- emv_removeCRL, 934
- emv_removeConfigurationGroup, 933
- emv_removeException, 934
- emv_removeTerminalData, 934
- emv_removeTransactionAmountLog, 935
- emv_resetConfigurationGroup, 935
- emv_retrieveAIDList, 935
- emv_retrieveApplicationData, 935
- emv_retrieveCAPKList, 936
- emv_retrieveCAPK, 936
- emv_retrieveCRLList, 937
- emv_retrieveExceptionList, 937
- emv_retrieveTerminalData, 937
- emv_retrieveTransactionResult, 938
- emv_setApplicationData, 938
- emv_setCAPK, 939
- emv_setCRL, 939
- emv_setException, 940
- emv_setTerminalDataVP8800, 940
- emv_startTransaction, 940
- emv_trySetTerminalData, 941
- felica_SendCommand, 943
- felica_authentication, 941
- felica_read, 942
- felica_readWithMac, 942
- felica_requestService, 943
- felica_write, 943
- felica_writeWithMac, 944
- getCommandTimeout, 944
- icc_exchangeAPDU, 944
- icc_powerOffICC, 944
- icc_powerOnICC, 945
- lcd_captureSignature, 945
- lcd_clearDisplay, 945
- lcd_clearInputEvents, 946
- lcd_customDisplayMode, 946
- lcd_displayButton, 946
- lcd_displayText, 948
- lcd_getInputEvent, 949
- lcd_resetInitialState, 950
- lcd_retrieveMessage, 950
- lcd_setBackgroundImage, 950
- lcd_setDisplayImage, 951

- lcd_setForeBackColor, [951](#)
- lcd_startSlideShow, [952](#)
- msr_cancelMSRSwipe, [952](#)
- msr_flushTrackData, [953](#)
- msr_startMSRSwipe, [953](#)
- pin_getEncryptedOnlinePIN, [953](#)
- pin_getPAN, [953](#)
- pin_promptCreditDebit, [954](#)
- SDK_Version, [954](#)
- setCallback, [954](#)
- setCommandTimeout, [955](#)
- SharedController, [956](#)
- useSerialPort, [955](#)
- useSerialPortLinux, [955](#), [956](#)
- IDTechSDK::IDT_VendIII
 - config_getSerialNumber, [787](#)
 - config_setBaudRate, [788](#)
 - ctls_activateTransaction, [788](#)
 - ctls_cancelTransaction, [789](#)
 - ctls_getAllConfigurationGroups, [790](#)
 - ctls_getConfigurationGroup, [790](#)
 - ctls_removeAllCAPK, [790](#)
 - ctls_removeApplicationData, [791](#)
 - ctls_removeCAPK, [791](#)
 - ctls_removeConfigurationGroup, [791](#)
 - ctls_retrieveAIDList, [791](#)
 - ctls_retrieveApplicationData, [792](#)
 - ctls_retrieveCAPKList, [793](#)
 - ctls_retrieveCAPK, [792](#)
 - ctls_retrieveTerminalData, [793](#)
 - ctls_setApplicationData, [794](#)
 - ctls_setCAPK, [794](#)
 - ctls_setConfigurationGroup, [795](#)
 - ctls_setDefaultConfiguration, [795](#)
 - ctls_setTerminalData, [795](#)
 - ctls_startTransaction, [795](#)
 - ctls_trySetTerminalData, [797](#)
 - device_RemoteKeyInjection, [801](#)
 - device_activateTransaction, [797](#)
 - device_controlLED, [798](#)
 - device_controlUserInterface, [798](#)
 - device_getAnyRKIStatus, [799](#)
 - device_getConfigurationFromMemory, [799](#)
 - device_getFirmwareVersion, [799](#)
 - device_getMerchantRecord, [800](#)
 - device_getRKIStatus, [800](#)
 - device_getTransactionResults, [800](#)
 - device_pingDevice, [801](#)
 - device_readConfigurationToMemory, [801](#)
 - device_retrieveAIDList, [802](#)
 - device_sendConfiguration, [802](#)
 - device_sendConfigurationFromZip, [803](#)
 - device_sendDataCommand, [804](#)
 - device_sendDataCommand_ext, [804](#)
 - device_sendPAE, [805](#)
 - device_sendVivoCommandP2, [805](#)
