



IDTech Universal SDK Guide

for C / C++ Developers

#80144504-001

Rev. A

Revision History

Revision	Description and Reason for Change	Date
A	Initial Release - Manual;User;C/C++;SDK	02/14/2017

Contents

1	ID TECH Universal SDK Reference Guide for Linux/Windows/Mac (C++)	1
1.1	Demo Apps	1
1.2	Purpose	1
2	Important Security Notice	3
2.1	Applicability	3
2.2	What Does PA-DSS Mean to You?	3
2.3	Third Party Applications	4
2.4	PA-DSS Guidelines	4
2.5	More Information	9
3	Main Transaction Commands	11
3.1	EMV Methods	11
3.2	MSR Methods	12
4	Core Implementation: C/C++	13
4.1	Integrating with IdTechSDK	13
4.2	Import the Necessary Libraries	13
4.3	Add Include Statements to Use Libraries	14
4.4	Implement the Callback Function	14
4.5	Initialize SDK and Set the Target Device	14
5	LCD Foreign Language Mapping Table	16
6	Error Code Reference	18
7	Enumeration Reference	23
8	EMV Callback	29
9	EMV Tag Reference	30
10	File Index	43
10.1	File List	43

11 File Documentation	44
11.1 Source_C/libIDT_Augusta.h File Reference	44
11.1.1 Detailed Description	47
11.1.2 Macro Definition Documentation	47
11.1.2.1 IN	47
11.1.2.2 IN_OUT	47
11.1.2.3 OUT	47
11.1.3 Typedef Documentation	47
11.1.3.1 ftpComm_callBack	47
11.1.3.2 httpComm_callBack	47
11.1.3.3 pCMR_callBack	48
11.1.3.4 pCSFS_callBack	48
11.1.3.5 pEMV_callBack	48
11.1.3.6 pFW_callBack	48
11.1.3.7 pMessageHotplug	48
11.1.3.8 pMSR_callBack	48
11.1.3.9 pMSR_callBackp	48
11.1.3.10 pPIN_callBack	48
11.1.3.11 pReadDataLog	49
11.1.3.12 pSendDataLog	49
11.1.3.13 v4Comm_callBack	49
11.1.4 Function Documentation	49
11.1.4.1 config_getBeeperController()	49
11.1.4.2 config_getEncryptionControl()	49
11.1.4.3 config_getLEDController()	50
11.1.4.4 config_getModelNumber()	50
11.1.4.5 config_getModelNumber_Len()	51
11.1.4.6 config_getSerialNumber()	51
11.1.4.7 config_getSerialNumber_Len()	51
11.1.4.8 config_setBeeperController()	52
11.1.4.9 config_setEncryptionControl()	52
11.1.4.10 config_setLEDController()	53
11.1.4.11 device_cancelTransaction()	53
11.1.4.12 device_close()	53
11.1.4.13 device_controlBeep()	54
11.1.4.14 device_controlLED()	54
11.1.4.15 device_controlLED_ICC()	55
11.1.4.16 device_controlLED_MSR()	55
11.1.4.17 device_getCurrentDeviceType()	56
11.1.4.18 device_getDRS()	56

11.1.4.19 device_getFirmwareVersion()	57
11.1.4.20 device_getFirmwareVersion_Len()	57
11.1.4.21 device_getKeyStatus()	58
11.1.4.22 device_getResponseCodeString()	58
11.1.4.23 device_getSDKWaitTime()	69
11.1.4.24 device_getThreadStackSize()	69
11.1.4.25 device_init()	69
11.1.4.26 device_isAttached()	69
11.1.4.27 device_isConnected()	70
11.1.4.28 device_rebootDevice()	70
11.1.4.29 device_registerCameraCallBk()	70
11.1.4.30 device_registerCardStatusFrontSwitchCallBk()	70
11.1.4.31 device_registerFWCallBk()	70
11.1.4.32 device_SendDataCommand()	71
11.1.4.33 device_setCurrentDevice()	71
11.1.4.34 device_setSDKWaitTime()	72
11.1.4.35 device_setThreadStackSize()	72
11.1.4.36 device_setTransactionExponent()	72
11.1.4.37 device_startTransaction()	72
11.1.4.38 device_updateFirmware()	73
11.1.4.39 emv_activateTransaction()	74
11.1.4.40 emv_allowFallback()	74
11.1.4.41 emv_authenticateTransaction()	75
11.1.4.42 emv_authenticateTransactionWithTimeout()	75
11.1.4.43 emv_callbackResponseLCD()	76
11.1.4.44 emv_callbackResponseMSR()	76
11.1.4.45 emv_cancelTransaction()	77
11.1.4.46 emv_completeTransaction()	77
11.1.4.47 emv_getAutoAuthenticateTransaction()	78
11.1.4.48 emv_getAutoCompleteTransaction()	78
11.1.4.49 emv_getEMVConfigurationCheckValue()	78
11.1.4.50 emv_getEMVKernelCheckValue()	79
11.1.4.51 emv_getEMVKernelVersion()	79
11.1.4.52 emv_getEMVKernelVersion_Len()	79
11.1.4.53 emv_registerCallBk()	80
11.1.4.54 emv_removeAllApplicationData()	80
11.1.4.55 emv_removeAllCAPK()	80
11.1.4.56 emv_removeAllCRL()	80
11.1.4.57 emv_removeApplicationData()	80
11.1.4.58 emv_removeCAPK()	81

11.1.4.59 emv_removeCRL()	81
11.1.4.60 emv_removeTerminalData()	82
11.1.4.61 emv_retrieveAIDList()	82
11.1.4.62 emv_retrieveApplicationData()	82
11.1.4.63 emv_retrieveCAPK()	83
11.1.4.64 emv_retrieveCAPKList()	83
11.1.4.65 emv_retrieveCRL()	84
11.1.4.66 emv_retrieveTerminalData()	84
11.1.4.67 emv_retrieveTerminalID()	85
11.1.4.68 emv_retrieveTerminalID_Len()	85
11.1.4.69 emv_retrieveTransactionResult()	85
11.1.4.70 emv_setApplicationData()	86
11.1.4.71 emv_setAutoAuthenticateTransaction()	86
11.1.4.72 emv_setAutoCompleteTransaction()	86
11.1.4.73 emv_setCAPK()	87
11.1.4.74 emv_setCRL()	87
11.1.4.75 emv_setTerminalData()	88
11.1.4.76 emv_setTerminalID()	88
11.1.4.77 emv_startTransaction()	88
11.1.4.78 icc_disable()	89
11.1.4.79 icc_enable()	89
11.1.4.80 icc_exchangeAPDU()	90
11.1.4.81 icc_exchangeEncryptedAPDU()	90
11.1.4.82 icc_getAPDU_KSN()	91
11.1.4.83 icc_getFunctionStatus()	91
11.1.4.84 icc_getICCReaderStatus()	92
11.1.4.85 icc_getKeyFormatForICCDUKPT()	92
11.1.4.86 icc_getKeyTypeForICCDUKPT()	93
11.1.4.87 icc_powerOffICC()	93
11.1.4.88 icc_powerOnICC()	93
11.1.4.89 msr_cancelMSRSwipe()	94
11.1.4.90 msr_captureMode()	94
11.1.4.91 msr_disable()	94
11.1.4.92 msr_getClearPANID()	95
11.1.4.93 msr_getExpirationMask()	95
11.1.4.94 msr_getKeyFormatForICCDUKPT()	95
11.1.4.95 msr_getKeyTypeForICCDUKPT()	96
11.1.4.96 msr_getMSRData()	96
11.1.4.97 msr_getSetting()	96
11.1.4.98 msr_getSwipeForcedEncryptionOption()	98

11.1.4.99	msr_getSwipeMaskOption()	98
11.1.4.100	msr_registerCallBk()	98
11.1.4.101	msr_registerCallBkp()	99
11.1.4.102	msr_setClearPANID()	99
11.1.4.103	msr_setExpirationMask()	99
11.1.4.104	msr_setKeyFormatForICCDUKPT()	99
11.1.4.105	msr_setKeyTypeForICCDUKPT()	100
11.1.4.106	msr_setSetting()	100
11.1.4.107	msr_setSwipeForcedEncryptionOption()	101
11.1.4.108	msr_setSwipeMaskOption()	101
11.1.4.109	msr_startMSRSwipe()	101
11.1.4.110	parseMSRData()	102
11.1.4.111	pin_cancelPINEntry()	102
11.1.4.112	pin_registerCallBk()	102
11.1.4.113	registerHotplugCallBk()	102
11.1.4.114	registerLogCallBk()	103
11.1.4.115	SDK_Version()	103
11.1.4.116	setAbsoluteLibraryPath()	103
11.2	Source_C/libIDT_Device.h File Reference	103
11.2.1	Detailed Description	112
11.2.2	Macro Definition Documentation	112
11.2.2.1	IN	112
11.2.2.2	IN_OUT	112
11.2.2.3	OUT	112
11.2.3	Typedef Documentation	112
11.2.3.1	ftpComm_callBack	113
11.2.3.2	httpComm_callBack	113
11.2.3.3	pCMR_callBack	113
11.2.3.4	pCSFS_callBack	113
11.2.3.5	pEMV_callBack	113
11.2.3.6	pFW_callBack	113
11.2.3.7	pLCD_callBack	113
11.2.3.8	pLog_callback	113
11.2.3.9	pMessageHotplug	114
11.2.3.10	pMSR_callBack	114
11.2.3.11	pMSR_callBackp	114
11.2.3.12	pPIN_callBack	114
11.2.3.13	pReadDataLog	114
11.2.3.14	pRKI_callBack	114
11.2.3.15	pSendDataLog	114

11.2.3.16 pWN_callback	114
11.2.3.17 pWP_callback	115
11.2.3.18 v4Comm_callback	115
11.2.4 Function Documentation	115
11.2.4.1 cancelWorldNet()	115
11.2.4.2 cancelWorldPay()	115
11.2.4.3 config_getBeeperController()	115
11.2.4.4 config_getEncryptionControl()	116
11.2.4.5 config_getLEDController()	116
11.2.4.6 config_getModelNumber()	117
11.2.4.7 config_getModelNumber_Len()	117
11.2.4.8 config_getSerialNumber()	117
11.2.4.9 config_getSerialNumber_Len()	118
11.2.4.10 config_setBeeperController()	118
11.2.4.11 config_setCmdTimeOutDuration()	118
11.2.4.12 config_setEncryptionControl()	119
11.2.4.13 config_setLEDController()	119
11.2.4.14 ctls_activateTransaction()	120
11.2.4.15 ctls_cancelTransaction()	121
11.2.4.16 ctls_displayOnlineAuthResult()	122
11.2.4.17 ctls_getAllConfigurationGroups()	122
11.2.4.18 ctls_getConfigurationGroup()	122
11.2.4.19 ctls_registerCallBkp()	123
11.2.4.20 ctls_removeAllApplicationData()	123
11.2.4.21 ctls_removeAllCAPK()	123
11.2.4.22 ctls_removeApplicationData()	123
11.2.4.23 ctls_removeCAPK()	124
11.2.4.24 ctls_removeConfigurationGroup()	124
11.2.4.25 ctls_retrieveAIDList()	124
11.2.4.26 ctls_retrieveApplicationData()	125
11.2.4.27 ctls_retrieveCAPK()	125
11.2.4.28 ctls_retrieveCAPKList()	126
11.2.4.29 ctls_retrieveTerminalData()	126
11.2.4.30 ctls_setApplicationData()	127
11.2.4.31 ctls_setCAPK()	127
11.2.4.32 ctls_setConfigurationGroup()	128
11.2.4.33 ctls_setTerminalData()	128
11.2.4.34 ctls_startTransaction()	129
11.2.4.35 device_activateTransaction()	130
11.2.4.36 device_buzzerOnOff()	132

11.2.4.37 device_calibrateParameters()	132
11.2.4.38 device_cancelTransaction()	132
11.2.4.39 device_cancelTransactionSilent()	132
11.2.4.40 device_close()	133
11.2.4.41 device_configureButtons()	133
11.2.4.42 device_controlBeep()	133
11.2.4.43 device_controlIndicator()	134
11.2.4.44 device_controlLED()	134
11.2.4.45 device_controlLED_ICC()	135
11.2.4.46 device_controlLED_MSR()	135
11.2.4.47 device_controlUserInterface()	136
11.2.4.48 device_createDirectory()	138
11.2.4.49 device_deleteDirectory()	138
11.2.4.50 device_deleteFile()	138
11.2.4.51 device_disableBlueLED()	139
11.2.4.52 device_enableBlueLED()	139
11.2.4.53 device_enableExternalLCDMessages()	139
11.2.4.54 device_enableL100PassThrough()	140
11.2.4.55 device_enableL80PassThrough()	140
11.2.4.56 device_enablePassThrough()	140
11.2.4.57 device_enableRFAntenna()	141
11.2.4.58 device_enhancedPassthrough()	141
11.2.4.59 device_enterStopMode()	142
11.2.4.60 device_getAudioVolume()	142
11.2.4.61 device_getButtonConfiguration()	142
11.2.4.62 device_getCameraParameters()	143
11.2.4.63 device_getCurrentDeviceType()	143
11.2.4.64 device_getDateTime()	143
11.2.4.65 device_getDateTime_Len()	144
11.2.4.66 device_getDeviceMemoryUsageInfo()	144
11.2.4.67 device_getDeviceTreeVersion()	145
11.2.4.68 device_getDriveFreeSpace()	145
11.2.4.69 device_getDRS()	145
11.2.4.70 device_getFirmwareVersion()	146
11.2.4.71 device_getFirmwareVersion_Len()	146
11.2.4.72 device_getIDGStatusCodeString()	147
11.2.4.73 device_getKeyStatus()	148
11.2.4.74 device_getL100PassThroughMode()	149
11.2.4.75 device_getL80PassThroughMode()	149
11.2.4.76 device_getMerchantRecord()	150

11.2.4.77 device_getMerchantRecord_Len()	150
11.2.4.78 device_getNEOAltDevice()	151
11.2.4.79 device_getResponseCodeString()	151
11.2.4.80 device_getRTCDateTime()	161
11.2.4.81 device_getSDKWaitTime()	161
11.2.4.82 device_getSpectrumProKSN()	162
11.2.4.83 device_getSpectrumProKSN_Len()	162
11.2.4.84 device_getTamperStatus()	163
11.2.4.85 device_getThreadStackSize()	163
11.2.4.86 device_init()	163
11.2.4.87 device_isAttached()	163
11.2.4.88 device_isConnected()	164
11.2.4.89 device_lcdDisplayClear()	164
11.2.4.90 device_lcdDisplayLine1Message()	164
11.2.4.91 device_lcdDisplayLine2Message()	165
11.2.4.92 device_listDirectory()	165
11.2.4.93 device_loadCertCA()	165
11.2.4.94 device_outputLog()	166
11.2.4.95 device_pingDevice()	166
11.2.4.96 device_playAudio()	166
11.2.4.97 device_pollCardReader()	167
11.2.4.98 device_pollCardReader_Len()	169
11.2.4.99 device_pollForToken()	170
11.2.4.100 device_queryFile()	171
11.2.4.101 device_readFileFromSD()	171
11.2.4.102 device_rebootDevice()	172
11.2.4.103 device_registerCameraCallBk()	172
11.2.4.104 device_registerCardStatusFrontSwitchCallBk()	172
11.2.4.105 device_registerFWCallBk()	173
11.2.4.106 device_registerRKICallBk()	173
11.2.4.107 device_rrcConnect()	173
11.2.4.108 device_rrcDisconnect()	173
11.2.4.109 device_rrcDownloadApp()	173
11.2.4.110 device_rrcInstallApp()	174
11.2.4.111 device_rrcRunApp()	174
11.2.4.112 device_rrcUninstallApp()	174
11.2.4.113 device_selfCheck()	175
11.2.4.114 device_SendDataCommand()	175
11.2.4.115 device_SendDataCommandITP()	175
11.2.4.116 device_SendDataCommandNEO()	176

11.2.4.117	device_setAudioVolume()	176
11.2.4.118	device_setBurstMode()	177
11.2.4.119	device_setCameraParameters()	177
11.2.4.120	device_setCancelTransactionMode()	177
11.2.4.121	device_setConfigPath()	178
11.2.4.122	device_setCoreDumpLogFile()	178
11.2.4.123	device_setCurrentDevice()	179
11.2.4.124	device_setMerchantRecord()	179
11.2.4.125	device_setNEO2DevicesConfigs()	179
11.2.4.126	device_setNEOAltDevice()	180
11.2.4.127	device_setNEOGen()	180
11.2.4.128	device_setPollMode()	180
11.2.4.129	device_setRKI_URL()	181
11.2.4.130	device_setRTCDateTime()	181
11.2.4.131	device_setSDKWaitTime()	181
11.2.4.132	device_setSleepModeTime()	182
11.2.4.133	device_setSystemLanguage()	182
11.2.4.134	device_setThreadStackSize()	183
11.2.4.135	device_setTransactionExponent()	183
11.2.4.136	device_startListenNotifications()	183
11.2.4.137	device_startQRCodeScan()	183
11.2.4.138	device_startQRCodeScanWithDisplayWindowInfo()	184
11.2.4.139	device_startRKI()	184
11.2.4.140	device_startTakingPhoto()	185
11.2.4.141	device_startTransaction()	185
11.2.4.142	device_stopAudio()	187
11.2.4.143	device_stopListenNotifications()	187
11.2.4.144	device_stopQRCodeScan()	187
11.2.4.145	device_stopTakingPhoto()	187
11.2.4.146	device_toSDCard()	187
11.2.4.147	device_transferFile()	188
11.2.4.148	device_turnOffYellowLED()	188
11.2.4.149	device_turnOnYellowLED()	188
11.2.4.150	device_updateFirmware()	189
11.2.4.151	device_verifyBackdoorKey()	189
11.2.4.152	emv_activateTransaction()	190
11.2.4.153	emv_allowFallback()	190
11.2.4.154	emv_authenticateTransaction()	190
11.2.4.155	emv_authenticateTransactionWithTimeout()	191
11.2.4.156	emv_callbackResponseLCD()	192

11.2.4.157emv_callbackResponseMSR()	192
11.2.4.158emv_cancelTransaction()	192
11.2.4.159emv_completeTransaction()	193
11.2.4.160emv_getAutoAuthenticateTransaction()	193
11.2.4.161emv_getAutoCompleteTransaction()	194
11.2.4.162emv_getEMVConfigurationCheckValue()	194
11.2.4.163emv_getEMVKernelCheckValue()	194
11.2.4.164emv_getEMVKernelVersion()	194
11.2.4.165emv_getEMVKernelVersion_Len()	195
11.2.4.166emv_registerCallBk()	195
11.2.4.167emv_removeAllApplicationData()	195
11.2.4.168emv_removeAllCAPK()	195
11.2.4.169emv_removeAllCRL()	196
11.2.4.170emv_removeApplicationData()	196
11.2.4.171emv_removeCAPK()	196
11.2.4.172emv_removeCRL()	197
11.2.4.173emv_removeTerminalData()	197
11.2.4.174emv_retrieveAIDList()	197
11.2.4.175emv_retrieveApplicationData()	198
11.2.4.176emv_retrieveCAPK()	198
11.2.4.177emv_retrieveCAPKList()	199
11.2.4.178emv_retrieveCRL()	199
11.2.4.179emv_retrieveTerminalData()	200
11.2.4.180emv_retrieveTerminalID()	200
11.2.4.181emv_retrieveTerminalID_Len()	200
11.2.4.182emv_retrieveTransactionResult()	201
11.2.4.183emv_setApplicationData()	201
11.2.4.184emv_setApplicationDataTLV()	202
11.2.4.185emv_setAutoAuthenticateTransaction()	202
11.2.4.186emv_setAutoCompleteTransaction()	202
11.2.4.187emv_setCAPK()	203
11.2.4.188emv_setCRL()	203
11.2.4.189emv_setTerminalData()	204
11.2.4.190emv_setTerminalID()	204
11.2.4.191emv_setTerminalMajorConfiguration()	204
11.2.4.192emv_setTransactionParameters()	205
11.2.4.193emv_startTransaction()	205
11.2.4.194executeTransaction()	206
11.2.4.195executeTransaction_WorldNet()	207
11.2.4.196felica_authentication()	207

11.2.4.197felica_cancelCodeEntry()	208
11.2.4.198felica_getCode()	208
11.2.4.199felica_poll()	208
11.2.4.200felica_read()	209
11.2.4.201felica_readWithMac()	209
11.2.4.202felica_requestService()	210
11.2.4.203felica_SendCommand()	210
11.2.4.204felica_write()	211
11.2.4.205felica_writeWithMac()	211
11.2.4.206forwardTransaction()	212
11.2.4.207forwardTransaction_WorldNet()	212
11.2.4.208icc_disable()	213
11.2.4.209icc_enable()	213
11.2.4.210icc_exchangeAPDU()	213
11.2.4.211icc_exchangeEncryptedAPDU()	214
11.2.4.212icc_getAPDU_KSN()	215
11.2.4.213icc_getFunctionStatus()	215
11.2.4.214icc_getICCReaderStatus()	215
11.2.4.215icc_getKeyFormatForICCDUKPT()	216
11.2.4.216icc_getKeyTypeForICCDUKPT()	216
11.2.4.217icc_powerOffICC()	216
11.2.4.218icc_powerOnICC()	217
11.2.4.219icc_setKeyFormatForICCDUKPT()	217
11.2.4.220icc_setKeyTypeForICCDUKPT()	218
11.2.4.221iso8583_deserializeFromXML()	218
11.2.4.222iso8583_displayMessage()	218
11.2.4.223iso8583_freeMessage()	219
11.2.4.224iso8583_get1987Handler()	219
11.2.4.225iso8583_get1993Handler()	219
11.2.4.226iso8583_get2003Handler()	220
11.2.4.227iso8583_getField()	220
11.2.4.228iso8583_getMessageField()	220
11.2.4.229iso8583_initializeMessage()	221
11.2.4.230iso8583_packMessage()	221
11.2.4.231iso8583_removeMessageField()	221
11.2.4.232iso8583_serializeToXML()	222
11.2.4.233iso8583_setMessageField()	222
11.2.4.234iso8583_unpackMessage()	223
11.2.4.235cd_addButton()	223
11.2.4.236cd_addEthernet()	224

11.2.4.237cd_addExtVideo()	225
11.2.4.238cd_addImage()	226
11.2.4.239cd_addItemToList()	227
11.2.4.240cd_addLED()	228
11.2.4.241lcd_addText()	229
11.2.4.242cd_addVideo()	232
11.2.4.243cd_cancelSlideShow()	233
11.2.4.244cd_captureSignature()	233
11.2.4.245cd_clearDisplay()	233
11.2.4.246cd_clearEventQueue()	234
11.2.4.247cd_clearScreenInfo()	234
11.2.4.248cd_cloneScreen()	234
11.2.4.249cd_createInputField()	235
11.2.4.250cd_createInputField_Len()	236
11.2.4.251lcd_createList()	237
11.2.4.252cd_createList_Len()	238
11.2.4.253cd_createScreen()	240
11.2.4.254cd_customDisplayMode()	240
11.2.4.255cd_destroyScreen()	241
11.2.4.256cd_displayButton()	241
11.2.4.257cd_displayButton_Len()	243
11.2.4.258cd_displayMessage()	244
11.2.4.259cd_displayParagraph()	244
11.2.4.260cd_displayPrompt()	246
11.2.4.261lcd_displayText()	246
11.2.4.262cd_displayText_Len()	247
11.2.4.263cd_enableBacklight()	249
11.2.4.264cd_getActiveScreen()	249
11.2.4.265cd_getAllObjects()	249
11.2.4.266cd_getAllScreens()	250
11.2.4.267cd_getBacklightStatus()	250
11.2.4.268cd_getButtonEvent()	251
11.2.4.269cd_getInputEvent()	251
11.2.4.270cd_getInputEvent_Len()	253
11.2.4.271lcd_getInputFieldValue()	255
11.2.4.272cd_getSelectedListItem()	255
11.2.4.273cd_getSelectedListItem_Len()	256
11.2.4.274cd_linkUIWithTransactionMessageId()	256
11.2.4.275cd_loadScreenInfo()	257
11.2.4.276cd_queryObjectbyID()	257

11.2.4.277cd_queryObjectbyName()	257
11.2.4.278cd_queryScreenbyID()	258
11.2.4.279cd_queryScreenbyName()	258
11.2.4.280cd_registerCallBk()	259
11.2.4.281lcd_removeItem()	259
11.2.4.282cd_resetInitialState()	259
11.2.4.283cd_savePrompt()	259
11.2.4.284cd_setBackgroundImage()	260
11.2.4.285cd_setBacklight()	260
11.2.4.286cd_setDisplayImage()	261
11.2.4.287cd_setForeBackColor()	261
11.2.4.288cd_showScreen()	262
11.2.4.289cd_startSlideShow()	262
11.2.4.290cd_storeScreenInfo()	263
11.2.4.291lcd_updateColor()	263
11.2.4.292cd_updateLabel()	264
11.2.4.293cd_updatePosition()	265
11.2.4.294loyalty_cancelTransaction()	265
11.2.4.295loyalty_cancelTransactionSilent()	265
11.2.4.296loyalty_registerCallBk()	266
11.2.4.297loyalty_startTransaction()	266
11.2.4.298msr_cancelMSRSwipe()	268
11.2.4.299msr_captureMode()	268
11.2.4.300msr_clearMSRData()	268
11.2.4.301msr_disable()	269
11.2.4.302msr_flushTrackData()	269
11.2.4.303msr_getClearPANID()	269
11.2.4.304msr_getExpirationMask()	269
11.2.4.305msr_getFunctionStatus()	270
11.2.4.306msr_getKeyFormatForICCDUKPT()	270
11.2.4.307msr_getKeyTypeForICCDUKPT()	271
11.2.4.308msr_getMSRData()	271
11.2.4.309msr_getSwipeForcedEncryptionOption()	271
11.2.4.310msr_getSwipeMaskOption()	272
11.2.4.311msr_registerCallBk()	272
11.2.4.312msr_registerCallBkp()	272
11.2.4.313msr_setClearPANID()	272
11.2.4.314msr_setExpirationMask()	273
11.2.4.315msr_setKeyFormatForICCDUKPT()	273
11.2.4.316msr_setKeyTypeForICCDUKPT()	273

11.2.4.317msr_setSwipeForcedEncryptionOption()	274
11.2.4.318msr_setSwipeMaskOption()	274
11.2.4.319msr_startMSRSwipe()	275
11.2.4.320parseMSRData()	275
11.2.4.321parsePINBlockData()	275
11.2.4.322parsePINData()	276
11.2.4.323pin_cancelPINEntry()	276
11.2.4.324pin_capturePin()	276
11.2.4.325pin_capturePinExt()	277
11.2.4.326pin_getEncryptedOnlinePIN()	278
11.2.4.327pin_getEncryptedPIN()	279
11.2.4.328pin_getFunctionKey()	279
11.2.4.329pin_getPAN()	280
11.2.4.330pin_getPanEntry()	280
11.2.4.331pin_getPIN()	281
11.2.4.332pin_inputFromPrompt()	281
11.2.4.333pin_promptCreditDebit()	282
11.2.4.334pin_promptForAmount()	283
11.2.4.335pin_promptForAmountInput()	283
11.2.4.336pin_promptForKeyInput()	286
11.2.4.337pin_promptForNumericKey()	290
11.2.4.338pin_promptForNumericKeyWithSwipe()	290
11.2.4.339pin_registerCallBk()	291
11.2.4.340pin_sendBeep()	291
11.2.4.341pin_setKeyValues()	292
11.2.4.342registerHotplugCallBk()	292
11.2.4.343registerLogCallBk()	292
11.2.4.344rs232_device_init()	292
11.2.4.345SDK_Version()	294
11.2.4.346set_open_com_port_timeout()	294
11.2.4.347setAbsoluteLibraryPath()	294
11.2.4.348ws_deleteSSLCert()	295
11.2.4.349ws_getCertChainType()	295
11.2.4.350ws_loadSSLCert()	295
11.2.4.351ws_requestCSR()	296
11.2.4.352ws_revokeSSLCert()	296
11.2.4.353ws_updateRootCertificate()	296
11.3 Source_C/libIDT_KioskIII.h File Reference	297
11.3.1 Detailed Description	299
11.3.2 Macro Definition Documentation	299

11.3.2.1	IN	299
11.3.2.2	IN_OUT	299
11.3.2.3	OUT	299
11.3.3	Typedef Documentation	299
11.3.3.1	ftpComm_callBack	299
11.3.3.2	httpComm_callBack	299
11.3.3.3	pCMR_callBack	299
11.3.3.4	pCSFS_callBack	300
11.3.3.5	pEMV_callBack	300
11.3.3.6	pMessageHotplug	300
11.3.3.7	pMSR_callBack	300
11.3.3.8	pMSR_callBackp	300
11.3.3.9	pPIN_callBack	300
11.3.3.10	pReadDataLog	300
11.3.3.11	pSendDataLog	300
11.3.3.12	v4Comm_callBack	301
11.3.4	Function Documentation	301
11.3.4.1	config_getSerialNumber()	301
11.3.4.2	config_getSerialNumber_Len()	301
11.3.4.3	ctls_activateTransaction()	302
11.3.4.4	ctls_cancelTransaction()	303
11.3.4.5	ctls_getAllConfigurationGroups()	303
11.3.4.6	ctls_getConfigurationGroup()	303
11.3.4.7	ctls_registerCallBk()	304
11.3.4.8	ctls_registerCallBkp()	304
11.3.4.9	ctls_removeAllApplicationData()	304
11.3.4.10	ctls_removeAllCAPK()	304
11.3.4.11	ctls_removeApplicationData()	305
11.3.4.12	ctls_removeCAPK()	305
11.3.4.13	ctls_removeConfigurationGroup()	305
11.3.4.14	ctls_retrieveAIDList()	306
11.3.4.15	ctls_retrieveApplicationData()	306
11.3.4.16	ctls_retrieveCAPK()	306
11.3.4.17	ctls_retrieveCAPKList()	307
11.3.4.18	ctls_retrieveTerminalData()	308
11.3.4.19	ctls_setApplicationData()	308
11.3.4.20	ctls_setCAPK()	309
11.3.4.21	ctls_setConfigurationGroup()	309
11.3.4.22	ctls_setTerminalData()	310
11.3.4.23	ctls_startTransaction()	310

11.3.4.24 device_close()	312
11.3.4.25 device_controlUserInterface()	312
11.3.4.26 device_enablePassThrough()	314
11.3.4.27 device_getCurrentDeviceType()	314
11.3.4.28 device_getFirmwareVersion()	314
11.3.4.29 device_getFirmwareVersion_Len()	314
11.3.4.30 device_getIDGStatusCodeString()	315
11.3.4.31 device_getMerchantRecord()	316
11.3.4.32 device_getMerchantRecord_Len()	317
11.3.4.33 device_getSDKWaitTime()	317
11.3.4.34 device_getThreadStackSize()	318
11.3.4.35 device_getTransactionResults()	318
11.3.4.36 device_init()	318
11.3.4.37 device_isAttached()	318
11.3.4.38 device_isConnected()	319
11.3.4.39 device_pingDevice()	319
11.3.4.40 device_registerCameraCallBk()	319
11.3.4.41 device_registerCardStatusFrontSwitchCallBk()	319
11.3.4.42 device_SendDataCommandNEO()	319
11.3.4.43 device_setBurstMode()	320
11.3.4.44 device_setCurrentDevice()	320
11.3.4.45 device_setMerchantRecord()	321
11.3.4.46 device_setPollMode()	321
11.3.4.47 device_setRKI_URL()	322
11.3.4.48 device_setSDKWaitTime()	322
11.3.4.49 device_setThreadStackSize()	322
11.3.4.50 device_startRKI()	322
11.3.4.51 emv_registerCallBk()	323
11.3.4.52 parseMSRData()	323
11.3.4.53 pin_registerCallBk()	323
11.3.4.54 registerHotplugCallBk()	323
11.3.4.55 registerLogCallBk()	324
11.3.4.56 rs232_device_init()	324
11.3.4.57 SDK_Version()	324
11.3.4.58 setAbsoluteLibraryPath()	325
11.4 Source_C/libIDT_L100.h File Reference	325
11.4.1 Detailed Description	326
11.4.2 Macro Definition Documentation	326
11.4.2.1 IN	326
11.4.2.2 IN_OUT	327

11.4.2.3	OUT	327
11.4.3	Typedef Documentation	327
11.4.3.1	ftpComm_callBack	327
11.4.3.2	httpComm_callBack	327
11.4.3.3	pCMR_callBack	327
11.4.3.4	pCSFS_callBack	327
11.4.3.5	pEMV_callBack	327
11.4.3.6	pFW_callBack	328
11.4.3.7	pMessageHotplug	328
11.4.3.8	pMSR_callBack	328
11.4.3.9	pMSR_callBackp	328
11.4.3.10	pPIN_callBack	328
11.4.3.11	pReadDataLog	328
11.4.3.12	pSendDataLog	328
11.4.3.13	v4Comm_callBack	328
11.4.4	Function Documentation	329
11.4.4.1	config_getModelNumber()	329
11.4.4.2	config_getModelNumber_Len()	329
11.4.4.3	config_getSerialNumber()	329
11.4.4.4	config_getSerialNumber_Len()	330
11.4.4.5	device_close()	330
11.4.4.6	device_enterStopMode()	330
11.4.4.7	device_getCurrentDeviceType()	331
11.4.4.8	device_getDateTime()	331
11.4.4.9	device_getDateTime_Len()	331
11.4.4.10	device_getFirmwareVersion()	331
11.4.4.11	device_getFirmwareVersion_Len()	332
11.4.4.12	device_getKeyStatus()	332
11.4.4.13	device_getResponseCodeString()	333
11.4.4.14	device_init()	343
11.4.4.15	device_isAttached()	343
11.4.4.16	device_isConnected()	344
11.4.4.17	device_rebootDevice()	344
11.4.4.18	device_registerCameraCallBk()	344
11.4.4.19	device_registerCardStatusFrontSwitchCallBk()	344
11.4.4.20	device_registerFWCallBk()	344
11.4.4.21	device_SendDataCommand()	344
11.4.4.22	device_setCurrentDevice()	345
11.4.4.23	device_setSleepModeTime()	345
11.4.4.24	device_updateFirmware()	346

11.4.4.25	emv_registerCallBk()	347
11.4.4.26	lcd_displayMessage()	347
11.4.4.27	lcd_displayPrompt()	347
11.4.4.28	lcd_enableBacklight()	348
11.4.4.29	lcd_getBacklightStatus()	348
11.4.4.30	lcd_savePrompt()	348
11.4.4.31	msr_registerCallBk()	349
11.4.4.32	msr_registerCallBkp()	349
11.4.4.33	pin_getEncryptedPIN()	349
11.4.4.34	pin_getFunctionKey()	350
11.4.4.35	pin_promptForAmountInput()	350
11.4.4.36	pin_promptForKeyInput()	353
11.4.4.37	pin_registerCallBk()	356
11.4.4.38	pin_sendBeep()	356
11.4.4.39	pin_setKeyValues()	357
11.4.4.40	registerHotplugCallBk()	357
11.4.4.41	registerLogCallBk()	357
11.4.4.42	SDK_Version()	357
11.4.4.43	setAbsoluteLibraryPath()	358
11.5	Source_C/libIDT_L80.h File Reference	358
11.5.1	Detailed Description	359
11.5.2	Macro Definition Documentation	360
11.5.2.1	IN	360
11.5.2.2	IN_OUT	360
11.5.2.3	OUT	360
11.5.3	Typedef Documentation	360
11.5.3.1	ftpComm_callBack	360
11.5.3.2	httpComm_callBack	360
11.5.3.3	pCMR_callBack	360
11.5.3.4	pCSFS_callBack	360
11.5.3.5	pEMV_callBack	361
11.5.3.6	pFW_callBack	361
11.5.3.7	pMessageHotplug	361
11.5.3.8	pMSR_callBack	361
11.5.3.9	pMSR_callBackp	361
11.5.3.10	pPIN_callBack	361
11.5.3.11	pReadDataLog	361
11.5.3.12	pSendDataLog	361
11.5.3.13	v4Comm_callBack	362
11.5.4	Function Documentation	362