 - device_sendVivoCommandP2_ext, [805](#)
 - device_setBurstMode, [806](#)
 - device_setMerchantRecord, [806](#)
 - device_setPollMode, [806](#)
 - device_startRKI, [807](#)
 - device_startTransaction, [807](#)
 - device_updateDeviceFirmware, [807](#)
 - device_updateFirmwareFromZip, [809](#)
 - emv_activateTransaction, [809](#)
 - emv_completeTransaction, [810](#)
 - emv_startTransaction, [810](#)
 - lcd_retrieveMessage, [811](#)
 - msr_cancelMSRSwipe, [811](#)
 - msr_startMSRSwipe, [811](#)
 - SDK_Version, [811](#)
 - setCallback, [812](#)
 - SharedController, [813](#)
 - useSerialPort, [812](#)
 - useSerialPortLinux, [813](#)
- IDTechSDK::IDT_Vendi
 - config_getSerialNumber, [759](#)
 - config_setBaudRate, [759](#)
 - ctls_activateTransaction, [760](#)
 - ctls_cancelTransaction, [761](#)
 - ctls_getAllConfigurationGroups, [761](#)
 - ctls_getConfigurationGroup, [762](#)
 - ctls_removeAllCAPK, [762](#)
 - ctls_removeApplicationData, [762](#)
 - ctls_removeCAPK, [762](#)
 - ctls_removeConfigurationGroup, [763](#)
 - ctls_retrieveAIDList, [763](#)
 - ctls_retrieveApplicationData, [763](#)
 - ctls_retrieveCAPKList, [764](#)
 - ctls_retrieveCAPK, [764](#)
 - ctls_retrieveTerminalData, [764](#)
 - ctls_setApplicationData, [765](#)
 - ctls_setCAPK, [765](#)
 - ctls_setConfigurationGroup, [766](#)
 - ctls_setDefaultConfiguration, [766](#)
 - ctls_setTerminalData, [767](#)
 - ctls_startTransaction, [767](#)
 - ctls_trySetTerminalData, [768](#)
 - device_RemoteKeyInjection, [774](#)
 - device_activateTransaction, [769](#)
 - device_controlLED, [770](#)
 - device_controlUserInterface, [770](#)
 - device_getAnyRKIStatus, [771](#)
 - device_getConfigurationFromMemory, [771](#)
 - device_getFirmwareVersion, [772](#)
 - device_getMerchantRecord, [772](#)
 - device_getRKIStatus, [772](#)
 - device_getTransactionResults, [773](#)
 - device_pingDevice, [773](#)
 - device_readConfigurationToMemory, [773](#)
 - device_retrieveAIDList, [774](#)
 - device_sendConfiguration, [774](#)
 - device_sendConfigurationFromZip, [775](#)
 - device_sendDataCommand, [776](#)
 - device_sendDataCommand_ext, [777](#)
 - device_sendPAE, [777](#)

- device_sendVivoCommandP2, 777
- device_sendVivoCommandP2_ext, 778
- device_setBurstMode, 778
- device_setMerchantRecord, 779
- device_setPollMode, 779
- device_startRKI, 779
- device_startTransaction, 779
- device_updateDeviceFirmware, 781
- device_updateFirmwareFromZip, 782
- getCommandTimeout, 782
- lcd_retrieveMessage, 783
- msr_cancelMSRSwipe, 783
- msr_startMSRSwipe, 783
- SDK_Version, 783
- setCallback, 784
- setCommandTimeout, 784
- SharedController, 785
- useSerialPort, 784
- useSerialPortLinux, 785
- IDTechSDK, 80
- icc_disable
 - IDTechSDK::IDT_Augusta, 125
 - IDTechSDK::IDT_BTPay, 211
- icc_enable
 - IDTechSDK::IDT_Augusta, 125
 - IDTechSDK::IDT_BTPay, 211
- icc_exchangeAPDU
 - IDTechSDK::IDT_Augusta, 126
 - IDTechSDK::IDT_BTPay, 211
 - IDTechSDK::IDT_MiniSmartII, 375
 - IDTechSDK::IDT_NEO2, 496
 - IDTechSDK::IDT_UniPay, 705
 - IDTechSDK::IDT_UniPayI_V, 753
 - IDTechSDK::IDT_VP3300, 872
 - IDTechSDK::IDT_VP8800, 944
- icc_exchangeEncryptedAPDU
 - IDTechSDK::IDT_Augusta, 126
 - IDTechSDK::IDT_BTPay, 212
- icc_getAPDU_KSN