11.5.4.1	config_getModelNumber()	362
11.5.4.2	config_getModelNumber_Len()	362
11.5.4.3	config_getSerialNumber()	363
11.5.4.4	config_getSerialNumber_Len()	363
11.5.4.5	device_close()	363
11.5.4.6	device_enterStopMode()	363
11.5.4.7	device_getCurrentDeviceType()	364
11.5.4.8	device_getDateTime()	364
11.5.4.9	device_getDateTime_Len()	364
11.5.4.10	device_getFirmwareVersion()	365
11.5.4.11	device_getFirmwareVersion_Len()	365
11.5.4.12	device_getKeyStatus()	365
11.5.4.13	device_getResponseCodeString()	366
11.5.4.14	device_init()	376
11.5.4.15	device_isAttached()	376
11.5.4.16	device_isConnected()	377
11.5.4.17	device_rebootDevice()	377
11.5.4.18	device_registerCameraCallBk()	377
11.5.4.19	device_registerCardStatusFrontSwitchCallBk()	377
11.5.4.20	device_registerFWCallBk()	378
11.5.4.21	device_SendDataCommand()	378
11.5.4.22	device_setCurrentDevice()	378
11.5.4.23	device_setSleepModeTime()	379
11.5.4.24	device_updateFirmware()	379
11.5.4.25	emv_registerCallBk()	380
11.5.4.26	lcd_displayMessage()	380
11.5.4.27	lcd_displayPrompt()	380
11.5.4.28	lcd_enableBacklight()	381
11.5.4.29	lcd_getBacklightStatus()	381
11.5.4.30	lcd_savePrompt()	381
11.5.4.31	msr_registerCallBk()	382
11.5.4.32	msr_registerCallBkp()	382
11.5.4.33	pin_getEncryptedPIN()	382
11.5.4.34	pin_getFunctionKey()	383
11.5.4.35	pin_promptForAmountInput()	383
11.5.4.36	pin_promptForKeyInput()	386
11.5.4.37	pin_registerCallBk()	389
11.5.4.38	pin_sendBeep()	390
11.5.4.39	pin_setKeyValues()	390
11.5.4.40	registerHotplugCallBk()	390

11.5.4.41 registerLogCallBk()	390
11.5.4.42 SDK_Version()	391
11.5.4.43 setAbsoluteLibraryPath()	391
11.6 Source_C/libIDT_MiniSmartII.h File Reference	391
11.6.1 Detailed Description	393
11.6.2 Macro Definition Documentation	394
11.6.2.1 IN	394
11.6.2.2 IN_OUT	394
11.6.2.3 OUT	394
11.6.3 Typedef Documentation	394
11.6.3.1 ftpComm_callBack	394
11.6.3.2 httpComm_callBack	394
11.6.3.3 pCMR_callBack	394
11.6.3.4 pCSFS_callBack	394
11.6.3.5 pEMV_callBack	395
11.6.3.6 pMessageHotplug	395
11.6.3.7 pMSR_callBack	395
11.6.3.8 pMSR_callBackp	395
11.6.3.9 pPIN_callBack	395
11.6.3.10 pReadDataLog	395
11.6.3.11 pSendDataLog	395
11.6.3.12 v4Comm_callBack	395
11.6.4 Function Documentation	396
11.6.4.1 comm_registerHTTPCallback()	396
11.6.4.2 comm_registerV4Callback()	396
11.6.4.3 config_getBeeperController()	396
11.6.4.4 config_getEncryptionControl()	396
11.6.4.5 config_getLEDController()	397
11.6.4.6 config_getModelNumber()	398
11.6.4.7 config_getModelNumber_Len()	398
11.6.4.8 config_getSerialNumber()	398
11.6.4.9 config_getSerialNumber_Len()	399
11.6.4.10 config_setBeeperController()	399
11.6.4.11 config_setEncryptionControl()	399
11.6.4.12 config_setLEDController()	400
11.6.4.13 device_cancelTransaction()	400
11.6.4.14 device_close()	401
11.6.4.15 device_controlBeep()	401
11.6.4.16 device_controlLED()	401
11.6.4.17 device_controlLED_ICC()	402

11.6.4.18 device_controlLED_MSR()	402
11.6.4.19 device_getCurrentDeviceType()	403
11.6.4.20 device_getFirmwareVersion()	403
11.6.4.21 device_getFirmwareVersion_Len()	404
11.6.4.22 device_getKeyStatus()	404
11.6.4.23 device_getResponseCodeString()	405
11.6.4.24 device_getSDKWaitTime()	415
11.6.4.25 device_getThreadStackSize()	415
11.6.4.26 device_init()	415
11.6.4.27 device_isAttached()	416
11.6.4.28 device_isConnected()	416
11.6.4.29 device_rebootDevice()	416
11.6.4.30 device_registerCameraCallBk()	416
11.6.4.31 device_registerCardStatusFrontSwitchCallBk()	416
11.6.4.32 device_SendDataCommand()	417
11.6.4.33 device_setCurrentDevice()	417
11.6.4.34 device_setSDKWaitTime()	418
11.6.4.35 device_setThreadStackSize()	418
11.6.4.36 device_setTransactionExponent()	418
11.6.4.37 device_startTransaction()	418
11.6.4.38 device_updateFirmware()	419
11.6.4.39 emv_activateTransaction()	420
11.6.4.40 emv_allowFallback()	420
11.6.4.41 emv_authenticateTransaction()	421
11.6.4.42 emv_authenticateTransactionWithTimeout()	421
11.6.4.43 emv_callbackResponseLCD()	422
11.6.4.44 emv_callbackResponseMSR()	422
11.6.4.45 emv_cancelTransaction()	423
11.6.4.46 emv_completeTransaction()	423
11.6.4.47 emv_getAutoAuthenticateTransaction()	424
11.6.4.48 emv_getAutoCompleteTransaction()	424
11.6.4.49 emv_getEMVConfigurationCheckValue()	424
11.6.4.50 emv_getEMVKernelCheckValue()	425
11.6.4.51 emv_getEMVKernelVersion()	425
11.6.4.52 emv_getEMVKernelVersion_Len()	425
11.6.4.53 emv_registerCallBk()	426
11.6.4.54 emv_removeAllApplicationData()	426
11.6.4.55 emv_removeAllCAPK()	426
11.6.4.56 emv_removeAllCRL()	426
11.6.4.57 emv_removeApplicationData()	426

11.6.4.58 emv_removeCAPK()	427
11.6.4.59 emv_removeCRL()	427
11.6.4.60 emv_removeTerminalData()	428
11.6.4.61 emv_retrieveAIDList()	428
11.6.4.62 emv_retrieveApplicationData()	428
11.6.4.63 emv_retrieveCAPK()	429
11.6.4.64 emv_retrieveCAPKList()	429
11.6.4.65 emv_retrieveCRL()	430
11.6.4.66 emv_retrieveTerminalData()	430
11.6.4.67 emv_retrieveTerminalID()	431
11.6.4.68 emv_retrieveTerminalID_Len()	431
11.6.4.69 emv_retrieveTransactionResult()	431
11.6.4.70 emv_setApplicationData()	432
11.6.4.71 emv_setAutoAuthenticateTransaction()	432
11.6.4.72 emv_setAutoCompleteTransaction()	432
11.6.4.73 emv_setCAPK()	433
11.6.4.74 emv_setCRL()	433
11.6.4.75 emv_setTerminalData()	434
11.6.4.76 emv_setTerminalID()	434
11.6.4.77 emv_startTransaction()	434
11.6.4.78 icc_disable()	435
11.6.4.79 icc_enable()	435
11.6.4.80 icc_exchangeAPDU()	436
11.6.4.81 icc_exchangeEncryptedAPDU()	436
11.6.4.82 icc_getAPDU_KSN()	437
11.6.4.83 icc_getFunctionStatus()	437
11.6.4.84 icc_getICCReaderStatus()	438
11.6.4.85 icc_getKeyFormatForICCDUKPT()	438
11.6.4.86 icc_getKeyTypeForICCDUKPT()	439
11.6.4.87 icc_powerOffICC()	439
11.6.4.88 icc_powerOnICC()	439
11.6.4.89 msr_registerCallBk()	440
11.6.4.90 msr_registerCallBkp()	440
11.6.4.91 pin_registerCallBk()	440
11.6.4.92 registerHotplugCallBk()	440
11.6.4.93 registerLogCallBk()	440
11.6.4.94 rs232_device_init()	440
11.6.4.95 SDK_Version()	441
11.6.4.96 setAbsoluteLibraryPath()	441
11.7 Source_C/libIDT_NEO2.h File Reference	442

11.7.1 Detailed Description	447
11.7.2 Macro Definition Documentation	448
11.7.2.1 IN	448
11.7.2.2 IN_OUT	448
11.7.2.3 OUT	448
11.7.3 Typedef Documentation	448
11.7.3.1 ftpComm_callBack	448
11.7.3.2 httpComm_callBack	448
11.7.3.3 pCMR_callBack	448
11.7.3.4 pCSFS_callBack	448
11.7.3.5 pEMV_callBack	449
11.7.3.6 pFW_callBack	449
11.7.3.7 pLCD_callBack	449
11.7.3.8 pMessageHotplug	449
11.7.3.9 pMSR_callBack	449
11.7.3.10 pMSR_callBackp	449
11.7.3.11 pPIN_callBack	449
11.7.3.12 pReadDataLog	449
11.7.3.13 pRKI_callBack	450
11.7.3.14 pSendDataLog	450
11.7.3.15 pWN_callBack	450
11.7.3.16 pWP_callBack	450
11.7.3.17 v4Comm_callBack	450
11.7.4 Function Documentation	450
11.7.4.1 cancelWorldNet()	450
11.7.4.2 cancelWorldPay()	451
11.7.4.3 comm_registerHTTPCallback()	451
11.7.4.4 comm_registerV4Callback()	451
11.7.4.5 config_getModelNumber()	451
11.7.4.6 config_getModelNumber_Len()	452
11.7.4.7 config_getSerialNumber()	452
11.7.4.8 config_getSerialNumber_Len()	452
11.7.4.9 config_setCmdTimeOutDuration()	453
11.7.4.10 config_setConfigByJsonFile()	453
11.7.4.11 ctls_activateTransaction()	453
11.7.4.12 ctls_cancelTransaction()	455
11.7.4.13 ctls_displayOnlineAuthResult()	455
11.7.4.14 ctls_getAllConfigurationGroups()	455
11.7.4.15 ctls_getConfigurationGroup()	456
11.7.4.16 ctls_registerCallBk()	456

11.7.4.17	ctls_registerCallBkp()	456
11.7.4.18	ctls_removeAllApplicationData()	456
11.7.4.19	ctls_removeAllCAPK()	456
11.7.4.20	ctls_removeApplicationData()	457
11.7.4.21	ctls_removeCAPK()	457
11.7.4.22	ctls_removeConfigurationGroup()	457
11.7.4.23	ctls_retrieveAIDList()	458
11.7.4.24	ctls_retrieveApplicationData()	458
11.7.4.25	ctls_retrieveCAPK()	459
11.7.4.26	ctls_retrieveCAPKList()	459
11.7.4.27	ctls_retrieveTerminalData()	460
11.7.4.28	ctls_setApplicationData()	460
11.7.4.29	ctls_setCAPK()	461
11.7.4.30	ctls_setConfigurationGroup()	461
11.7.4.31	ctls_setTerminalData()	462
11.7.4.32	ctls_startTransaction()	462
11.7.4.33	device_activateTransaction()	464
11.7.4.34	device_buzzerOnOff()	465
11.7.4.35	device_cancelTransaction()	465
11.7.4.36	device_cancelTransactionSilent()	465
11.7.4.37	device_close()	466
11.7.4.38	device_configureButtons()	466
11.7.4.39	device_controlUserInterface()	466
11.7.4.40	device_deleteDirectory()	468
11.7.4.41	device_deleteFile()	468
11.7.4.42	device_disableBlueLED()	468
11.7.4.43	device_enableBlueLED()	469
11.7.4.44	device_enableExternalLCDMessages()	469
11.7.4.45	device_enableL100PassThrough()	469
11.7.4.46	device_enableL80PassThrough()	470
11.7.4.47	device_enablePassThrough()	470
11.7.4.48	device_enableRFAntenna()	470
11.7.4.49	device_getAudioVolume()	471
11.7.4.50	device_getButtonConfiguration()	471
11.7.4.51	device_getCameraParameters()	472
11.7.4.52	device_getCurrentDeviceType()	472
11.7.4.53	device_getDeviceMemoryUsageInfo()	472
11.7.4.54	device_getDeviceTreeVersion()	472
11.7.4.55	device_getDRS()	473
11.7.4.56	device_getFirmwareVersion()	473

11.7.4.57 device_getFirmwareVersion_Len()	474
11.7.4.58 device_getIDGStatusCodeString()	474
11.7.4.59 device_getKeyStatus()	476
11.7.4.60 device_getL100PassThroughMode()	477
11.7.4.61 device_getL80PassThroughMode()	477
11.7.4.62 device_getMerchantRecord()	477
11.7.4.63 device_getMerchantRecord_Len()	478
11.7.4.64 device_getNEOAltDevice()	478
11.7.4.65 device_getResponseCodeString()	478
11.7.4.66 device_getSDKWaitTime()	489
11.7.4.67 device_getTamperStatus()	489
11.7.4.68 device_getThreadStackSize()	489
11.7.4.69 device_getTransactionResults()	489
11.7.4.70 device_init()	490
11.7.4.71 device_isAttached()	490
11.7.4.72 device_isConnected()	490
11.7.4.73 device_lcdDisplayClear()	490
11.7.4.74 device_lcdDisplayLine1Message()	491
11.7.4.75 device_lcdDisplayLine2Message()	491
11.7.4.76 device_listDirectory()	491
11.7.4.77 device_loadCertCA()	492
11.7.4.78 device_outputLog()	492
11.7.4.79 device_pingDevice()	493
11.7.4.80 device_playAudio()	493
11.7.4.81 device_pollForToken()	493
11.7.4.82 device_queryFile()	494
11.7.4.83 device_readFileFromSD()	494
11.7.4.84 device_rebootDevice()	495
11.7.4.85 device_registerCameraCallBk()	495
11.7.4.86 device_registerCardStatusFrontSwitchCallBk()	495
11.7.4.87 device_registerFWCallBk()	496
11.7.4.88 device_registerRKICallBk()	496
11.7.4.89 device_rrcConnect()	496
11.7.4.90 device_rrcDisconnect()	496
11.7.4.91 device_rrcDownloadApp()	496
11.7.4.92 device_rrcInstallApp()	497
11.7.4.93 device_rrcRunApp()	497
11.7.4.94 device_rrcUninstallApp()	497
11.7.4.95 device_SendDataCommandNEO()	498
11.7.4.96 device_setAudioVolume()	498

11.7.4.97 device_setBurstMode()	498
11.7.4.98 device_setCameraParameters()	499
11.7.4.99 device_setCancelTransactionMode()	499
11.7.4.100 device_setConfigPath()	500
11.7.4.101 device_setCoreDumpLogFile()	500
11.7.4.102 device_setCurrentDevice()	500
11.7.4.103 device_setMerchantRecord()	501
11.7.4.104 device_setNEO2DevicesConfigs()	501
11.7.4.105 device_setNEOAltDevice()	502
11.7.4.106 device_setNEOGen()	502
11.7.4.107 device_setPollMode()	502
11.7.4.108 device_setRKI_URL()	502
11.7.4.109 device_setSDKWaitTime()	504
11.7.4.110 device_setThreadStackSize()	504
11.7.4.111 device_setTransactionExponent()	504
11.7.4.112 device_startListenNotifications()	504
11.7.4.113 device_startQRCodeScan()	505
11.7.4.114 device_startQRCodeScanWithDisplayWindowInfo()	505
11.7.4.115 device_startRKI()	505
11.7.4.116 device_startTakingPhoto()	506
11.7.4.117 device_startTransaction()	506
11.7.4.118 device_stopAudio()	508
11.7.4.119 device_stopListenNotifications()	508
11.7.4.120 device_stopQRCodeScan()	508
11.7.4.121 device_stopTakingPhoto()	509
11.7.4.122 device_toSDCard()	509
11.7.4.123 device_transferFile()	509
11.7.4.124 device_turnOffYellowLED()	510
11.7.4.125 device_turnOnYellowLED()	510
11.7.4.126 device_updateFirmware()	510
11.7.4.127 emv_activateTransaction()	511
11.7.4.128 emv_allowFallback()	512
11.7.4.129 emv_authenticateTransaction()	512
11.7.4.130 emv_authenticateTransactionWithTimeout()	512
11.7.4.131 emv_cancelTransaction()	513
11.7.4.132 emv_completeTransaction()	513
11.7.4.133 emv_getAutoAuthenticateTransaction()	515
11.7.4.134 emv_getAutoCompleteTransaction()	515
11.7.4.135 emv_getEMVConfigurationCheckValue()	515
11.7.4.136 emv_getEMVKernelCheckValue()	516

11.7.4.137	emv_getEMVKernelVersion()	516
11.7.4.138	emv_getEMVKernelVersion_Len()	516
11.7.4.139	emv_registerCallBk()	517
11.7.4.140	emv_removeAllApplicationData()	517
11.7.4.141	emv_removeAllCAPK()	517
11.7.4.142	emv_removeAllCRL()	517
11.7.4.143	emv_removeApplicationData()	517
11.7.4.144	emv_removeCAPK()	519
11.7.4.145	emv_removeCRL()	519
11.7.4.146	emv_retrieveAIDList()	520
11.7.4.147	emv_retrieveApplicationData()	520
11.7.4.148	emv_retrieveCAPK()	520
11.7.4.149	emv_retrieveCAPKList()	521
11.7.4.150	emv_retrieveCRL()	521
11.7.4.151	emv_retrieveTerminalData()	522
11.7.4.152	emv_retrieveTransactionResult()	522
11.7.4.153	emv_setApplicationData()	523
11.7.4.154	emv_setApplicationDataTLV()	523
11.7.4.155	emv_setAutoAuthenticateTransaction()	523
11.7.4.156	emv_setAutoCompleteTransaction()	524
11.7.4.157	emv_setCAPK()	524
11.7.4.158	emv_setCRL()	525
11.7.4.159	emv_setTerminalData()	525
11.7.4.160	emv_setTerminalMajorConfiguration()	525
11.7.4.161	emv_setTransactionParameters()	526
11.7.4.162	emv_startTransaction()	527
11.7.4.163	executeTransaction()	527
11.7.4.164	executeTransaction_WorldNet()	528
11.7.4.165	felica_authentication()	528
11.7.4.166	felica_cancelCodeEntry()	529
11.7.4.167	felica_getCode()	529
11.7.4.168	felica_poll()	529
11.7.4.169	felica_read()	530
11.7.4.170	felica_readWithMac()	530
11.7.4.171	felica_requestService()	531
11.7.4.172	felica_SendCommand()	531
11.7.4.173	felica_write()	532
11.7.4.174	felica_writeWithMac()	532
11.7.4.175	forwardTransaction()	533
11.7.4.176	forwardTransaction_WorldNet()	533

11.7.4.177cc_exchangeAPDU()	534
11.7.4.178cc_getICCReaderStatus()	534
11.7.4.179cc_powerOffICC()	534
11.7.4.180cc_powerOnICC()	535
11.7.4.181lcd_addButton()	535
11.7.4.182cd_addEthernet()	536
11.7.4.183cd_addExtVideo()	537
11.7.4.184cd_addImage()	538
11.7.4.185cd_addLED()	539
11.7.4.186cd_addText()	541
11.7.4.187cd_addVideo()	543
11.7.4.188cd_clearDisplay()	544
11.7.4.189cd_clearScreenInfo()	545
11.7.4.190cd_cloneScreen()	545
11.7.4.191lcd_createScreen()	545
11.7.4.192cd_destroyScreen()	546
11.7.4.193cd_displayMessage()	546
11.7.4.194cd_getActiveScreen()	546
11.7.4.195cd_getAllObjects()	547
11.7.4.196cd_getAllScreens()	547
11.7.4.197cd_getButtonEvent()	548
11.7.4.198cd_linkUIWithTransactionMessageId()	548
11.7.4.199cd_loadScreenInfo()	549
11.7.4.200cd_queryObjectbyID()	549
11.7.4.201lcd_queryObjectbyName()	549
11.7.4.202cd_queryScreenbyID()	550
11.7.4.203cd_queryScreenbyName()	550
11.7.4.204cd_registerCallBk()	551
11.7.4.205cd_removeItem()	551
11.7.4.206cd_setBacklight()	551
11.7.4.207cd_showScreen()	552
11.7.4.208cd_storeScreenInfo()	552
11.7.4.209cd_updateColor()	552
11.7.4.210cd_updateLabel()	553
11.7.4.211lcd_updatePosition()	554
11.7.4.212loyalty_cancelTransaction()	554
11.7.4.213loyalty_cancelTransactionSilent()	555
11.7.4.214loyalty_registerCallBk()	555
11.7.4.215loyalty_startTransaction()	555
11.7.4.216msr_cancelMSRSwipe()	557

11.7.4.217	msr_registerCallBk()	557
11.7.4.218	msr_registerCallBkp()	557
11.7.4.219	msr_startMSRSwipe()	557
11.7.4.220	parseMSRData()	558
11.7.4.221	pin_cancelPINEntry()	558
11.7.4.222	pin_capturePin()	558
11.7.4.223	pin_capturePinExt()	559
11.7.4.224	pin_getPanEntry()	560
11.7.4.225	pin_inputFromPrompt()	561
11.7.4.226	pin_promptForNumericKey()	562
11.7.4.227	pin_promptForNumericKeyWithSwipe()	562
11.7.4.228	pin_registerCallBk()	563
11.7.4.229	pin_setKeyValues()	563
11.7.4.230	registerHotplugCallBk()	564
11.7.4.231	registerLogCallBk()	564
11.7.4.232	rs232_device_init()	564
11.7.4.233	SDK_Version()	566
11.7.4.234	set_open_com_port_timeout()	566
11.7.4.235	setAbsoluteLibraryPath()	566
11.8	Source_C/libIDT_PipReader.h File Reference	566
11.8.1	Detailed Description	568
11.8.2	Macro Definition Documentation	568
11.8.2.1	IN	568
11.8.2.2	IN_OUT	568
11.8.2.3	OUT	568
11.8.3	Typedef Documentation	569
11.8.3.1	ftpComm_callBack	569
11.8.3.2	httpComm_callBack	569
11.8.3.3	pCMR_callBack	569
11.8.3.4	pCSFS_callBack	569
11.8.3.5	pEMV_callBack	569
11.8.3.6	pMessageHotplug	569
11.8.3.7	pMSR_callBack	569
11.8.3.8	pMSR_callBackp	570
11.8.3.9	pPIN_callBack	570
11.8.3.10	pReadDataLog	570
11.8.3.11	pSendDataLog	570
11.8.3.12	v4Comm_callBack	570
11.8.4	Function Documentation	570
11.8.4.1	config_getSerialNumber()	570

11.8.4.2	config_getSerialNumber_Len()	571
11.8.4.3	ctls_activateTransaction()	571
11.8.4.4	ctls_cancelTransaction()	572
11.8.4.5	ctls_getAllConfigurationGroups()	573
11.8.4.6	ctls_getConfigurationGroup()	573
11.8.4.7	ctls_registerCallBk()	573
11.8.4.8	ctls_registerCallBkp()	573
11.8.4.9	ctls_removeAllApplicationData()	574
11.8.4.10	ctls_removeAllCAPK()	574
11.8.4.11	ctls_removeApplicationData()	574
11.8.4.12	ctls_removeCAPK()	574
11.8.4.13	ctls_removeConfigurationGroup()	575
11.8.4.14	ctls_retrieveAIDList()	575
11.8.4.15	ctls_retrieveApplicationData()	575
11.8.4.16	ctls_retrieveCAPK()	576
11.8.4.17	ctls_retrieveCAPKList()	577
11.8.4.18	ctls_retrieveTerminalData()	577
11.8.4.19	ctls_setApplicationData()	577
11.8.4.20	ctls_setCAPK()	578
11.8.4.21	ctls_setConfigurationGroup()	579
11.8.4.22	ctls_setTerminalData()	579
11.8.4.23	ctls_startTransaction()	579
11.8.4.24	device_close()	581
11.8.4.25	device_controlUserInterface()	581
11.8.4.26	device_enablePassThrough()	583
11.8.4.27	device_getCurrentDeviceType()	583
11.8.4.28	device_getFirmwareVersion()	583
11.8.4.29	device_getFirmwareVersion_Len()	583
11.8.4.30	device_getIDGStatusCodeString()	584
11.8.4.31	device_getMerchantRecord()	585
11.8.4.32	device_getMerchantRecord_Len()	586
11.8.4.33	device_getSDKWaitTime()	586
11.8.4.34	device_getTransactionResults()	587
11.8.4.35	device_init()	587
11.8.4.36	device_isAttached()	587
11.8.4.37	device_isConnected()	588
11.8.4.38	device_pingDevice()	588
11.8.4.39	device_registerCameraCallBk()	588
11.8.4.40	device_registerCardStatusFrontSwitchCallBk()	588
11.8.4.41	device_SendDataCommandNEO()	588

11.8.4.42 device_setBurstMode()	589
11.8.4.43 device_setCurrentDevice()	589
11.8.4.44 device_setMerchantRecord()	590
11.8.4.45 device_setPollMode()	590
11.8.4.46 device_setSDKWaitTime()	590
11.8.4.47 emv_registerCallBk()	591
11.8.4.48 parseMSRData()	591
11.8.4.49 pin_registerCallBk()	591
11.8.4.50 registerHotplugCallBk()	591
11.8.4.51 registerLogCallBk()	591
11.8.4.52 rs232_device_init()	592
11.8.4.53 SDK_Version()	592
11.8.4.54 setAbsoluteLibraryPath()	592
11.9 Source_C/libIDT_SpectrumPro.h File Reference	593
11.9.1 Detailed Description	595
11.9.2 Macro Definition Documentation	595
11.9.2.1 IN	595
11.9.2.2 IN_OUT	595
11.9.2.3 OUT	595
11.9.3 Typedef Documentation	595
11.9.3.1 ftpComm_callBack	596
11.9.3.2 httpComm_callBack	596
11.9.3.3 pCMR_callBack	596
11.9.3.4 pCSFS_callBack	596
11.9.3.5 pEMV_callBack	596
11.9.3.6 pMessageHotplug	596
11.9.3.7 pMSR_callBack	596
11.9.3.8 pMSR_callBackp	596
11.9.3.9 pPIN_callBack	597
11.9.3.10 pReadDataLog	597
11.9.3.11 pSendDataLog	597
11.9.3.12 v4Comm_callBack	597
11.9.4 Function Documentation	597
11.9.4.1 config_getModelNumber()	597
11.9.4.2 config_getModelNumber_Len()	598
11.9.4.3 config_getSerialNumber()	598
11.9.4.4 config_getSerialNumber_Len()	598
11.9.4.5 device_close()	599
11.9.4.6 device_getCurrentDeviceType()	599
11.9.4.7 device_getFirmwareVersion()	599

11.9.4.8	<code>device_getFirmwareVersion_Len()</code>	599
11.9.4.9	<code>device_getResponseCodeString()</code>	600
11.9.4.10	<code>device_getSDKWaitTime()</code>	610
11.9.4.11	<code>device_getSpectrumProKSN()</code>	610
11.9.4.12	<code>device_getSpectrumProKSN_Len()</code>	610
11.9.4.13	<code>device_getThreadStackSize()</code>	611
11.9.4.14	<code>device_init()</code>	611
11.9.4.15	<code>device_isAttached()</code>	612
11.9.4.16	<code>device_isConnected()</code>	612
11.9.4.17	<code>device_pollCardReader()</code>	612
11.9.4.18	<code>device_pollCardReader_Len()</code>	614
11.9.4.19	<code>device_rebootDevice()</code>	616
11.9.4.20	<code>device_registerCameraCallBk()</code>	616
11.9.4.21	<code>device_registerCardStatusFrontSwitchCallBk()</code>	616
11.9.4.22	<code>device_SendDataCommand()</code>	616
11.9.4.23	<code>device_setCurrentDevice()</code>	617
11.9.4.24	<code>device_setSDKWaitTime()</code>	617
11.9.4.25	<code>device_setThreadStackSize()</code>	617
11.9.4.26	<code>device_updateFirmware()</code>	618
11.9.4.27	<code>emv_activateTransaction()</code>	618
11.9.4.28	<code>emv_allowFallback()</code>	619
11.9.4.29	<code>emv_authenticateTransaction()</code>	619
11.9.4.30	<code>emv_authenticateTransactionWithTimeout()</code>	620
11.9.4.31	<code>emv_callbackResponseLCD()</code>	621
11.9.4.32	<code>emv_callbackResponseMSR()</code>	621
11.9.4.33	<code>emv_cancelTransaction()</code>	621
11.9.4.34	<code>emv_completeTransaction()</code>	622
11.9.4.35	<code>emv_getAutoAuthenticateTransaction()</code>	622
11.9.4.36	<code>emv_getAutoCompleteTransaction()</code>	623
11.9.4.37	<code>emv_getEMVConfigurationCheckValue()</code>	623
11.9.4.38	<code>emv_getEMVKernelCheckValue()</code>	623
11.9.4.39	<code>emv_getEMVKernelVersion()</code>	623
11.9.4.40	<code>emv_getEMVKernelVersion_Len()</code>	624
11.9.4.41	<code>emv_registerCallBk()</code>	624
11.9.4.42	<code>emv_removeAllApplicationData()</code>	624
11.9.4.43	<code>emv_removeAllCAPK()</code>	624
11.9.4.44	<code>emv_removeAllCRL()</code>	625
11.9.4.45	<code>emv_removeApplicationData()</code>	625
11.9.4.46	<code>emv_removeCAPK()</code>	625
11.9.4.47	<code>emv_removeCRL()</code>	626

11.9.4.48 emv_removeTerminalData()	626
11.9.4.49 emv_retrieveAIDList()	626
11.9.4.50 emv_retrieveApplicationData()	627
11.9.4.51 emv_retrieveCAPK()	627
11.9.4.52 emv_retrieveCAPKList()	628
11.9.4.53 emv_retrieveCRL()	628
11.9.4.54 emv_retrieveTerminalData()	629
11.9.4.55 emv_retrieveTerminalID()	629
11.9.4.56 emv_retrieveTerminalID_Len()	630
11.9.4.57 emv_retrieveTransactionResult()	630
11.9.4.58 emv_setApplicationData()	630
11.9.4.59 emv_setAutoAuthenticateTransaction()	631
11.9.4.60 emv_setAutoCompleteTransaction()	631
11.9.4.61 emv_setCAPK()	631
11.9.4.62 emv_setCRL()	632
11.9.4.63 emv_setTerminalData()	632
11.9.4.64 emv_setTerminalID()	634
11.9.4.65 emv_startTransaction()	634
11.9.4.66 icc_getICCReaderStatus()	635
11.9.4.67 icc_powerOffICC()	635
11.9.4.68 icc_powerOnICC()	635
11.9.4.69 msr_cancelMSRSwipe()	636
11.9.4.70 msr_clearMSRData()	636
11.9.4.71 msr_getMSRData()	636
11.9.4.72 msr_registerCallBk()	637
11.9.4.73 msr_registerCallBkp()	637
11.9.4.74 msr_startMSRSwipe()	637
11.9.4.75 parseMSRData()	637
11.9.4.76 parsePINBlockData()	637
11.9.4.77 parsePINData()	638
11.9.4.78 pin_cancelPINEntry()	638
11.9.4.79 pin_getPIN()	638
11.9.4.80 pin_registerCallBk()	639
11.9.4.81 registerHotplugCallBk()	639
11.9.4.82 registerLogCallBk()	639
11.9.4.83 rs232_device_init()	640
11.9.4.84 SDK_Version()	640
11.9.4.85 setAbsoluteLibraryPath()	640
11.10Source_C/libIDT_SREDKey2.h File Reference	641
11.10.1 Detailed Description	642

11.10.2 Macro Definition Documentation	642
11.10.2.1 IN	643
11.10.2.2 IN_OUT	643
11.10.2.3 OUT	643
11.10.3 Typedef Documentation	643
11.10.3.1 ftpComm_callBack	643
11.10.3.2 httpComm_callBack	643
11.10.3.3 pCMR_callBack	643
11.10.3.4 pCSFS_callBack	643
11.10.3.5 pEMV_callBack	643
11.10.3.6 pFW_callBack	644
11.10.3.7 pLCD_callBack	644
11.10.3.8 pMessageHotplug	644
11.10.3.9 pMSR_callBack	644
11.10.3.10pMSR_callBackp	644
11.10.3.11pPIN_callBack	644
11.10.3.12pReadDataLog	644
11.10.3.13pSendDataLog	645
11.10.3.14v4Comm_callBack	645
11.10.4 Function Documentation	645
11.10.4.1 comm_registerHTTPCallback()	645
11.10.4.2 comm_registerV4Callback()	645
11.10.4.3 config_getModelNumber()	645
11.10.4.4 config_getModelNumber_Len()	646
11.10.4.5 config_getSerialNumber()	646
11.10.4.6 config_getSerialNumber_Len()	646
11.10.4.7 ctls_registerCallBk()	647
11.10.4.8 ctls_registerCallBkp()	647
11.10.4.9 device_close()	647
11.10.4.10device_getCurrentDeviceType()	647
11.10.4.11device_getFirmwareVersion()	647
11.10.4.12device_getFirmwareVersion_Len()	648
11.10.4.13device_getIDGStatusCodeString()	648
11.10.4.14device_getKeyStatus()	650
11.10.4.15device_init()	650
11.10.4.16device_isAttached()	651
11.10.4.17device_isConnected()	651
11.10.4.18device_pingDevice()	651
11.10.4.19device_rebootDevice()	651
11.10.4.20device_registerCameraCallBk()	652

11.10.4.21	device_registerCardStatusFrontSwitchCallBk()	652
11.10.4.22	device_registerFWCallBk()	652
11.10.4.23	device_SendDataCommand()	652
11.10.4.24	device_SendDataCommandITP()	653
11.10.4.25	device_SendDataCommandNEO()	653
11.10.4.26	device_setConfigPath()	654
11.10.4.27	device_setCurrentDevice()	654
11.10.4.28	device_setNEO2DevicesConfigs()	654
11.10.4.29	device_setSystemLanguage()	655
11.10.4.30	device_setTransactionExponent()	655
11.10.4.31	device_updateFirmware()	655
11.10.4.32	mv_registerCallBk()	656
11.10.4.33	cd_registerCallBk()	656
11.10.4.34	msr_disable()	656
11.10.4.35	msr_getClearPANID()	657
11.10.4.36	msr_getExpirationMask()	657
11.10.4.37	msr_getFunctionStatus()	657
11.10.4.38	msr_getSwipeForcedEncryptionOption()	658
11.10.4.39	msr_getSwipeMaskOption()	658
11.10.4.40	msr_registerCallBk()	658
11.10.4.41	msr_registerCallBkp()	658
11.10.4.42	msr_setClearPANID()	659
11.10.4.43	msr_setExpirationMask()	659
11.10.4.44	msr_setSwipeForcedEncryptionOption()	659
11.10.4.45	msr_setSwipeMaskOption()	660
11.10.4.46	pin_registerCallBk()	660
11.10.4.47	registerHotplugCallBk()	660
11.10.4.48	registerLogCallBk()	660
11.10.4.49	s232_device_init()	660
11.10.4.50	SDK_Version()	661
11.10.4.51	setAbsoluteLibraryPath()	661
11.11	Source_C/libIDT_UniPayI_V.h File Reference	662
11.11.1	Detailed Description	664
11.11.2	Macro Definition Documentation	664
11.11.2.1	IN	664
11.11.2.2	IN_OUT	664
11.11.2.3	OUT	664
11.11.3	Typedef Documentation	664
11.11.3.1	ftpComm_callBack	664
11.11.3.2	httpComm_callBack	664