 - IDTechSDK::IDT_Augusta, 127
 - IDTechSDK::IDT_BTPay, 212
 - IDTechSDK::IDT_MiniSmartII, 376
 - IDTechSDK::IDT_UniPay, 706
- icc_getClearPANID
 - IDTechSDK::IDT_MiniSmartII, 376
- icc_getFunctionStatus
 - IDTechSDK::IDT_Augusta, 127
 - IDTechSDK::IDT_BTPay, 213
- icc_getICCReaderStatus
 - IDTechSDK::IDT_Augusta, 128
 - IDTechSDK::IDT_BTPay, 213
 - IDTechSDK::IDT_MiniSmartII, 376
 - IDTechSDK::IDT_NEO2, 496
 - IDTechSDK::IDT_SpectrumPro, 652
 - IDTechSDK::IDT_UniPay, 706
 - IDTechSDK::IDT_UniPayI_V, 753
 - IDTechSDK::IDT_VP3300, 872
- icc_getICCStatus
 - IDTechSDK::IDT_SpectrumPro, 652
- icc_getKeyFormatForICCDUKPT
 - IDTechSDK::IDT_Augusta, 128
 - IDTechSDK::IDT_BTPay, 213
 - IDTechSDK::IDT_MiniSmartII, 376
 - IDTechSDK::IDT_UniPay, 706
- icc_getKeyTypeForICCDUKPT
 - IDTechSDK::IDT_Augusta, 128
 - IDTechSDK::IDT_BTPay, 214
 - IDTechSDK::IDT_MiniSmartII, 377
 - IDTechSDK::IDT_UniPay, 707
- icc_getReadingCharacteristics
 - IDTechSDK::IDT_MiniSmartII, 377
- icc_getSetting
 - IDTechSDK::IDT_Augusta, 129
- icc_getSettings
 - IDTechSDK::IDT_Augusta, 129
- icc_powerOffICC
 - IDTechSDK::IDT_Augusta, 129
 - IDTechSDK::IDT_BTPay, 214
 - IDTechSDK::IDT_MiniSmartII, 378
 - IDTechSDK::IDT_NEO2, 496
 - IDTechSDK::IDT_SpectrumPro, 653
 - IDTechSDK::IDT_UniPay, 707
 - IDTechSDK::IDT_UniPayI_V, 753
 - IDTechSDK::IDT_VP3300, 872
 - IDTechSDK::IDT_VP8800, 944
- icc_powerOnICC
 - IDTechSDK::IDT_Augusta, 129
 - IDTechSDK::IDT_BTPay, 214
 - IDTechSDK::IDT_MiniSmartII, 378
 - IDTechSDK::IDT_NEO2, 497
 - IDTechSDK::IDT_SpectrumPro, 653
 - IDTechSDK::IDT_UniPay, 707
 - IDTechSDK::IDT_UniPayI_V, 753
 - IDTechSDK::IDT_VP3300, 873
 - IDTechSDK::IDT_VP8800, 945
- icc_setClearPANID
 - IDTechSDK::IDT_MiniSmartII, 378
- icc_setKeyFormatForICCDUKPT
 - IDTechSDK::IDT_Augusta, 130
 - IDTechSDK::IDT_BTPay, 215
 - IDTechSDK::IDT_MiniSmartII, 379
 - IDTechSDK::IDT_UniPay, 707
- icc_setKeyTypeForICCDUKPT
 - IDTechSDK::IDT_Augusta, 130
 - IDTechSDK::IDT_BTPay, 215
 - IDTechSDK::IDT_MiniSmartII, 379
 - IDTechSDK::IDT_UniPay, 708
- icc_setReadingCharacteristics
 - IDTechSDK::IDT_MiniSmartII, 379
- icc_setSetting
 - IDTechSDK::IDT_Augusta, 130, 131
- ip_autoConnectToSocket
 - IDTechSDK::IDT_NEO2, 497
- ip_connectToSocket
 - IDTechSDK::IDT_NEO2, 497
- ip_firstConnectToSocket

- IDTechSDK::IDT_NEO2, [498](#)
- ip_getSocketList
 - IDTechSDK::IDT_NEO2, [498](#)
- ip_isConnected
 - IDTechSDK::IDT_NEO2, [499](#)
- ip_monitorSocketConnectionStatus
 - IDTechSDK::IDT_NEO2, [499](#)
- ip_switchToSocket
 - IDTechSDK::IDT_NEO2, [500](#)
- lcd_addButton
 - IDTechSDK::IDT_NEO2, [500](#)
- lcd_addEthernet
 - IDTechSDK::IDT_NEO2, [501](#)
- lcd_addImage
 - IDTechSDK::IDT_NEO2, [501](#)
- lcd_addLED
 - IDTechSDK::IDT_NEO2, [502](#)
- lcd_addText
 - IDTechSDK::IDT_NEO2, [503](#)
- lcd_captureSignature
 - IDTechSDK::IDT_VP8800, [945](#)
- lcd_clearAllLines
 - IDTechSDK::IDT_L100, [303](#)
 - IDTechSDK::IDT_L80, [331](#)
 - IDTechSDK::IDT_NEO2, [505](#)
- lcd_clearDisplay
 - IDTechSDK::IDT_L100, [303](#)
 - IDTechSDK::IDT_L80, [331](#)
 - IDTechSDK::IDT_NEO2, [506](#)
 - IDTechSDK::IDT_VP8800, [945](#)
- lcd_clearInputEvents
 - IDTechSDK::IDT_VP8800, [946](#)
- lcd_clearScreenInfo