11.11.3.3 pCMR_callback	664
11.11.3.4 pCSFS_callback	665
11.11.3.5 pEMV_callback	665
11.11.3.6 pMessageHotplug	665
11.11.3.7 pMSR_callback	665
11.11.3.8 pMSR_callbackp	665
11.11.3.9 pPIN_callback	665
11.11.3.10pReadDataLog	665
11.11.3.11pSendDataLog	665
11.11.3.12/4Comm_callback	666
11.11.4 Function Documentation	666
11.11.4.1 comm_registerHTTPCallback()	666
11.11.4.2 comm_registerV4Callback()	666
11.11.4.3 config_getSerialNumber()	666
11.11.4.4 config_getSerialNumber_Len()	667
11.11.4.5 device_close()	667
11.11.4.6 device_enablePassThrough()	667
11.11.4.7 device_getCurrentDeviceType()	668
11.11.4.8 device_getFirmwareVersion()	668
11.11.4.9 device_getFirmwareVersion_Len()	668
11.11.4.10device_getIDGStatusCodeString()	668
11.11.4.11device_getMerchantRecord()	670
11.11.4.12device_getMerchantRecord_Len()	671
11.11.4.13device_getSDKWaitTime()	671
11.11.4.14device_getThreadStackSize()	671
11.11.4.15device_init()	672
11.11.4.16device_isAttached()	672
11.11.4.17device_isConnected()	672
11.11.4.18device_pingDevice()	672
11.11.4.19device_registerCameraCallbk()	673
11.11.4.20device_registerCardStatusFrontSwitchCallbk()	673
11.11.4.21device_SendDataCommandNEO()	673
11.11.4.22device_setCurrentDevice()	674
11.11.4.23device_setMerchantRecord()	674
11.11.4.24device_setSDKWaitTime()	675
11.11.4.25device_setThreadStackSize()	675
11.11.4.26emv_activateTransaction()	675
11.11.4.27emv_allowFallback()	676
11.11.4.28emv_authenticateTransaction()	676
11.11.4.29emv_authenticateTransactionWithTimeout()	676

11.11.4.30	emv_cancelTransaction()	677
11.11.4.31	emv_completeTransaction()	677
11.11.4.32	emv_getAutoAuthenticateTransaction()	678
11.11.4.33	emv_getAutoCompleteTransaction()	678
11.11.4.34	emv_registerCallBk()	678
11.11.4.35	emv_removeAllApplicationData()	678
11.11.4.36	emv_removeAllCAPK()	679
11.11.4.37	emv_removeAllCRL()	679
11.11.4.38	emv_removeApplicationData()	679
11.11.4.39	emv_removeCAPK()	679
11.11.4.40	emv_removeCRL()	680
11.11.4.41	emv_retrieveAIDList()	680
11.11.4.42	emv_retrieveApplicationData()	681
11.11.4.43	emv_retrieveCAPK()	681
11.11.4.44	emv_retrieveCAPKList()	682
11.11.4.45	emv_retrieveCRL()	682
11.11.4.46	emv_retrieveTerminalData()	683
11.11.4.47	emv_setApplicationData()	683
11.11.4.48	emv_setApplicationDataTLV()	684
11.11.4.49	emv_setAutoAuthenticateTransaction()	684
11.11.4.50	emv_setAutoCompleteTransaction()	684
11.11.4.51	emv_setCAPK()	685
11.11.4.52	emv_setCRL()	685
11.11.4.53	emv_setTerminalData()	686
11.11.4.54	emv_setTerminalMajorConfiguration()	686
11.11.4.55	emv_startTransaction()	686
11.11.4.56	cc_exchangeAPDU()	687
11.11.4.57	cc_getICCReaderStatus()	688
11.11.4.58	cc_powerOffICC()	688
11.11.4.59	cc_powerOnICC()	688
11.11.4.60	msr_cancelMSRSwipe()	689
11.11.4.61	msr_registerCallBk()	689
11.11.4.62	msr_registerCallBkp()	689
11.11.4.63	msr_startMSRSwipe()	689
11.11.4.64	parseMSRData()	689
11.11.4.65	pin_registerCallBk()	690
11.11.4.66	registerHotplugCallBk()	690
11.11.4.67	registerLogCallBk()	690
11.11.4.68	SDK_Version()	690
11.11.4.69	setAbsoluteLibraryPath()	690

11.12Source_C/libIDT_Vendi.h File Reference	691
11.12.1 Detailed Description	692
11.12.2 Macro Definition Documentation	692
11.12.2.1 IN	693
11.12.2.2 IN_OUT	693
11.12.2.3 OUT	693
11.12.3 Typedef Documentation	693
11.12.3.1 ftpComm_callBack	693
11.12.3.2 httpComm_callBack	693
11.12.3.3 pCMR_callBack	693
11.12.3.4 pCSFS_callBack	693
11.12.3.5 pEMV_callBack	693
11.12.3.6 pMessageHotplug	694
11.12.3.7 pMSR_callBack	694
11.12.3.8 pMSR_callBackp	694
11.12.3.9 pPIN_callBack	694
11.12.3.10pReadDataLog	694
11.12.3.11pSendDataLog	694
11.12.3.12v4Comm_callBack	694
11.12.4 Function Documentation	695
11.12.4.1 comm_registerHTTPCallback()	695
11.12.4.2 comm_registerV4Callback()	695
11.12.4.3 config_getSerialNumber()	695
11.12.4.4 config_getSerialNumber_Len()	695
11.12.4.5 ctls_activateTransaction()	696
11.12.4.6 ctls_cancelTransaction()	697
11.12.4.7 ctls_getAllConfigurationGroups()	697
11.12.4.8 ctls_getConfigurationGroup()	698
11.12.4.9 ctls_registerCallBk()	698
11.12.4.10ctls_registerCallBkp()	698
11.12.4.11ctls_removeAllApplicationData()	698
11.12.4.12ctls_removeAllCAPK()	699
11.12.4.13ctls_removeApplicationData()	699
11.12.4.14ctls_removeCAPK()	699
11.12.4.15ctls_removeConfigurationGroup()	700
11.12.4.16ctls_retrieveAIDList()	700
11.12.4.17ctls_retrieveApplicationData()	700
11.12.4.18ctls_retrieveCAPK()	701
11.12.4.19ctls_retrieveCAPKList()	701
11.12.4.20ctls_retrieveTerminalData()	702

11.12.4.21	ctls_setApplicationData()	702
11.12.4.22	ctls_setCAPK()	703
11.12.4.23	ctls_setConfigurationGroup()	703
11.12.4.24	ctls_setTerminalData()	704
11.12.4.25	ctls_startTransaction()	704
11.12.4.26	device_close()	706
11.12.4.27	device_controlUserInterface()	706
11.12.4.28	device_enablePassThrough()	708
11.12.4.29	device_getCurrentDeviceType()	708
11.12.4.30	device_getFirmwareVersion()	708
11.12.4.31	device_getFirmwareVersion_Len()	708
11.12.4.32	device_getIDGStatusCodeString()	709
11.12.4.33	device_getMerchantRecord()	710
11.12.4.34	device_getMerchantRecord_Len()	711
11.12.4.35	device_getSDKWaitTime()	711
11.12.4.36	device_getThreadStackSize()	712
11.12.4.37	device_getTransactionResults()	712
11.12.4.38	device_init()	712
11.12.4.39	device_isAttached()	712
11.12.4.40	device_isConnected()	713
11.12.4.41	device_pingDevice()	713
11.12.4.42	device_registerCameraCallBk()	713
11.12.4.43	device_registerCardStatusFrontSwitchCallBk()	713
11.12.4.44	device_SendDataCommandNEO()	713
11.12.4.45	device_setBurstMode()	714
11.12.4.46	device_setCurrentDevice()	715
11.12.4.47	device_setMerchantRecord()	715
11.12.4.48	device_setPollMode()	716
11.12.4.49	device_setSDKWaitTime()	716
11.12.4.50	device_setThreadStackSize()	716
11.12.4.51	emv_registerCallBk()	716
11.12.4.52	msr_cancelMSRSwipe()	716
11.12.4.53	msr_registerCallBk()	717
11.12.4.54	msr_registerCallBkp()	717
11.12.4.55	msr_startMSRSwipe()	717
11.12.4.56	parseMSRData()	717
11.12.4.57	pin_registerCallBk()	718
11.12.4.58	registerHotplugCallBk()	718
11.12.4.59	registerLogCallBk()	718
11.12.4.60	SDK_Version()	718

11.12.4.6	setAbsoluteLibraryPath()	718
11.13	Source_C/libIDT_VP3300_AJ.h File Reference	718
11.13.1	Detailed Description	721
11.13.2	Macro Definition Documentation	721
11.13.2.1	IN	722
11.13.2.2	IN_OUT	722
11.13.2.3	OUT	722
11.13.3	Typedef Documentation	722
11.13.3.1	ftpComm_callBack	722
11.13.3.2	httpComm_callBack	722
11.13.3.3	pCMR_callBack	722
11.13.3.4	pCSFS_callBack	722
11.13.3.5	pEMV_callBack	722
11.13.3.6	pMessageHotplug	723
11.13.3.7	pMSR_callBack	723
11.13.3.8	pMSR_callBackp	723
11.13.3.9	pPIN_callBack	723
11.13.3.10	pReadDataLog	723
11.13.3.11	pRKI_callBack	723
11.13.3.12	pSendDataLog	723
11.13.3.13	pWN_callBack	724
11.13.3.14	pWP_callBack	724
11.13.3.15	v4Comm_callBack	724
11.13.4	Function Documentation	724
11.13.4.1	cancelWorldNet()	724
11.13.4.2	cancelWorldPay()	724
11.13.4.3	comm_registerHTTPCallback()	724
11.13.4.4	comm_registerV4Callback()	725
11.13.4.5	config_getSerialNumber()	725
11.13.4.6	config_getSerialNumber_Len()	725
11.13.4.7	ctls_activateTransaction()	726
11.13.4.8	ctls_cancelTransaction()	727
11.13.4.9	ctls_getAllConfigurationGroups()	727
11.13.4.10	ctls_getConfigurationGroup()	728
11.13.4.11	ctls_registerCallBk()	728
11.13.4.12	ctls_registerCallBkp()	728
11.13.4.13	ctls_removeAllApplicationData()	728
11.13.4.14	ctls_removeAllCAPK()	728
11.13.4.15	ctls_removeApplicationData()	729
11.13.4.16	ctls_removeCAPK()	729

11.13.4.17	tls_removeConfigurationGroup()	729
11.13.4.18	tls_retrieveAIDList()	730
11.13.4.19	tls_retrieveApplicationData()	730
11.13.4.20	tls_retrieveCAPK()	731
11.13.4.21	tls_retrieveCAPKList()	731
11.13.4.22	tls_retrieveTerminalData()	732
11.13.4.23	tls_setApplicationData()	732
11.13.4.24	tls_setCAPK()	733
11.13.4.25	tls_setConfigurationGroup()	733
11.13.4.26	tls_setTerminalData()	734
11.13.4.27	tls_startTransaction()	734
11.13.4.28	device_activateTransaction()	736
11.13.4.29	device_cancelTransaction()	737
11.13.4.30	device_close()	737
11.13.4.31	device_controlUserInterface()	737
11.13.4.32	device_enablePassThrough()	739
11.13.4.33	device_getCurrentDeviceType()	739
11.13.4.34	device_getFirmwareVersion()	739
11.13.4.35	device_getFirmwareVersion_Len()	739
11.13.4.36	device_getIDGStatusCodeString()	740
11.13.4.37	device_getMerchantRecord()	741
11.13.4.38	device_getMerchantRecord_Len()	742
11.13.4.39	device_getRTCDateTime()	742
11.13.4.40	device_getSDKWaitTime()	743
11.13.4.41	device_getThreadStackSize()	743
11.13.4.42	device_getTransactionResults()	743
11.13.4.43	device_init()	744
11.13.4.44	device_isAttached()	744
11.13.4.45	device_isConnected()	744
11.13.4.46	device_pingDevice()	744
11.13.4.47	device_pollForToken()	745
11.13.4.48	device_registerCameraCallBk()	745
11.13.4.49	device_registerCardStatusFrontSwitchCallBk()	745
11.13.4.50	device_registerRKICallBk()	745
11.13.4.51	device_SendDataCommandNEO()	745
11.13.4.52	device_setBurstMode()	746
11.13.4.53	device_setCurrentDevice()	747
11.13.4.54	device_setMerchantRecord()	747
11.13.4.55	device_setPollMode()	747
11.13.4.56	device_setRKI_URL()	748

11.13.4.57device_setRTCDateTime()	748
11.13.4.58device_setSDKWaitTime()	749
11.13.4.59device_setThreadStackSize()	749
11.13.4.60device_setTransactionExponent()	749
11.13.4.61device_startRKL()	749
11.13.4.62device_startTransaction()	750
11.13.4.63emv_activateTransaction()	751
11.13.4.64emv_allowFallback()	752
11.13.4.65emv_authenticateTransaction()	752
11.13.4.66emv_authenticateTransactionWithTimeout()	752
11.13.4.67emv_cancelTransaction()	753
11.13.4.68emv_completeTransaction()	753
11.13.4.69emv_getAutoAuthenticateTransaction()	754
11.13.4.70emv_getAutoCompleteTransaction()	754
11.13.4.71emv_registerCallBk()	754
11.13.4.72emv_removeAllApplicationData()	755
11.13.4.73emv_removeAllCAPK()	755
11.13.4.74emv_removeAllCRL()	755
11.13.4.75emv_removeApplicationData()	755
11.13.4.76emv_removeCAPK()	756
11.13.4.77emv_removeCRL()	756
11.13.4.78emv_retrieveAIDList()	756
11.13.4.79emv_retrieveApplicationData()	757
11.13.4.80emv_retrieveCAPK()	757
11.13.4.81emv_retrieveCAPKList()	758
11.13.4.82emv_retrieveCRL()	758
11.13.4.83emv_retrieveTerminalData()	759
11.13.4.84emv_setApplicationData()	759
11.13.4.85emv_setApplicationDataTLV()	760
11.13.4.86emv_setAutoAuthenticateTransaction()	760
11.13.4.87emv_setAutoCompleteTransaction()	760
11.13.4.88emv_setCAPK()	761
11.13.4.89emv_setCRL()	761
11.13.4.90emv_setTerminalData()	762
11.13.4.91emv_setTerminalMajorConfiguration()	762
11.13.4.92emv_setTransactionParameters()	762
11.13.4.93emv_startTransaction()	763
11.13.4.94executeTransaction()	764
11.13.4.95executeTransaction_WorldNet()	764
11.13.4.96forwardTransaction()	765

11.13.4.97forwardTransaction_WorldNet()	765
11.13.4.98cc_exchangeAPDU()	766
11.13.4.99cc_getICCReaderStatus()	766
11.13.4.100c_powerOffICC()	768
11.13.4.101cc_powerOnICC()	768
11.13.4.102sr_cancelMSRSwipe()	768
11.13.4.103sr_registerCallBk()	769
11.13.4.104sr_registerCallBkp()	769
11.13.4.105sr_startMSRSwipe()	769
11.13.4.106parseMSRData()	769
11.13.4.107in_registerCallBk()	769
11.13.4.108registerHotplugCallBk()	770
11.13.4.109registerLogCallBk()	770
11.13.4.110SDK_Version()	770
11.13.4.111setAbsoluteLibraryPath()	770
11.14Source_C/libIDT_VP3300_BT.h File Reference	770
11.14.1 Detailed Description	773
11.14.2 Macro Definition Documentation	773
11.14.2.1 IN	773
11.14.2.2 IN_OUT	773
11.14.2.3 OUT	774
11.14.3 Typedef Documentation	774
11.14.3.1 ftpComm_callBack	774
11.14.3.2 httpComm_callBack	774
11.14.3.3 pCMR_callBack	774
11.14.3.4 pCSFS_callBack	774
11.14.3.5 pEMV_callBack	774
11.14.3.6 pMessageHotplug	774
11.14.3.7 pMSR_callBack	775
11.14.3.8 pMSR_callBackp	775
11.14.3.9 pPIN_callBack	775
11.14.3.10pReadDataLog	775
11.14.3.11pRKI_callBack	775
11.14.3.12pSendDataLog	775
11.14.3.13pWN_callBack	775
11.14.3.14pWP_callBack	775
11.14.3.15v4Comm_callBack	776
11.14.4 Function Documentation	776
11.14.4.1 cancelWorldNet()	776
11.14.4.2 cancelWorldPay()	776

11.14.4.3 comm_registerHTTPCallback()	776
11.14.4.4 comm_registerV4Callback()	776
11.14.4.5 config_getSerialNumber()	777
11.14.4.6 config_getSerialNumber_Len()	777
11.14.4.7 ctls_activateTransaction()	777
11.14.4.8 ctls_cancelTransaction()	779
11.14.4.9 ctls_getAllConfigurationGroups()	779
11.14.4.10ctls_getConfigurationGroup()	779
11.14.4.11ctls_registerCallBk()	780
11.14.4.12ctls_registerCallBkp()	780
11.14.4.13ctls_removeAllApplicationData()	780
11.14.4.14ctls_removeAllCAPK()	780
11.14.4.15ctls_removeApplicationData()	780
11.14.4.16ctls_removeCAPK()	781
11.14.4.17ctls_removeConfigurationGroup()	781
11.14.4.18ctls_retrieveAIDList()	781
11.14.4.19ctls_retrieveApplicationData()	782
11.14.4.20ctls_retrieveCAPK()	782
11.14.4.21ctls_retrieveCAPKList()	783
11.14.4.22ctls_retrieveTerminalData()	783
11.14.4.23ctls_setApplicationData()	784
11.14.4.24ctls_setCAPK()	784
11.14.4.25ctls_setConfigurationGroup()	785
11.14.4.26ctls_setTerminalData()	785
11.14.4.27ctls_startTransaction()	786
11.14.4.28device_activateTransaction()	787
11.14.4.29device_cancelTransaction()	789
11.14.4.30device_close()	789
11.14.4.31device_controlUserInterface()	789
11.14.4.32device_enablePassThrough()	791
11.14.4.33device_getCurrentDeviceType()	791
11.14.4.34device_getFirmwareVersion()	791
11.14.4.35device_getFirmwareVersion_Len()	791
11.14.4.36device_getIDGStatusCodeString()	792
11.14.4.37device_getMerchantRecord()	793
11.14.4.38device_getMerchantRecord_Len()	794
11.14.4.39device_getRTCDateTime()	794
11.14.4.40device_getSDKWaitTime()	795
11.14.4.41device_getThreadStackSize()	795
11.14.4.42device_getTransactionResults()	795

11.14.4.43	device_init()	796
11.14.4.44	device_isAttached()	796
11.14.4.45	device_isConnected()	796
11.14.4.46	device_pingDevice()	796
11.14.4.47	device_pollForToken()	797
11.14.4.48	device_registerCameraCallBk()	797
11.14.4.49	device_registerCardStatusFrontSwitchCallBk()	797
11.14.4.50	device_registerRKICallBk()	797
11.14.4.51	device_SendDataCommandNEO()	797
11.14.4.52	device_setBurstMode()	798
11.14.4.53	device_setCurrentDevice()	799
11.14.4.54	device_setMerchantRecord()	799
11.14.4.55	device_setPollMode()	799
11.14.4.56	device_setRKI_URL()	800
11.14.4.57	device_setRTCDateTime()	800
11.14.4.58	device_setSDKWaitTime()	801
11.14.4.59	device_setThreadStackSize()	801
11.14.4.60	device_setTransactionExponent()	801
11.14.4.61	device_startRKI()	801
11.14.4.62	device_startTransaction()	802
11.14.4.63	emv_activateTransaction()	803
11.14.4.64	emv_allowFallback()	804
11.14.4.65	emv_authenticateTransaction()	804
11.14.4.66	emv_authenticateTransactionWithTimeout()	804
11.14.4.67	emv_cancelTransaction()	805
11.14.4.68	emv_completeTransaction()	805
11.14.4.69	emv_getAutoAuthenticateTransaction()	806
11.14.4.70	emv_getAutoCompleteTransaction()	806
11.14.4.71	emv_registerCallBk()	806
11.14.4.72	emv_removeAllApplicationData()	807
11.14.4.73	emv_removeAllCAPK()	807
11.14.4.74	emv_removeAllCRL()	807
11.14.4.75	emv_removeApplicationData()	807
11.14.4.76	emv_removeCAPK()	808
11.14.4.77	emv_removeCRL()	808
11.14.4.78	emv_retrieveAIDList()	808
11.14.4.79	emv_retrieveApplicationData()	809
11.14.4.80	emv_retrieveCAPK()	809
11.14.4.81	emv_retrieveCAPKList()	810
11.14.4.82	emv_retrieveCRL()	810

11.14.4.83	emv_retrieveTerminalData()	811
11.14.4.84	emv_setApplicationData()	811
11.14.4.85	emv_setApplicationDataTLV()	812
11.14.4.86	emv_setAutoAuthenticateTransaction()	812
11.14.4.87	emv_setAutoCompleteTransaction()	812
11.14.4.88	emv_setCAPK()	813
11.14.4.89	emv_setCRL()	813
11.14.4.90	emv_setTerminalData()	814
11.14.4.91	emv_setTerminalMajorConfiguration()	814
11.14.4.92	emv_setTransactionParameters()	814
11.14.4.93	emv_startTransaction()	815
11.14.4.94	executeTransaction()	816
11.14.4.95	executeTransaction_WorldNet()	816
11.14.4.96	forwardTransaction()	817
11.14.4.97	forwardTransaction_WorldNet()	817
11.14.4.98	icc_exchangeAPDU()	818
11.14.4.99	icc_getICCRReaderStatus()	818
11.14.4.100	icc_powerOffICC()	820
11.14.4.101	icc_powerOnICC()	820
11.14.4.102	msr_cancelMSRSwipe()	820
11.14.4.103	msr_registerCallBk()	821
11.14.4.104	msr_registerCallBkp()	821
11.14.4.105	msr_startMSRSwipe()	821
11.14.4.106	parseMSRData()	821
11.14.4.107	pin_registerCallBk()	821
11.14.4.108	registerHotplugCallBk()	822
11.14.4.109	registerLogCallBk()	822
11.14.4.110	SDK_Version()	822
11.14.4.111	setAbsoluteLibraryPath()	822
11.15	Source_C/libIDT_VP3300_COM.h File Reference	822
11.15.1	Detailed Description	825
11.15.2	Macro Definition Documentation	825
11.15.2.1	IN	825
11.15.2.2	IN_OUT	825
11.15.2.3	OUT	826
11.15.3	Typedef Documentation	826
11.15.3.1	ftpComm_callBack	826
11.15.3.2	httpComm_callBack	826
11.15.3.3	pCMR_callBack	826
11.15.3.4	pCSFS_callBack	826

11.15.3.5 pEMV_callback	826
11.15.3.6 pMessageHotplug	826
11.15.3.7 pMSR_callback	827
11.15.3.8 pMSR_callbackp	827
11.15.3.9 pPIN_callback	827
11.15.3.10pReadDataLog	827
11.15.3.11pRKI_callback	827
11.15.3.12pSendDataLog	827
11.15.3.13pWN_callback	827
11.15.3.14pWP_callback	827
11.15.3.15v4Comm_callback	828
11.15.4 Function Documentation	828
11.15.4.1 cancelWorldNet()	828
11.15.4.2 cancelWorldPay()	828
11.15.4.3 comm_registerHTTPCallback()	828
11.15.4.4 comm_registerV4Callback()	828
11.15.4.5 config_getSerialNumber()	829
11.15.4.6 config_getSerialNumber_Len()	829
11.15.4.7 ctls_activateTransaction()	829
11.15.4.8 ctls_cancelTransaction()	831
11.15.4.9 ctls_getAllConfigurationGroups()	831
11.15.4.10ctls_getConfigurationGroup()	831
11.15.4.11ctls_registerCallBk()	832
11.15.4.12ctls_registerCallBkp()	832
11.15.4.13ctls_removeAllApplicationData()	832
11.15.4.14ctls_removeAllCAPK()	832
11.15.4.15ctls_removeApplicationData()	832
11.15.4.16ctls_removeCAPK()	833
11.15.4.17ctls_removeConfigurationGroup()	833
11.15.4.18ctls_retrieveAIDList()	833
11.15.4.19ctls_retrieveApplicationData()	834
11.15.4.20ctls_retrieveCAPK()	834
11.15.4.21ctls_retrieveCAPKList()	835
11.15.4.22ctls_retrieveTerminalData()	835
11.15.4.23ctls_setApplicationData()	836
11.15.4.24ctls_setCAPK()	836
11.15.4.25ctls_setConfigurationGroup()	837
11.15.4.26ctls_setTerminalData()	837
11.15.4.27ctls_startTransaction()	838
11.15.4.28device_activateTransaction()	839

11.15.4.29	device_cancelTransaction()	841
11.15.4.30	device_close()	841
11.15.4.31	device_controlUserInterface()	841
11.15.4.32	device_enablePassThrough()	843
11.15.4.33	device_getCurrentDeviceType()	843
11.15.4.34	device_getFirmwareVersion()	843
11.15.4.35	device_getFirmwareVersion_Len()	843
11.15.4.36	device_getIDGStatusCodeString()	844
11.15.4.37	device_getMerchantRecord()	845
11.15.4.38	device_getMerchantRecord_Len()	846
11.15.4.39	device_getRTCDateTime()	846
11.15.4.40	device_getSDKWaitTime()	847
11.15.4.41	device_getThreadStackSize()	847
11.15.4.42	device_getTransactionResults()	847
11.15.4.43	device_init()	848
11.15.4.44	device_isAttached()	848
11.15.4.45	device_isConnected()	848
11.15.4.46	device_pingDevice()	848
11.15.4.47	device_pollForToken()	849
11.15.4.48	device_registerCameraCallBk()	849
11.15.4.49	device_registerCardStatusFrontSwitchCallBk()	849
11.15.4.50	device_registerRKICallBk()	849
11.15.4.51	device_SendDataCommandNEO()	849
11.15.4.52	device_setBurstMode()	850
11.15.4.53	device_setCurrentDevice()	851
11.15.4.54	device_setMerchantRecord()	851
11.15.4.55	device_setPollMode()	851
11.15.4.56	device_setRKI_URL()	852
11.15.4.57	device_setRTCDateTime()	852
11.15.4.58	device_setSDKWaitTime()	853
11.15.4.59	device_setThreadStackSize()	853
11.15.4.60	device_setTransactionExponent()	853
11.15.4.61	device_startRKI()	853
11.15.4.62	device_startTransaction()	854
11.15.4.63	emv_activateTransaction()	854
11.15.4.64	emv_allowFallback()	855
11.15.4.65	emv_authenticateTransaction()	855
11.15.4.66	emv_authenticateTransactionWithTimeout()	856
11.15.4.67	emv_cancelTransaction()	856
11.15.4.68	emv_completeTransaction()	857

11.15.4.69	emv_getAutoAuthenticateTransaction()	857
11.15.4.70	emv_getAutoCompleteTransaction()	857
11.15.4.71	emv_registerCallBk()	858
11.15.4.72	emv_removeAllApplicationData()	858
11.15.4.73	emv_removeAllCAPK()	858
11.15.4.74	emv_removeAllCRL()	858
11.15.4.75	emv_removeApplicationData()	858
11.15.4.76	emv_removeCAPK()	860
11.15.4.77	emv_removeCRL()	860
11.15.4.78	emv_retrieveAIDList()	861
11.15.4.79	emv_retrieveApplicationData()	861
11.15.4.80	emv_retrieveCAPK()	861
11.15.4.81	emv_retrieveCAPKList()	862
11.15.4.82	emv_retrieveCRL()	862
11.15.4.83	emv_retrieveTerminalData()	863
11.15.4.84	emv_setApplicationData()	863
11.15.4.85	emv_setApplicationDataTLV()	864
11.15.4.86	emv_setAutoAuthenticateTransaction()	864
11.15.4.87	emv_setAutoCompleteTransaction()	864
11.15.4.88	emv_setCAPK()	865
11.15.4.89	emv_setCRL()	865
11.15.4.90	emv_setTerminalData()	866
11.15.4.91	emv_setTerminalMajorConfiguration()	866
11.15.4.92	emv_setTransactionParameters()	867
11.15.4.93	emv_startTransaction()	867
11.15.4.94	executeTransaction()	868
11.15.4.95	executeTransaction_WorldNet()	868
11.15.4.96	forwardTransaction()	869
11.15.4.97	forwardTransaction_WorldNet()	869
11.15.4.98	icc_exchangeAPDU()	870
11.15.4.99	icc_getICCRReaderStatus()	870
11.15.4.100	icc_powerOffICC()	871
11.15.4.101	icc_powerOnICC()	871
11.15.4.102	msr_cancelMSRSwipe()	871
11.15.4.103	msr_registerCallBk()	872
11.15.4.104	msr_registerCallBkp()	872
11.15.4.105	msr_startMSRSwipe()	872
11.15.4.106	parseMSRData()	872
11.15.4.107	in_registerCallBk()	872
11.15.4.108	registerHotplugCallBk()	873

11.15.4.10 registerLogCallBk()	873
11.15.4.11 id232_device_init()	873
11.15.4.12 SDK_Version()	874
11.15.4.13 SetAbsoluteLibraryPath()	874
11.16 Source_C/libIDT_VP3300_USB.h File Reference	874
11.16.1 Detailed Description	877
11.16.2 Macro Definition Documentation	877
11.16.2.1 IN	877
11.16.2.2 IN_OUT	877
11.16.2.3 OUT	877
11.16.3 Typedef Documentation	877
11.16.3.1 ftpComm_callBack	878
11.16.3.2 httpComm_callBack	878
11.16.3.3 pCMR_callBack	878
11.16.3.4 pCSFS_callBack	878
11.16.3.5 pEMV_callBack	878
11.16.3.6 pMessageHotplug	878
11.16.3.7 pMSR_callBack	878
11.16.3.8 pMSR_callBackp	878
11.16.3.9 pPIN_callBack	879
11.16.3.10 pReadDataLog	879
11.16.3.11 pRKI_callBack	879
11.16.3.12 pSendDataLog	879
11.16.3.13 pWN_callBack	879
11.16.3.14 pWP_callBack	879
11.16.3.15 v4Comm_callBack	879
11.16.4 Function Documentation	880
11.16.4.1 cancelWorldNet()	880
11.16.4.2 cancelWorldPay()	880
11.16.4.3 comm_registerHTTPCallback()	880
11.16.4.4 comm_registerV4Callback()	880
11.16.4.5 config_getSerialNumber()	881
11.16.4.6 config_getSerialNumber_Len()	881
11.16.4.7 ctls_activateTransaction()	881
11.16.4.8 ctls_cancelTransaction()	883
11.16.4.9 ctls_getAllConfigurationGroups()	883
11.16.4.10 ctls_getConfigurationGroup()	883
11.16.4.11 ctls_registerCallBk()	884
11.16.4.12 ctls_registerCallBkp()	884
11.16.4.13 ctls_removeAllApplicationData()	884

11.16.4.14	ctls_removeAllCAPK()	884
11.16.4.15	ctls_removeApplicationData()	884
11.16.4.16	ctls_removeCAPK()	885
11.16.4.17	ctls_removeConfigurationGroup()	885
11.16.4.18	ctls_retrieveAIDList()	885
11.16.4.19	ctls_retrieveApplicationData()	886
11.16.4.20	ctls_retrieveCAPK()	886
11.16.4.21	ctls_retrieveCAPKList()	887
11.16.4.22	ctls_retrieveTerminalData()	887
11.16.4.23	ctls_setApplicationData()	888
11.16.4.24	ctls_setCAPK()	888
11.16.4.25	ctls_setConfigurationGroup()	889
11.16.4.26	ctls_setTerminalData()	889
11.16.4.27	ctls_startTransaction()	890
11.16.4.28	device_activateTransaction()	891
11.16.4.29	device_cancelTransaction()	893
11.16.4.30	device_close()	893
11.16.4.31	device_controlUserInterface()	893
11.16.4.32	device_enablePassThrough()	895
11.16.4.33	device_getCurrentDeviceType()	895
11.16.4.34	device_getFirmwareVersion()	895
11.16.4.35	device_getFirmwareVersion_Len()	895
11.16.4.36	device_getIDGStatusCodeString()	896
11.16.4.37	device_getMerchantRecord()	897
11.16.4.38	device_getMerchantRecord_Len()	898
11.16.4.39	device_getRTCDateTime()	898
11.16.4.40	device_getSDKWaitTime()	899
11.16.4.41	device_getThreadStackSize()	899
11.16.4.42	device_getTransactionResults()	899
11.16.4.43	device_init()	900
11.16.4.44	device_isAttached()	900
11.16.4.45	device_isConnected()	900
11.16.4.46	device_pingDevice()	900
11.16.4.47	device_pollForToken()	901
11.16.4.48	device_registerCameraCallBk()	901
11.16.4.49	device_registerCardStatusFrontSwitchCallBk()	901
11.16.4.50	device_registerRKICallBk()	901
11.16.4.51	device_SendDataCommandNEO()	901
11.16.4.52	device_setBurstMode()	902
11.16.4.53	device_setCurrentDevice()	903

11.16.4.54	device_setMerchantRecord()	903
11.16.4.55	device_setPollMode()	903
11.16.4.56	device_setRKI_URL()	904
11.16.4.57	device_setRTCDateTime()	904
11.16.4.58	device_setSDKWaitTime()	905
11.16.4.59	device_setThreadStackSize()	905
11.16.4.60	device_setTransactionExponent()	905
11.16.4.61	device_startRKI()	905
11.16.4.62	device_startTransaction()	906
11.16.4.63	emv_activateTransaction()	906
11.16.4.64	emv_allowFallback()	907
11.16.4.65	emv_authenticateTransaction()	907
11.16.4.66	emv_authenticateTransactionWithTimeout()	908
11.16.4.67	emv_cancelTransaction()	908
11.16.4.68	emv_completeTransaction()	909
11.16.4.69	emv_getAutoAuthenticateTransaction()	909
11.16.4.70	emv_getAutoCompleteTransaction()	909
11.16.4.71	emv_registerCallBk()	910
11.16.4.72	emv_removeAllApplicationData()	910
11.16.4.73	emv_removeAllCAPK()	910
11.16.4.74	emv_removeAllCRL()	910
11.16.4.75	emv_removeApplicationData()	910
11.16.4.76	emv_removeCAPK()	912
11.16.4.77	emv_removeCRL()	912
11.16.4.78	emv_retrieveAIDList()	913
11.16.4.79	emv_retrieveApplicationData()	913
11.16.4.80	emv_retrieveCAPK()	913
11.16.4.81	emv_retrieveCAPKList()	914
11.16.4.82	emv_retrieveCRL()	914
11.16.4.83	emv_retrieveTerminalData()	915
11.16.4.84	emv_setApplicationData()	915
11.16.4.85	emv_setApplicationDataTLV()	916
11.16.4.86	emv_setAutoAuthenticateTransaction()	916
11.16.4.87	emv_setAutoCompleteTransaction()	916
11.16.4.88	emv_setCAPK()	917
11.16.4.89	emv_setCRL()	917
11.16.4.90	emv_setTerminalData()	918
11.16.4.91	emv_setTerminalMajorConfiguration()	918
11.16.4.92	emv_setTransactionParameters()	919
11.16.4.93	emv_startTransaction()	919

11.16.4.94	executeTransaction()	920
11.16.4.95	executeTransaction_WorldNet()	920
11.16.4.96	forwardTransaction()	921
11.16.4.97	forwardTransaction_WorldNet()	921
11.16.4.98	icc_exchangeAPDU()	922
11.16.4.99	icc_getICCRReaderStatus()	922
11.16.4.100	icc_powerOffICC()	923
11.16.4.101	icc_powerOnICC()	923
11.16.4.102	msr_cancelMSRSwipe()	923
11.16.4.103	msr_registerCallBk()	924
11.16.4.104	msr_registerCallBkp()	924
11.16.4.105	msr_startMSRSwipe()	924
11.16.4.106	parseMSRData()	924
11.16.4.107	pin_registerCallBk()	924
11.16.4.108	registerHotplugCallBk()	925
11.16.4.109	registerLogCallBk()	925
11.16.4.110	SDK_Version()	925
11.16.4.111	setAbsoluteLibraryPath()	925
11.17	Source_C/libIDT_VP8800.h File Reference	925
11.17.1	Detailed Description	929
11.17.2	Macro Definition Documentation	929
11.17.2.1	IN	929
11.17.2.2	IN_OUT	930
11.17.2.3	OUT	930
11.17.3	Typedef Documentation	930
11.17.3.1	ftpComm_callBack	930
11.17.3.2	httpComm_callBack	930
11.17.3.3	pCMR_callBack	930
11.17.3.4	pCSFS_callBack	930
11.17.3.5	pEMV_callBack	930
11.17.3.6	pLog_callback	931
11.17.3.7	pMessageHotplug	931
11.17.3.8	pMSR_callBack	931
11.17.3.9	pMSR_callBackp	931
11.17.3.10	pPIN_callBack	931
11.17.3.11	pReadDataLog	931
11.17.3.12	pSendDataLog	931
11.17.3.13	p4Comm_callBack	931
11.17.4	Function Documentation	932
11.17.4.1	comm_registerHTTPCallback()	932