 - IDTechSDK::IDT_NEO2, [506](#)
- lcd_cloneScreen
 - IDTechSDK::IDT_NEO2, [506](#)
- lcd_createScreen
 - IDTechSDK::IDT_NEO2, [506](#)
- lcd_customDisplayMode
 - IDTechSDK::IDT_VP8800, [946](#)
- lcd_destroyScreen
 - IDTechSDK::IDT_NEO2, [507](#)
- lcd_displayButton
 - IDTechSDK::IDT_VP8800, [946](#)
- lcd_displayMessage
 - IDTechSDK::IDT_L100, [303](#)
 - IDTechSDK::IDT_L80, [332](#)
 - IDTechSDK::IDT_NEO2, [507](#)
- lcd_displayPrompt
 - IDTechSDK::IDT_L100, [304](#)
 - IDTechSDK::IDT_L80, [332](#)
- lcd_displayText
 - IDTechSDK::IDT_VP8800, [948](#)
- lcd_enableBacklight
 - IDTechSDK::IDT_L100, [304](#)
 - IDTechSDK::IDT_L80, [332](#)
- lcd_getActiveScreen
 - IDTechSDK::IDT_NEO2, [507](#)
- lcd_getAllObjects
 - IDTechSDK::IDT_NEO2, [508](#)
- lcd_getAllScreens
 - IDTechSDK::IDT_NEO2, [508](#)
- lcd_getAudioVolume
 - IDTechSDK::IDT_NEO2, [508](#)
- lcd_getBacklightStatus
 - IDTechSDK::IDT_L100, [304](#)
 - IDTechSDK::IDT_L80, [332](#)
- lcd_getButtonEvent
 - IDTechSDK::IDT_NEO2, [509](#)
- lcd_getInputEvent
 - IDTechSDK::IDT_VP8800, [949](#)
- lcd_linkUIWithTransactionMessageId
 - IDTechSDK::IDT_NEO2, [509](#)
- lcd_loadScreenInfo
 - IDTechSDK::IDT_NEO2, [510](#)
- lcd_playAudio
 - IDTechSDK::IDT_NEO2, [510](#)
- lcd_queryObjectbyID
 - IDTechSDK::IDT_NEO2, [510](#)
- lcd_queryObjectbyName
 - IDTechSDK::IDT_NEO2, [511](#)
- lcd_queryScreenbyID
 - IDTechSDK::IDT_NEO2, [511](#)
- lcd_queryScreenbyName
 - IDTechSDK::IDT_NEO2, [511](#)
- lcd_removeItem
 - IDTechSDK::IDT_NEO2, [512](#)
- lcd_resetInitialState
 - IDTechSDK::IDT_VP8800, [950](#)
- lcd_retrieveMessage
 - IDTechSDK::IDT_Augusta, [131](#)
 - IDTechSDK::IDT_KioskIII, [285](#)
 - IDTechSDK::IDT_L100, [305](#)
 - IDTechSDK::IDT_L80, [333](#)
 - IDTechSDK::IDT_MiniSmartII, [380](#)
 - IDTechSDK::IDT_NEO2, [512](#)
 - IDTechSDK::IDT_PIP, [562](#)
 - IDTechSDK::IDT_SREDKey2, [686](#)
 - IDTechSDK::IDT_SecureKey, [584](#)
 - IDTechSDK::IDT_SecureMag, [605](#)
 - IDTechSDK::IDT_SpectrumPro, [654](#)
 - IDTechSDK::IDT_UniPayI_V, [754](#)
 - IDTechSDK::IDT_VP3300, [873](#)
 - IDTechSDK::IDT_VP8800, [950](#)
 - IDTechSDK::IDT_VendIII, [811](#)
 - IDTechSDK::IDT_Vendi, [783](#)
- lcd_savePrompt
 - IDTechSDK::IDT_L100, [305](#)
 - IDTechSDK::IDT_L80, [333](#)
- lcd_setAudioVolume
 - IDTechSDK::IDT_NEO2, [512](#)
- lcd_setBackgroundImage
 - IDTechSDK::IDT_VP8800, [950](#)
- lcd_setBacklight
 - IDTechSDK::IDT_NEO2, [514](#)
- lcd_setButtonCallback

- IDTechSDK::IDT_NEO2, [514](#)
- lcd_setDisplayImage
 - IDTechSDK::IDT_VP8800, [951](#)
- lcd_setForeBackColor
 - IDTechSDK::IDT_VP8800, [951](#)
- lcd_setPinCancelPromptCallback
 - IDTechSDK::IDT_NEO2, [514](#)
- lcd_setPinFailureCallback
 - IDTechSDK::IDT_NEO2, [515](#)
- lcd_setPinInputCallback
 - IDTechSDK::IDT_NEO2, [515](#)
- lcd_setPinSwipeCallback
 - IDTechSDK::IDT_NEO2, [515](#)
- lcd_setPinTimeoutCallback
 - IDTechSDK::IDT_NEO2, [515](#)
- lcd_showScreen
 - IDTechSDK::IDT_NEO2, [516](#)
- lcd_startCameraCapture
 - IDTechSDK::IDT_NEO2, [516](#)
- lcd_startScanQR_ext
 - IDTechSDK::IDT_NEO2, [517](#)
- lcd_startScanQR
 - IDTechSDK::IDT_NEO2, [516](#)
- lcd_startScreenSaver
 - IDTechSDK::IDT_NEO2, [517](#)