11.17.4.2 comm_registerV4Callback()	932
11.17.4.3 config_getSerialNumber()	932
11.17.4.4 config_getSerialNumber_Len()	932
11.17.4.5 ctls_activateTransaction()	933
11.17.4.6 ctls_cancelTransaction()	934
11.17.4.7 ctls_displayOnlineAuthResult()	934
11.17.4.8 ctls_getAllConfigurationGroups()	935
11.17.4.9 ctls_getConfigurationGroup()	935
11.17.4.10ctls_registerCallBk()	935
11.17.4.11ctls_registerCallBkp()	936
11.17.4.12ctls_removeAllApplicationData()	936
11.17.4.13ctls_removeAllCAPK()	936
11.17.4.14ctls_removeApplicationData()	936
11.17.4.15ctls_removeCAPK()	936
11.17.4.16ctls_removeConfigurationGroup()	937
11.17.4.17ctls_retrieveAIDList()	937
11.17.4.18ctls_retrieveApplicationData()	938
11.17.4.19ctls_retrieveCAPK()	938
11.17.4.20ctls_retrieveCAPKList()	939
11.17.4.21ctls_retrieveTerminalData()	939
11.17.4.22ctls_setApplicationData()	940
11.17.4.23ctls_setCAPK()	940
11.17.4.24ctls_setConfigurationGroup()	941
11.17.4.25ctls_setTerminalData()	941
11.17.4.26ctls_startTransaction()	942
11.17.4.27device_activateTransaction()	943
11.17.4.28device_calibrateParameters()	945
11.17.4.29device_cancelTransaction()	945
11.17.4.30device_close()	945
11.17.4.31device_controlIndicator()	945
11.17.4.32device_controlUserInterface()	946
11.17.4.33device_createDirectory()	948
11.17.4.34device_deleteDirectory()	948
11.17.4.35device_deleteFile()	948
11.17.4.36device_enablePassThrough()	949
11.17.4.37device_enhancedPassthrough()	949
11.17.4.38device_getCurrentDeviceType()	949
11.17.4.39device_getDriveFreeSpace()	950
11.17.4.40device_getFirmwareVersion()	950
11.17.4.41device_getFirmwareVersion_Len()	950

11.17.4.42	device_getIDGStatusCodeString()	951
11.17.4.43	device_getMerchantRecord()	952
11.17.4.44	device_getMerchantRecord_Len()	953
11.17.4.45	device_getSDKWaitTime()	953
11.17.4.46	device_getThreadStackSize()	953
11.17.4.47	device_getTransactionResults()	954
11.17.4.48	device_init()	954
11.17.4.49	device_isAttached()	954
11.17.4.50	device_isConnected()	955
11.17.4.51	device_listDirectory()	955
11.17.4.52	device_pingDevice()	955
11.17.4.53	device_registerCameraCallBk()	956
11.17.4.54	device_registerCardStatusFrontSwitchCallBk()	956
11.17.4.55	device_SendDataCommandNEO()	956
11.17.4.56	device_setCurrentDevice()	957
11.17.4.57	device_setMerchantRecord()	957
11.17.4.58	device_setSDKWaitTime()	958
11.17.4.59	device_setThreadStackSize()	958
11.17.4.60	device_setTransactionExponent()	958
11.17.4.61	device_startTransaction()	958
11.17.4.62	device_transferFile()	960
11.17.4.63	emv_activateTransaction()	960
11.17.4.64	emv_allowFallback()	961
11.17.4.65	emv_authenticateTransaction()	961
11.17.4.66	emv_authenticateTransactionWithTimeout()	961
11.17.4.67	emv_cancelTransaction()	962
11.17.4.68	emv_completeTransaction()	962
11.17.4.69	emv_getAutoAuthenticateTransaction()	963
11.17.4.70	emv_getAutoCompleteTransaction()	963
11.17.4.71	emv_getEMVConfigurationCheckValue()	963
11.17.4.72	emv_getEMVKernelCheckValue()	964
11.17.4.73	emv_getEMVKernelVersion()	964
11.17.4.74	emv_getEMVKernelVersion_Len()	964
11.17.4.75	emv_registerCallBk()	966
11.17.4.76	emv_removeAllApplicationData()	966
11.17.4.77	emv_removeAllCAPK()	966
11.17.4.78	emv_removeAllCRL()	966
11.17.4.79	emv_removeAllExceptions()	967
11.17.4.80	emv_removeApplicationData()	967
11.17.4.81	emv_removeCAPK()	967

11.17.4.82	emv_removeCRL()	968
11.17.4.83	emv_removeException()	968
11.17.4.84	emv_removeTransactionLog()	968
11.17.4.85	emv_retrieveAIDList()	969
11.17.4.86	emv_retrieveApplicationData()	969
11.17.4.87	emv_retrieveCAPK()	969
11.17.4.88	emv_retrieveCAPKList()	970
11.17.4.89	emv_retrieveCRL()	970
11.17.4.90	emv_retrieveExceptionList()	971
11.17.4.91	emv_retrieveExceptionLogStatus()	971
11.17.4.92	emv_retrieveTerminalData()	972
11.17.4.93	emv_retrieveTransactionLog()	972
11.17.4.94	emv_retrieveTransactionLogStatus()	974
11.17.4.95	emv_setApplicationData()	974
11.17.4.96	emv_setApplicationDataTLV()	974
11.17.4.97	emv_setAutoAuthenticateTransaction()	975
11.17.4.98	emv_setAutoCompleteTransaction()	975
11.17.4.99	emv_setCAPK()	975
11.17.4.100	emv_setCRL()	976
11.17.4.101	emv_setException()	976
11.17.4.102	emv_setTerminalData()	978
11.17.4.103	emv_startTransaction()	978
11.17.4.104	ed_addItemToList()	979
11.17.4.105	ed_cancelSlideShow()	979
11.17.4.106	ed_captureSignature()	980
11.17.4.107	ed_clearDisplay()	980
11.17.4.108	ed_clearEventQueue()	980
11.17.4.109	ed_createInputField()	981
11.17.4.110	ed_createInputField_Len()	982
11.17.4.111	ed_createList()	983
11.17.4.112	ed_createList_Len()	985
11.17.4.113	ed_customDisplayMode()	986
11.17.4.114	ed_displayButton()	986
11.17.4.115	ed_displayButton_Len()	988
11.17.4.116	ed_displayParagraph()	989
11.17.4.117	ed_displayText()	991
11.17.4.118	ed_displayText_Len()	992
11.17.4.119	ed_getInputEvent()	993
11.17.4.120	ed_getInputEvent_Len()	995
11.17.4.121	ed_getInputFieldValue()	997

11.17.4.122d_getSelectedItem()	998
11.17.4.123d_getSelectedItem_Len()	998
11.17.4.124d_resetInitialState()	998
11.17.4.125d_setBackgroundImage()	998
11.17.4.126d_setDisplayImage()	999
11.17.4.127d_setForeBackColor()	1000
11.17.4.128d_startSlideShow()	1000
11.17.4.129msr_cancelMSRSwipe()	1001
11.17.4.130msr_flushTrackData()	1001
11.17.4.131msr_registerCallBk()	1001
11.17.4.132msr_registerCallBkp()	1001
11.17.4.133msr_startMSRSwipe()	1002
11.17.4.134parseMSRData()	1002
11.17.4.135in_getEncryptedOnlinePIN()	1002
11.17.4.136in_getPAN()	1003
11.17.4.137in_promptCreditDebit()	1003
11.17.4.138in_registerCallBk()	1004
11.17.4.139registerHotplugCallBk()	1004
11.17.4.140registerLogCallBk()	1004
11.17.4.141SDK_Version()	1004
11.17.4.142GetAbsoluteLibraryPath()	1004
11.17.4.143s_deleteSSLCert()	1004
11.17.4.144s_getCertChainType()	1005
11.17.4.145s_loadSSLCert()	1005
11.17.4.146s_requestCSR()	1006
11.17.4.147s_revokeSSLCert()	1006
11.17.4.148s_updateRootCertificate()	1006

Chapter 1

ID TECH Universal SDK Reference Guide for Linux/Windows/Mac (C++)

ID TECH provides this Universal SDK to drive multiple devices across multiple platforms.

The current version of the SDK supports the USB-HID interface of the listed ID TECH products. For devices that also have RS-232 interfaces, SDK communication support for COM on those devices is under development and will be released at a later date.

This SDK encompasses support for the following devices and platforms. Other ID TECH products are scheduled to be added in an upcoming release.

C/C++

- **Platforms:** Macintosh, Windows, Linux Desktop (x86_64/amd64), Linux ARM (RaspberryPi)
- **Products:** UniPay 1.5, UniPay III, VP4880, MiniSmartII, BTPayMini, SpectrumPro, Kiosk III, Augusta

Pre-requisites:

- **Macintosh:** None
- **Windows, Linux:** libusb-1.0
Note:** We recommended installing libusb from the distributors' downloads. If that becomes a challenge, the SDK includes just the libusb library for Windows, Linux x86_64, and Linux ARM.

1.1 Demo Apps

Pre-Requisites C/C++:

- Eclipse for C/C++
- Windows: MinGW
- Linux ARM on x86_64 (cross compile): Poky

1.2 Purpose

This document describes API requirements as well as the interface definitions and requirements for an integrator wishing to integrate it into a payment application.

- [Core Implementation: C/C++](#)

- [Important Security Notice](#)
- [Main Transaction Commands](#)
- [EMV Callback](#)
- [EMV Tag Reference](#)
- [Enumeration Reference](#)
- [Error Code Reference](#)
- [LCD Foreign Language Mapping Table](#)

Chapter 2

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 MasterCard 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.

2.1 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 merchant's 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.

2.2 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.

2.3 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.

2.4 PA-DSS Guidelines

The following PA-DSS Guidelines are being provided by ID TECH 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.

2.5 More Information

ID TECH 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 3

Main Transaction Commands

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

3.1 EMV Methods

Start EMV Transaction

`emv_startTransaction()`

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

Authenticate EMV Transaction

`emv_authenticateTransaction()`

When the results to `emv_startTransaction()` come back as **EMV_RESULT_CODE.EMV_RESULT_CODE_AUTHENTICATE_TRANSACTION**, continuing the EMV transaction requires calling this method.

Complete Online EMV Transaction

`emv_completeTransaction()`

If start/authenticate transaction returns **EMV_RESULT_CODE.EMV_RESULT_CODE_GO_ONLINE**, finishing the transaction requires executing `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, finishing the EMV transaction still requires passing "TRUE" for **commError**.

Terminal Configuration

`emv_retrieveTerminalData()`

`emv_removeTerminalData()`

`emv_setTerminalData()`

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

AID Management

`emv_retrieveApplicationData()`

`emv_removeApplicationData()`

`emv_removeAllApplicationData()`

`emv_setApplicationData()`

`emv_retrieveAIDList()`

Methods for AID management. When setting the AID, pass the tags in TLV format. When retrieving the AID, receive the results as tags in TLV format.

CAPK Management

`emv_retrieveCAPK()`
`emv_removeCAPK()`
`emv_removeAllCAPK()`
`emv_setCAPK()`
`emv_retrieveCAPKList()`

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

CRL Management

`emv_removeCRL()`
`emv_removeAllCRL()`
`emv_retrieveCRL()`
`emv_setCRL()`

Methods for Certificate Revocation List management.

Kernel Version

`emv_getEMVKernelVersion()`

Method to retrieve the kernel version.

Kernel Check Value

`emv_getEMVKernelCheckValue()`

Method to retrieve the kernel Check Value.

EMV Configuration Check Value

`emv_getEMVConfigurationCheckValue()`

Method to retrieve the EMV configuration check value.

3.2 MSR Methods

Start MSR Swipe

`msr_startMSRSwipe()`

Starts a swipe request. Returns card data in the callback method.

Cancel MSR Swipe

`msr_cancelMSRSwipe()`

Cancels a swipe request.

Chapter 4

Core Implementation: C/C++

IDTechSDK (libIDTechSDK.so-x_xx_xxx / libIDTechSDK-x_xx_xxx.dll / libIDTechSDK-x_xx_xxx.dylib) includes API methods to interface with ID TECH devices. This guide assumes a fair understanding of Eclipse, C, and general Linux, Windows, or Mac programming knowledge.

4.1 Integrating with IdTechSDK

- [Import the Necessary Libraries](#)
- [Add Include Statements to Use Libraries](#)
- [Implement the Callback Function](#)
- [Initialize the Target Device](#)

4.2 Import the Necessary Libraries

Header Files

Communicating with ID TECH devices requires that developers include the following header files in the project's source code folder:

- IDTDef.h
- The appropriate device header file for a single device OR **IDT_Device.h** to expose all methods for all devices. Usually, using the single device header file is appropriate as it filters out all unrelated methods.

Libraries

- libUSB installed on the system (not applicable to Mac).
- Place the IDTechSDK library file the system PATH of the target device.

IMPORTANT:

IDTechSDK libraries are distributed with versioning as part of the library name (libIDTechSDK.so-x_xx_xx x/ libIDTechSDK-x_xx_xxx.dll / libIDTechSDK-x_xx_xxx.dylib). The system must recognize them as a file WITHOUT the version info ((libIDTechSDK.so / libIDTechSDK.dll / libIDTechSDK.dylib). Accomplish this by either RENAMING the libraries by removing the version info OR create a Symbolic Link pointing to the original libraries and then remove the version info from the symbolic link:

(Symbolic Link) libIDTechSDK.so -> libIDTechSDK.so-x_xx_xxx (Compiled Library)

(Symbolic Link) libIDTechSDK.dll -> libIDTechSDK-x_xx_xxx.dll (Compiled Library)

(Symbolic Link) libIDTechSDK.dylib -> libIDTechSDK-x_xx_xxx.dylib (Compiled Library)

(Symbolic Link) /lib/libIDTechSDK.so -> /home/<USER>/proj/libIDTechSDK.so-x_xx_xxx (Compiled Library)

4.3 Add Include Statements to Use Libraries

Add a line of code to use the header files for IDTechSDK at the start of the file (a SpectrumPro header file is used here as an example):

```
#include <stdlib.h>
#include <stdio.h>

#include "IDTDef.h"
#include "libIDT_SpectrumPro.h"
```

4.4 Implement the Callback Function

There are two callbacks to implement, one for MSR and one for EMV:

```
void MSR_callBack(int type, IDTMSRData cardData){
    printf("\nMSR Callback\n");
    switch (type){
        case MSR_callBack_type_ERR:
            printf("Callback MSR cancelled\n");
            break;
        case MSR_callBack_type_RETURN_CODE:
            printf("Callback MSR data received\n");
            break;
        case MSR_callBack_type_TIMEOUT:
            printf("MSR Callback Timeout\n");
            break;
        default:
            break;
    }
}

void EMV_callBack(int device_type, int device_state, unsigned char * data, int dataLen, IDTTransactionData*
cardData, EMV_Callback* emvCallback, int transactionResultCode){

    switch(device_state)
    {
        case EMVCallback:
            printf ("EMV Callback\n");
            break;
        case TransactionData:
            printf ("Transaction Data Callback\n");
            break;
        case TransactionFailed:
            printf("Transaction Failed Callback");
            break;
    }
}
```

4.5 Initialize SDK and Set the Target Device

Perform [device_init\(\)](#) to initialize the SDK, establish the callbacks, and use the [device_setCurrentDevice\(\)](#) function to specify the device to use.

```
int main(void) {
    int r = 0;
    printf("Initializing SDK...\n");
    r = device_init();
    if ( r != RETURN_CODE_DO_SUCCESS ) {
        printf(" Fail to init!\n");
        return 0;
    }
    emv_registerCallBk (EMV_callBack);
```

```
msr_registerCallBk (MSR_callBack);  
device_setCurrentDevice (IDT_DEVICE_SPECTRUM_PRO)  
return 0;  
}
```

Chapter 5

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 CORR↔ ECTO?	确定金
3	MSG_APPROV↔ ED	APPROVED	APPROUVE	APROVADO	通
4	MSG_CALL_Y↔ OUR_BANK	CALL YOUR BA↔ NK	APPE VOTRE B↔ ANQE	LLAME A SU BA↔ NCO	系您的行
5	MSG_CANCEL↔ _OR_ENTER	CANCEL OR E↔ ENTER	ANNULE OU E↔ NTRER	CANCEL O ENT↔ RAR	取消或确定
6	MSG_CARD_E↔ RROR	CARD ERROR	ERREUR CARTE	ERROR DE TA↔ RJETA	卡
7	MSG_DECLINED	DECLINED	REFUSE	DECLINADO	卡被拒
8	MSG_ENTER_↔ AMOUNT	ENTER AMOUNT	ENTRER MONT↔ ANT	INGRESE MON↔ TO	入金
9	MSG_ENTER_↔ PIN	ENTER PIN:	ENTRER PIN:	ENTRAR NPI:	入密
10	MSG_INCORR↔ ECT_PIN	INCORRECT PIN	NIP INCORRECT	NPI INCORREC↔ TO	密
11	MSG_ICC_MSR1	SWIPE OR INS↔ ERT	PASSER OU IN↔ SERT	MOVER O INSE↔ RT	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 REA↔ DER UTI	LECTEUR CHIP	USO CHIP LEC↔ TOR	使用芯片卡
15	MSG_NOT_AC↔ CEPTED	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 E↔ SPERE	等候中
18	MSG_PROCES↔ SING_ERROR	PROCESSING ERROR	ERREUR DE T↔ RAITE	ERROR PROC↔ ESANDO	理
19	MSG_USE_MA↔ GSTRIPE	USE MAGSTRIPE	USAGE MAGST↔ RIPE	USO DE MAGS↔ TRIPE	使用磁卡