- lcd_startSlideShow
 - IDTechSDK::IDT_VP8800, [952](#)
- lcd_stopAudio
 - IDTechSDK::IDT_NEO2, [517](#)
- lcd_stopCameraCapture
 - IDTechSDK::IDT_NEO2, [518](#)
- lcd_stopScanQR
 - IDTechSDK::IDT_NEO2, [518](#)
- lcd_storeScreenInfo
 - IDTechSDK::IDT_NEO2, [518](#)
- lcd_updateColor
 - IDTechSDK::IDT_NEO2, [519](#)
- lcd_updateLabel
 - IDTechSDK::IDT_NEO2, [519](#)
- lcd_updatePosition
 - IDTechSDK::IDT_NEO2, [520](#)
- monitorNetworkForDevices
 - IDTechSDK::IDT_NEO2, [520](#)
- msr_RetrieveWhiteList
 - IDTechSDK::IDT_Augusta, [135](#)
 - IDTechSDK::IDT_BTMag, [168](#)
 - IDTechSDK::IDT_BTPay, [219](#)
 - IDTechSDK::IDT_SecureMag, [608](#)
- msr_cancelMSRSwipe
 - IDTechSDK::IDT_Augusta, [131](#)
 - IDTechSDK::IDT_BTMag, [163](#)
 - IDTechSDK::IDT_BTPay, [215](#)
 - IDTechSDK::IDT_NEO2, [521](#)
 - IDTechSDK::IDT_SecureMag, [605](#)
 - IDTechSDK::IDT_SpectrumPro, [654](#)
 - IDTechSDK::IDT_UniPay, [708](#)
 - IDTechSDK::IDT_UniPayl_V, [754](#)
 - IDTechSDK::IDT_VP3300, [873](#)
 - IDTechSDK::IDT_VP8800, [952](#)
 - IDTechSDK::IDT_VendIII, [811](#)
 - IDTechSDK::IDT_Vendi, [783](#)
- msr_cancelTransaction
 - IDTechSDK::IDT_SREDKey2, [687](#)
- msr_captureMode
 - IDTechSDK::IDT_Augusta, [132](#)
 - IDTechSDK::IDT_BTMag, [163](#)
 - IDTechSDK::IDT_BTPay, [216](#)
 - IDTechSDK::IDT_SecureMag, [605](#)
- msr_clearMSRData
 - IDTechSDK::IDT_SpectrumPro, [654](#)
- msr_disable
 - IDTechSDK::IDT_Augusta, [132](#)
 - IDTechSDK::IDT_BTMag, [165](#)
 - IDTechSDK::IDT_BTPay, [216](#)
 - IDTechSDK::IDT_SREDKey2, [687](#)
 - IDTechSDK::IDT_SecureMag, [606](#)
- msr_enable
 - IDTechSDK::IDT_SREDKey2, [687](#)
- msr_flushTrackData
 - IDTechSDK::IDT_VP8800, [953](#)
- msr_getClearPANID
 - IDTechSDK::IDT_Augusta, [132](#)
 - IDTechSDK::IDT_BTMag, [165](#)
 - IDTechSDK::IDT_BTPay, [216](#)
 - IDTechSDK::IDT_SREDKey2, [688](#)
 - IDTechSDK::IDT_SecureMag, [606](#)
 - IDTechSDK::IDT_UniPay, [708](#)
- msr_getConfiguration
 - IDTechSDK::IDT_NEO2, [521](#)
 - IDTechSDK::IDT_SREDKey2, [688](#)
- msr_getExpirationMask
 - IDTechSDK::IDT_Augusta, [133](#)
 - IDTechSDK::IDT_BTMag, [165](#)
 - IDTechSDK::IDT_BTPay, [217](#)
 - IDTechSDK::IDT_SREDKey2, [688](#)
 - IDTechSDK::IDT_SecureMag, [606](#)
 - IDTechSDK::IDT_UniPay, [709](#)
- msr_getFunctionStatus
 - IDTechSDK::IDT_Augusta, [133](#)
 - IDTechSDK::IDT_BTMag, [166](#)
 - IDTechSDK::IDT_BTPay, [217](#)
 - IDTechSDK::IDT_SREDKey2, [689](#)
 - IDTechSDK::IDT_SecureMag, [606](#)
- msr_getMSRData
 - IDTechSDK::IDT_SpectrumPro, [655](#)
- msr_getMSRTrack
 - IDTechSDK::IDT_NEO2, [521](#)
- msr_getSetting
 - IDTechSDK::IDT_Augusta, [133](#)
 - IDTechSDK::IDT_BTMag, [166](#)
 - IDTechSDK::IDT_BTPay, [218](#)
 - IDTechSDK::IDT_SecureMag, [607](#)
 - IDTechSDK::IDT_UniPay, [709](#)
- msr_getSettings
 - IDTechSDK::IDT_Augusta, [134](#)
 - IDTechSDK::IDT_BTMag, [166](#)

- IDTechSDK::IDT_BTPay, [218](#)
- IDTechSDK::IDT_SecureMag, [607](#)
- msr_getSwipeEncryption
 - IDTechSDK::IDT_Augusta, [134](#)
 - IDTechSDK::IDT_BTMag, [167](#)
 - IDTechSDK::IDT_BTPay, [218](#)
 - IDTechSDK::IDT_SREDKey2, [689](#)
 - IDTechSDK::IDT_UniPay, [709](#)
- msr_getSwipeForcedEncryptionOption
 - IDTechSDK::IDT_Augusta, [134](#)
 - IDTechSDK::IDT_BTMag, [167](#)
 - IDTechSDK::IDT_BTPay, [218](#)
 - IDTechSDK::IDT_SREDKey2, [689](#)
 - IDTechSDK::IDT_SecureMag, [608](#)
 - IDTechSDK::IDT_UniPay, [710](#)
- msr_getSwipeMaskOption
 - IDTechSDK::IDT_Augusta, [135](#)
 - IDTechSDK::IDT_BTMag, [167](#)
 - IDTechSDK::IDT_BTPay, [219](#)
 - IDTechSDK::IDT_SREDKey2, [690](#)
 - IDTechSDK::IDT_SecureMag, [608](#)
 - IDTechSDK::IDT_UniPay, [710](#)
- msr_getTerminalData
 - IDTechSDK::IDT_SREDKey2, [690](#)
- msr_resetTerminalData
 - IDTechSDK::IDT_SREDKey2, [690](#)
- msr_setClearPANID
 - IDTechSDK::IDT_Augusta, [135](#)
 - IDTechSDK::IDT_BTMag, [168](#)
 - IDTechSDK::IDT_BTPay, [220](#)
 - IDTechSDK::IDT_SREDKey2, [691](#)
 - IDTechSDK::IDT_SecureMag, [609](#)
 - IDTechSDK::IDT_UniPay, [710](#)
- msr_setConfiguration
 - IDTechSDK::IDT_NEO2, [522](#)
 - IDTechSDK::IDT_SREDKey2, [691](#)
- msr_setExpirationMask
 - IDTechSDK::IDT_Augusta, [136](#)
 - IDTechSDK::IDT_BTMag, [168](#)
 - IDTechSDK::IDT_BTPay, [220](#)
 - IDTechSDK::IDT_SREDKey2, [691](#)
 - IDTechSDK::IDT_SecureMag, [609](#)
 - IDTechSDK::IDT_UniPay, [711](#)
- msr_setMSRTrack
 - IDTechSDK::IDT_NEO2, [522](#)
- msr_setSetting
 - IDTechSDK::IDT_Augusta, [136](#)
 - IDTechSDK::IDT_BTMag, [169](#)
 - IDTechSDK::IDT_BTPay, [220](#)
 - IDTechSDK::IDT_UniPay, [711](#)
- msr_setSettings
 - IDTechSDK::IDT_Augusta, [136](#)
 - IDTechSDK::IDT_BTMag, [169](#)
 - IDTechSDK::IDT_BTPay, [220](#)
 - IDTechSDK::IDT_SecureMag, [609](#)
- msr_setSwipeEncryption
 - IDTechSDK::IDT_Augusta, [137](#)
 - IDTechSDK::IDT_BTMag, [169](#)
- IDTechSDK::IDT_BTPay, [221](#)
- IDTechSDK::IDT_SecureMag, [610](#)
- IDTechSDK::IDT_UniPay, [711](#)
- msr_setSwipeForcedEncryptionOption
 - IDTechSDK::IDT_Augusta, [137](#)
 - IDTechSDK::IDT_BTMag, [170](#)
 - IDTechSDK::IDT_BTPay, [221](#)
 - IDTechSDK::IDT_SREDKey2, [692](#)
 - IDTechSDK::IDT_SecureMag, [610](#)
 - IDTechSDK::IDT_UniPay, [712](#)
- msr_setSwipeMaskOption
 - IDTechSDK::IDT_Augusta, [137](#)
 - IDTechSDK::IDT_BTMag, [170](#)
 - IDTechSDK::IDT_BTPay, [221](#)
 - IDTechSDK::IDT_SREDKey2, [692](#)
 - IDTechSDK::IDT_SecureMag, [610](#)
 - IDTechSDK::IDT_UniPay, [712](#)
- msr_setTerminalData
 - IDTechSDK::IDT_SREDKey2, [692](#)
- msr_setWhiteList
 - IDTechSDK::IDT_Augusta, [138](#)
 - IDTechSDK::IDT_BTMag, [170](#)
 - IDTechSDK::IDT_BTPay, [222](#)
 - IDTechSDK::IDT_SecureMag, [611](#)
- msr_setWhiteListFromBDK
 - IDTechSDK::IDT_Augusta, [138](#)
- msr_startKeyedTransaction
 - IDTechSDK::IDT_SREDKey2, [693](#)
- msr_startMSRSwipe
 - IDTechSDK::IDT_Augusta, [138](#)
 - IDTechSDK::IDT_BTMag, [171](#)
 - IDTechSDK::IDT_BTPay, [222](#)
 - IDTechSDK::IDT_NEO2, [523](#)
 - IDTechSDK::IDT_SecureMag, [611](#)
 - IDTechSDK::IDT_SpectrumPro, [655](#)
 - IDTechSDK::IDT_UniPay, [712](#)
 - IDTechSDK::IDT_UniPayI_V, [754](#)
 - IDTechSDK::IDT_VP3300, [874](#)
 - IDTechSDK::IDT_VP8800, [953](#)
 - IDTechSDK::IDT_VendIII, [811](#)
 - IDTechSDK::IDT_Vendi, [783](#)
- msr_startMSRSwipe_ext
 - IDTechSDK::IDT_NEO2, [523](#)
- msr_startSwipeTransaction
 - IDTechSDK::IDT_SREDKey2, [693](#)
- msr_switchUSBInterfaceMode
 - IDTechSDK::IDT_Augusta, [139](#)
 - IDTechSDK::IDT_BTMag, [171](#)
 - IDTechSDK::IDT_BTPay, [222, 223](#)
 - IDTechSDK::IDT_SREDKey2, [693, 694](#)
 - IDTechSDK::IDT_SecureKey, [584](#)
 - IDTechSDK::IDT_UniPayI_V, [756](#)
- pin_cancelPINEntry
 - IDTechSDK::IDT_K100, [244](#)
 - IDTechSDK::IDT_L100, [305](#)
 - IDTechSDK::IDT_L80, [333](#)
 - IDTechSDK::IDT_NEO2, [524](#)
 - IDTechSDK::IDT_SpectrumPro, [655](#)

- pin_capturePin
 - IDTechSDK::IDT_NEO2, [524](#)
- pin_capturePin_ext
 - IDTechSDK::IDT_NEO2, [525](#)
- pin_capturePinExt
 - IDTechSDK::IDT_NEO2, [526](#)
- pin_enableKeypad
 - IDTechSDK::IDT_K100, [244](#)
- pin_getEncryptedOnlinePIN
 - IDTechSDK::IDT_VP8800, [953](#)
- pin_getEncryptedPIN
 - IDTechSDK::IDT_K100, [244](#)
 - IDTechSDK::IDT_L100, [306](#)
 - IDTechSDK::IDT_L80, [334](#)
- pin_getFunctionKey
 - IDTechSDK::IDT_K100, [244](#)
 - IDTechSDK::IDT_L100, [306](#)
 - IDTechSDK::IDT_L80, [334](#)
 - IDTechSDK::IDT_NEO2, [527](#)
- pin_getFunctionKey_ext
 - IDTechSDK::IDT_NEO2, [528](#)
- pin_getManualPanEntry
 - IDTechSDK::IDT_L100, [308](#)
 - IDTechSDK::IDT_L80, [335](#)
- pin_getPAN
 - IDTechSDK::IDT_VP8800, [953](#)
- pin_getPIN
 - IDTechSDK::IDT_SpectrumPro, [656](#)
- pin_getPanEntry
 - IDTechSDK::IDT_NEO2, [528](#)
- pin_getPanEntry_ext
 - IDTechSDK::IDT_NEO2, [529](#)
- pin_passThroughMode
 - IDTechSDK::IDT_SpectrumPro, [656](#)
- pin_promptCreditDebit
 - IDTechSDK::IDT_VP8800, [954](#)
- pin_promptForAmount
 - IDTechSDK::IDT_NEO2, [530](#)
- pin_promptForAmount_ext
 - IDTechSDK::IDT_NEO2, [530](#)
- pin_promptForAmountInput
 - IDTechSDK::IDT_L100, [308](#)
 - IDTechSDK::IDT_L80, [335](#)
- pin_promptForInput
 - IDTechSDK::IDT_NEO2, [531](#)
- pin_promptForInput_ext
 - IDTechSDK::IDT_NEO2, [531](#)
- pin_promptForKeyInput
 - IDTechSDK::IDT_L100, [311](#), [313](#)
 - IDTechSDK::IDT_L80, [336](#)
- pin_promptForNumericKeyWithSwipe
 - IDTechSDK::IDT_NEO2, [532](#)
- pin_promptForNumericKeyWithSwipe_ext
 - IDTechSDK::IDT_NEO2, [533](#)
- pin_sendBeep
 - IDTechSDK::IDT_K100, [245](#)
 - IDTechSDK::IDT_L100, [315](#)
 - IDTechSDK::IDT_L80, [337](#)
- IDTechSDK::IDT_NEO2, [534](#)
- pin_setKeyPressCapture
 - IDTechSDK::IDT_L100, [316](#)
 - IDTechSDK::IDT_L80, [337](#)
- SDK_Version
 - IDTechSDK::IDT_Augusta, [139](#)
 - IDTechSDK::IDT_BTMag, [172](#)
 - IDTechSDK::IDT_BTPay, [223](#)
 - IDTechSDK::IDT_CM100, [233](#)
 - IDTechSDK::IDT_K100, [245](#)
 - IDTechSDK::IDT_KioskIII, [285](#)
 - IDTechSDK::IDT_L100, [316](#)
 - IDTechSDK::IDT_L80, [337](#)
 - IDTechSDK::IDT_MiniSmartII, [380](#)
 - IDTechSDK::IDT_NEO2, [534](#)
 - IDTechSDK::IDT_PIP, [562](#)
 - IDTechSDK::IDT_SREDKey2, [694](#)
 - IDTechSDK::IDT_SecureKey, [585](#)
 - IDTechSDK::IDT_SecureMag, [611](#)
 - IDTechSDK::IDT_SpectrumPro, [656](#)
 - IDTechSDK::IDT_UniPay, [713](#)
 - IDTechSDK::IDT_UniPayI_V, [756](#)
 - IDTechSDK::IDT_VP3300, [874](#)
 - IDTechSDK::IDT_VP8800, [954](#)
 - IDTechSDK::IDT_VendIII, [811](#)
 - IDTechSDK::IDT_Vendi, [783](#)
- setCallback
 - IDTechSDK::IDT_Augusta, [140](#)
 - IDTechSDK::IDT_BTMag, [172](#)
 - IDTechSDK::IDT_BTPay, [223](#), [224](#)
 - IDTechSDK::IDT_CM100, [233](#)
 - IDTechSDK::IDT_K100, [245](#), [246](#)
 - IDTechSDK::IDT_KioskIII, [285](#)
 - IDTechSDK::IDT_L100, [316](#)
 - IDTechSDK::IDT_L80, [337](#), [338](#)
 - IDTechSDK::IDT_MiniSmartII, [380](#), [381](#)
 - IDTechSDK::IDT_NEO2, [534](#)
 - IDTechSDK::IDT_PIP, [563](#)
 - IDTechSDK::IDT_SREDKey2, [694](#), [695](#)
 - IDTechSDK::IDT_SecureKey, [585](#)
 - IDTechSDK::IDT_SecureMag, [611](#), [612](#)
 - IDTechSDK::IDT_SpectrumPro, [657](#)
 - IDTechSDK::IDT_UniPay, [713](#)
 - IDTechSDK::IDT_UniPayI_V, [757](#)
 - IDTechSDK::IDT_VP3300, [874](#)
 - IDTechSDK::IDT_VP8800, [954](#)
 - IDTechSDK::IDT_VendIII, [812](#)
 - IDTechSDK::IDT_Vendi, [784](#)
- setCallbackIP
 - IDTechSDK::IDT_NEO2, [534](#)
- setCardNotificationCallback
 - IDTechSDK::IDT_SpectrumPro, [657](#)
- setCommandTimeout
 - IDTechSDK::IDT_Augusta, [140](#)
 - IDTechSDK::IDT_BTMag, [172](#)
 - IDTechSDK::IDT_BTPay, [224](#)
 - IDTechSDK::IDT_CM100, [234](#)
 - IDTechSDK::IDT_K100, [246](#)

- IDTechSDK::IDT_KioskIII, [286](#)
- IDTechSDK::IDT_L100, [317](#)
- IDTechSDK::IDT_L80, [338](#)
- IDTechSDK::IDT_MiniSmartII, [381](#)
- IDTechSDK::IDT_NEO2, [535](#)
- IDTechSDK::IDT_PIP, [563](#)
- IDTechSDK::IDT_SREDKey2, [695](#)
- IDTechSDK::IDT_SecureKey, [586](#)
- IDTechSDK::IDT_SecureMag, [612](#)
- IDTechSDK::IDT_SpectrumPro, [657](#)
- IDTechSDK::IDT_UniPay, [713](#)
- IDTechSDK::IDT_UniPayI_V, [757](#)
- IDTechSDK::IDT_VP3300, [874](#)
- IDTechSDK::IDT_VP8800, [955](#)
- IDTechSDK::IDT_Vendi, [784](#)
- setLongPressCallback
 - IDTechSDK::IDT_NEO2, [535](#)
- SharedController
 - IDTechSDK::IDT_Augusta, [140](#)
 - IDTechSDK::IDT_BTMag, [174](#)
 - IDTechSDK::IDT_BTPay, [225](#)
 - IDTechSDK::IDT_CM100, [234](#)
 - IDTechSDK::IDT_K100, [247](#)
 - IDTechSDK::IDT_KioskIII, [287](#)
 - IDTechSDK::IDT_L100, [318](#)
 - IDTechSDK::IDT_L80, [339](#)
 - IDTechSDK::IDT_MiniSmartII, [382](#)
 - IDTechSDK::IDT_NEO2, [536](#)
 - IDTechSDK::IDT_PIP, [563](#)
 - IDTechSDK::IDT_SREDKey2, [695](#)
 - IDTechSDK::IDT_SecureKey, [586](#)
 - IDTechSDK::IDT_SecureMag, [612](#)
 - IDTechSDK::IDT_SpectrumPro, [659](#)
 - IDTechSDK::IDT_UniPay, [714](#)
 - IDTechSDK::IDT_UniPayI_V, [757](#)
 - IDTechSDK::IDT_VP3300, [875](#)
 - IDTechSDK::IDT_VP8800, [956](#)
 - IDTechSDK::IDT_VendIII, [813](#)
 - IDTechSDK::IDT_Vendi, [785](#)
- useSerialPort
 - IDTechSDK::IDT_BTMag, [173](#)
 - IDTechSDK::IDT_BTPay, [224](#)
 - IDTechSDK::IDT_K100, [246](#)
 - IDTechSDK::IDT_KioskIII, [286](#)
 - IDTechSDK::IDT_L100, [317](#)
 - IDTechSDK::IDT_L80, [338](#)
 - IDTechSDK::IDT_MiniSmartII, [381](#)
 - IDTechSDK::IDT_NEO2, [535](#), [536](#)
 - IDTechSDK::IDT_SpectrumPro, [658](#)
 - IDTechSDK::IDT_VP3300, [875](#)
 - IDTechSDK::IDT_VP8800, [955](#)
 - IDTechSDK::IDT_VendIII, [812](#)
 - IDTechSDK::IDT_Vendi, [784](#)
- useSerialPortLinux
 - IDTechSDK::IDT_BTMag, [173](#), [174](#)
 - IDTechSDK::IDT_BTPay, [225](#)
 - IDTechSDK::IDT_K100, [247](#)
 - IDTechSDK::IDT_KioskIII, [286](#), [287](#)
- IDTechSDK::IDT_L100, [317](#), [318](#)
- IDTechSDK::IDT_L80, [339](#)
- IDTechSDK::IDT_MiniSmartII, [382](#)
- IDTechSDK::IDT_NEO2, [536](#)
- IDTechSDK::IDT_SpectrumPro, [658](#)
- IDTechSDK::IDT_VP8800, [955](#), [956](#)
- IDTechSDK::IDT_VendIII, [813](#)
- IDTechSDK::IDT_Vendi, [785](#)