ID	Message ID	English	French	Spanish	Chinese
20	MSG_TRY_AGAIN	TRY AGAIN	REESSAYER	VUELV INTENTARLO	重
21	MSG_ONLINE	GO ONLINE	GO LIGNE	GO LINEA	在
22	MSG_TRANSACTION_ERROR	TRANSACTION ERROR	ERREUR DE TRANS	ERROR DE TRANSAC	交易
23	MSG_TERMINATE	TERMINATE	RESILIER	TERMINAR	止
24	MSG_ADVICE	ADVICE	CONSEILS	CONSEJOS	建
25	MSG_TIMEOUT	TIME OUT	TIMEOUT	TIEMPO DE ESPERA	超
26	MSG_PROCESSING	PROCESSING...	PROCESSUS...	PROCESANDO...	理中。。。
27	MSG_PIN_TRY_EX	PIN TRY LIMIT EX	PIN TRY DEPASSE	TRY PIN SUPERADA	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 PROCES	理
30	MSG_GET_PIN_ERROR	GET PIN ERROR	GET PIN ERROR	OBTENER PIN ERR	密
31	MSG_GET_PIN_FAIL	GET PIN FAIL	GET PIN FAIL	OBTENER PIN FALL	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
35	MSG_WELCOME	WELCOME	BIENVENUE	BIENVENIDOS	迎使用
36	MSG_AMOUNT_OTHER	AMOUNT OTHER	MONTANT AUTRES	IMPORTE OTRAS	返
37	MSG_ENTER_AMOUNT_OTHER	ENTER AMOUNT OTHER	ENTRER MONTANT AUTRES	ENTRAR IMPORTE OTRAS	入返
38	MSG_CAPK_HASH_VALUE_FAIL	CAPK HASH VALUE FAIL	CAPK HASH VALEUR FAIL	CAPK HASH VALOR FAIL	公哈希值
64	MSG_TRY_MSR_AGAIN	TRY MSR AGAIN	-	-	再次使用磁卡
65	MSG_LAST_MSR_TRY	LAST MSR TRY	-	-	最后一次使用磁卡
66	MSG_TRY_ICC_AGAIN	TRY ICC AGAIN	INSERER VOTRE CARTE	INTENTE ICC DE NUEVO	再次使用芯片卡
67	MSG_REMOVE_CARD	REMOVE CARD	RETIRER VOTRE CARTE	QUITE TARJETA	移卡

Chapter 6

Error Code Reference

```

0: "no error, beginning task";
1: "no response from reader";
2: "invalid response data";
3: "time out for task or CMD";
4: "wrong parameter";
5: "SDK is doing MSR or ICC task";
6: "SDK is doing PINPad task";
7: "SDK is doing CTLS task";
8: "SDK is doing EMV task";
9: "SDK is doing Other task";
10: "err response or data";
11: "no reader attached";
12: "mono audio is enabled";
13: "did connection";
14: "audio volume is too low";
15: "task or CMD be canceled";
16: "UF wrong string format";
17: "UF file not found";
18: "UF wrong file format";
19: "Attempt to contact online host failed";
20: "Attempt to perform RKI failed";
22: "Buffer size is not enough";
0x300: "Key Type(TDES) of Session Key is not same as the related Master Key.";
0x400: "Related Key was not loaded.";
0x500: "Key Same.";
0x501: "Key is all zero";
0x502: "TR-31 format error";
0x702: "PAN is Error Key.";
0x705: "No Internal MSR PAN (or Internal MSR PAN is erased timeout)";
0X0C01: "Incorrect Frame Tag";
0X0C02: "Incorrect Frame Type";
0X0C03: "Unknown Frame Type";
0X0C04: "Unknown Command";
0X0C05: "Unknown Sub-Command";
0X0C06: "CRC Error";
0X0C07: "Failed";
0X0C08: "Timeout";
0X0C0A: "Incorrect Parameter";
0X0C0B: "Command Not Supported";
0X0C0C: "Sub-Command Not Supported";
0X0C0D: "Parameter Not Supported / Status Abort Command";
0X0C0F: "Sub-Command Not Allowed";
0X0D01: "Incorrect Header Tag";
0X0D02: "Unknown Command";
0X0D03: "Unknown Sub-Command";
0X0D04: "CRC Error in Frame";
0X0D05: "Incorrect Parameter";
0X0D06: "Parameter Not Supported";
0X0D07: "Mal-formatted Data";
0X0D08: "Timeout";
0X0D0A: "Failed / NACK";
0X0D0B: "Command not Allowed";
0X0D0C: "Sub-Command not Allowed";
0X0D0D: "Buffer Overflow (Data Length too large for reader buffer)";
0X0D0E: "User Interface Event";
0X0D11: "Communication type not supported, VT-1, burst, etc.";
0X0D12: "Secure interface is not functional or is in an intermediate state.";
0X0D13: "Data field is not mod 8";
0X0D14: "Pad 0x80 not found where expected";

```

```

0X0D15: "Specified key type is invalid";
0X0D1: "Could not retrieve key from the SAM(InitSecureComm)";
0X0D17: "Hash code problem";
0X0D18: "Could not store the key into the SAM(InstallKey)";
0X0D19: "Frame is too large";
0X0D1A: "Unit powered up in authentication state but POS must resend the InitSecureComm command";
0X0D1B: "The EEPROM may not be initialized because SecCommInterface does not make sense";
0X0D1C: "Problem encoding APDU";
0X0D20: "Unsupported Index(ILM) SAM Transceiver error - problem communicating with the SAM(Key Mgr)";
0X0D2: "Unexpected Sequence Counter in multiple frames for single bitmap(ILM) Length error in data returned
from the SAM(Key Mgr)";
0X0D22: "Improper bit map(ILM)";
0X0D23: "Request Online Authorization";
0X0D24: "ViVOCard3 raw data read successful";
0X0D25: "Message index not available(ILM) ViVOcomm activate transaction card type(ViVOcomm)";
0X0D26: "Version Information Mismatch(ILM)";
0X0D27: "Not sending commands in correct index message index(ILM)";
0X0D28: "Time out or next expected message not received(ILM)";
0X0D29: "ILM languages not available for viewing(ILM)";
0X0D2A: "Other language not supported(ILM)";
0X0D41: "Unknown Error from SAM";
0X0D42: "Invalid data detected by SAM";
0X0D43: "Incomplete data detected by SAM";
0X0D44: "Reserved";
0X0D45: "Invalid key hash algorithm";
0X0D46: "Invalid key encryption algorithm";
0X0D47: "Invalid modulus length";
0X0D48: "Invalid exponent";
0X0D49: "Key already exists";
0X0D4A: "No space for new RID";
0X0D4B: "Key not found";
0X0D4C: "Crypto not responding";
0X0D4D: "Crypto communication error";
0X0D4E: "Module-specific error for Key Manager";
0X0D4F: "All key slots are full (maximum number of keys has been installed)";
0X0D50: "Auto-Switch OK";
0X0D51: "Auto-Switch failed";
0X0D90: "Account DUKPT Key not exist";
0X0D91: "Account DUKPT Key KSN exhausted";
0x0D00: "This Key had been loaded.";
0x0E00: "Base Time was loaded.";
0x0F00: "Encryption Or Decryption Failed.";
0x1000: "Battery Low Warning (It is High Priority Response while Battery is Low.)";
0x1800: "Send 'Cancel Command' after send 'Get Encrypted PIN' & 'Get Numeric' & 'Get Amount'";
0x1900: "Press 'Cancel' key after send 'Get Encrypted PIN' & 'Get Numeric' & 'Get Amount'";
0x30FF: "Security Chip is not connect";
0x3000: "Security Chip is deactivation & Device is In Removal Legally State.";
0x3101: "Security Chip is activation & Device is In Removal Legally State.";
0x5500: "No Admin DUKPT Key.";
0x5501: "Admin DUKPT Key STOP.";
0x5502: "Admin DUKPT Key KSN is Error.";
0x5503: "Get Authentication Code1 Failed.";
0x5504: "Validate Authentication Code Error.";
0x5505: "Encrypt or Decrypt data failed.";
0x5506: "Not Support the New Key Type.";
0x5507: "New Key Index is Error.";
0x5508: "Step Error.";
0x5509: "KSN Error";
0x550A: "MAC Error.";
0x550B: "Key Usage Error.";
0x550C: "Mode Of Use Error.";
0x550F: "Other Error.";
0x6000: "Save or Config Failed / Or Read Config Error.";
0x6200: "No Serial Number.";
0x6900: "Invalid Command - Protocol is right, but task ID is invalid.";
0x6A01: "Unsupported Command - Protocol and task ID are right, but command is invalid - In this State";
0x6A00: "Unsupported Command - Protocol and task ID are right, but command is invalid.";
0x6B00: "Unknown parameter in command - Protocol task ID and command are right, but parameter is invalid.";
0x6C00: "Unknown parameter in command - Protocol task ID and command are right, but length is out of the
requirement.";
0x7200: "Device is suspend (MKSK suspend or press password suspend).";
0x7300: "PIN DUKPT is STOP (21 bit 1).";
0x7400: "Device is Busy.";
0xE100: "Can not enter sleep mode";
0xE200: "File has existed";
0xE300: "File has not existed";
0xE313: "IO line low -- Card error after session start";
0xE400: "Open File Error";
0xE500: "SmartCard Error";
0xE600: "Get MSR Card data is error";
0xE700: "Command time out";
0xE800: "File read or write is error";
0xE900: "Active 1850 error!";
0xEA00: "Load bootloader error";
0xEF00: "Protocol Error- STX or ETX or check error.";
0xEB00: "Picture is not exist";
0x2C02: "No Microprocessor ICC seated";

```

```

0x2C06: "no card seated to request ATR";
0x2D01: "Card Not Supported,";
0x2D03: "Card Not Supported, wants CRC";
0x690D: "Command not supported on reader without ICC support";
0x8100: "ICC error time out on power-up";
0x8200: "invalid TS character received - Wrong operation step";
0x8300: "Decode MSR Error";
0x8400: "TriMagII no Response";
0x8500: "No Swipe MSR Card";
0x8510: "No Financial Card";
0x8600: "Unsupported F, D, or combination of F and D";
0x8700: "protocol not supported EMV TD1 out of range";
0x8800: "power not at proper level";
0x8900: "ATR length too long";
0x8B01: "EMV invalid TA1 byte value";
0x8B02: "EMV TB1 required";
0x8B03: "EMV Unsupported TB1 only 00 allowed";
0x8B04: "EMV Card Error, invalid BWI or CWI";
0x8B06: "EMV TB2 not allowed in ATR";
0x8B07: "EMV TC2 out of range";
0x8B08: "EMV TC2 out of range";
0x8B09: "per EMV96 TA3 must be > 0xF";
0x8B10: "ICC error on power-up";
0x8B11: "EMV T=1 then TB3 required";
0x8B12: "Card Error, invalid BWI or CWI";
0x8B13: "Card Error, invalid BWI or CWI";
0x8B17: "EMV TC1/TB3 conflict-";
0x8B20: "EMV TD2 out of range must be T=1";
0x8C00: "TCK error";
0xA304: "connector has no voltage setting";
0xA305: "ICC error on power-up invalid (SBLK(IFSD) exchange";
0xE301: "ICC error after session start";
0xFF00: "Request to go online";
0xFF01: "EMV: Accept the offline transaction";
0xFF02: "EMV: Decline the offline transaction";
0xFF03: "EMV: Accept the online transaction";
0xFF04: "EMV: Decline the online transaction";
0xFF05: "EMV: Application may fallback to magstripe technology";
0xFF06: "EMV: ICC detected tah the conditions of use are not satisfied";
0xFF07: "EMV: ICC didn't accept transaction";
0xFF08: "EMV: Transaction was cancelled";
0xFF09: "EMV: Application was not selected by kernel or ICC format error or ICC missing data error";
0xFF0A: "EMV: Transaction is terminated";
0xFF0B: "EMV: Other EMV Error";
0xFFFF: "NO RESPONSE";
0xF002: "ICC communication timeout";
0xF003: "ICC communication Error";
0xF00F: "ICC Card Seated and Highest Priority, disable MSR work request";
0xF200: "AID List / Application Data is not exist";
0xF201: "Terminal Data is not exist";
0xF202: "TLV format is error";
0xF203: "AID List is full";
0xF204: "Any CA Key is not exist";
0xF205: "CA Key RID is not exist";
0xF206: "CA Key Index it not exist";
0xF207: "CA Key is full";
0xF208: "CA Key Hash Value is Error";
0xF209: "Transaction format error";
0xF20A: "The command will not be processing";
0xF20B: "CRL is not exist";
0xF20C: "CRL number exceed max number";
0xF20D: "Amount,Other Amount,Trasaction Type are missing";
0xF20E: "The Identification of algorithm is mistake";
0xF20F: "No Financial Card";
0xF210: "In Encrypt Result state, TLV total Length is greater than Max Length";
0x1001: "INVALID ARG";
0x1002: "FILE_OPEN_FAILED";
0x1003: "FILE_OPERATION_FAILED";
0x2001: "MEMORY_NOT_ENOUGH";
0x3002: "SMARTCARD_FAIL";
0x3003: "SMARTCARD_INIT_FAILED";
0x3004: "FALLBACK_SITUATION";
0x3005: "SMARTCARD_ABSENT";
0x3006: "SMARTCARD_TIMEOUT";
0x5001: "EMV_PARSING_TAGS_FAILED";
0x5002: "EMV_DUPLICATE_CARD_DATA_ELEMENT";
0x5003: "EMV_DATA_FORMAT_INCORRECT";
0x5004: "EMV_NO_TERM_APP";
0x5005: "EMV_NO_MATCHING_APP";
0x5006: "EMV_MISSING_MANDATORY_OBJECT";
0x5007: "EMV_APP_SELECTION_RETRY";
0x5008: "EMV_GET_AMOUNT_ERROR";
0x5009: "EMV_CARD_REJECTED";
0x5010: "EMV_AIP_NOT_RECEIVED";
0x5011: "EMV_AFL_NOT_RECEIVED";
0x5012: "EMV_AFL_LEN_OUT_OF_RANGE";
0x5013: "EMV_SFI_OUT_OF_RANGE";

```



```
0x5014: "EMV_AFL_INCORRECT";
0x5015: "EMV_EXP_DATE_INCORRECT";
0x5016: "EMV_EFF_DATE_INCORRECT";
0x5017: "EMV_ISS_COD_TBL_OUT_OF_RANGE";
0x5018: "EMV_CRYPTOGAM_TYPE_INCORRECT";
0x5019: "EMV_PSE_NOT_SUPPORTED_BY_CARD";
0x5020: "EMV_USER_SELECTED_LANGUAGE";
0x5021: "EMV_SERVICE_NOT_ALLOWED";
0x5022: "EMV_NO_TAG_FOUND";
0x5023: "EMV_CARD_BLOCKED";
0x5024: "EMV_LEN_INCORRECT";
0x5025: "CARD_COM_ERROR";
0x5026: "EMV_TSC_NOT_INCREASED";
0x5027: "EMV_HASH_INCORRECT";
0x5028: "EMV_NO_ARC";
0x5029: "EMV_INVALID_ARC";
0x5030: "EMV_NO_ONLINE_COMM";
0x5031: "TRAN_TYPE_INCORRECT";
0x5032: "EMV_APP_NO_SUPPORT";
0x5033: "EMV_APP_NOT_SELECT";
0x5034: "EMV_LANG_NOT_SELECT";
0x5035: "EMV_NO_TERM_DATA";
0x6001: "CVM_TYPE_UNKNOWN";
0x6002: "CVM_AIP_NOT_SUPPORTED";
0x6003: "CVM_TAG_8E_MISSING";
0x6004: "CVM_TAG_8E_FORMAT_ERROR";
0x6005: "CVM_CODE_IS_NOT_SUPPORTED";
0x6006: "CVM_COND_CODE_IS_NOT_SUPPORTED";
0x6007: "NO_MORE_CVM";
0x6008: "PIN_BYPASSED_BEFORE";
0x7001: "PK_BUFFER_SIZE_TOO_BIG";
0x7002: "PK_FILE_WRITE_ERROR";
0x7003: "PK_HASH_ERROR";
0x8001: "NO_CARD HOLDER CONFIRMATION";
0x8002: "GET_ONLINE_PIN";
0xD000: "Data not exist";
0xD001: "Data access error";
0xD100: "RID not exist";
0xD101: "RID existed";
0xD102: "Index not exist";
0xD200: "Maximum exceeded";
0xD201: "Hash error";
0xD205: "System Busy";
0x0E01: "Unable to go online";
0x0E02: "Technical Issue";
0x0E03: "Declined";
0x0E04: "Issuer Referral transaction";
0x0F01: "Decline the online transaction";
0x0F02: "Request to go online";
0x0F03: "Transaction is terminated";
0x0F05: "Application was not selected by kernel or ICC format error or ICC missing data error";
0x0F07: "ICC didn't accept transaction";
0x0F0A: "Application may fallback to magstripe technology";
0x0F0C: "Transaction was cancelled";
0x0F0D: "Timeout";
0x0F0F: "Other EMV Error";
0x0F10: "Accept the offline transaction";
0x0F11: "Decline the offline transaction";
0x0F21: "ICC detected tah the conditions of use are not satisfied";
0x0F22: "No app were found on card matching terminal configuration";
0x0F23: "Terminal file does not exist";
0x0F24: "CAPK file does not exist";
0x0F25: "CRL Entry does not exist";
0x0FFE: "code when blocking is disabled";
0x0FFF: "code when command is not applicable on the selected device";
0xF005: "ICC Encrypted C-APDU Data Structure Length Error Or Format Error.";
0xBBE0: "CM100 Success";
0xBBE1: "CM100 Parameter Error";
0xBBE2: "CM100 Low Output Buffer";
0xBBE3: "CM100 Card Not Found";
0xBBE4: "CM100 Collision Card Exists";
0xBBE5: "CM100 Too Many Cards Exist";
0xBBE6: "CM100 Saved Data Does Not Exist";
0xBBE8: "CM100 No Data Available";
0xBBE9: "CM100 Invalid CID Returned";
0xBBEA: "CM100 Invalid Card Exists";
0xBBEC: "CM100 Command Unsupported";
0xBBED: "CM100 Error In Command Process";
0xBBEE: "CM100 Invalid Command";

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

0X9038: "Wait (the command couldnt be finished in BWT)";
0X9039: "Busy (a previously command has not been finished)";
0X903A: "Number of retries over limit";
```

0X9040: "Invalid Manufacturing system data";
0X9041: "Not authenticated";
0X9042: "Invalid Master DUKPT Key";
0X9043: "Invalid MAC Key";
0X9044: "Reserved for future use";
0X9045: "Reserved for future use";
0X9046: "Invalid DATA DUKPT Key";
0X9047: "Invalid PIN Pairing DUKPT Key";
0X9048: "Invalid DATA Pairing DUKPT Key";
0X9049: "No nonce generated";
0X9949: "No GUID available. Perform getVersion first.";
0X9950: "MAC Calculation unsuccessful. Check BDK value.";

0X904A: "Not ready";
0X904B: "Not MAC data";

0X9050: "Invalid Certificate";
0X9051: "Duplicate key detected";
0X9052: "AT checks failed";
0X9053: "TR34 checks failed";
0X9054: "TR31 checks failed";
0X9055: "MAC checks failed";
0X9056: "Firmware download failed";

0X9060: "Log is full";
0X9061: "Removal sensor unengaged";
0X9062: "Any hardware problems";

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

Chapter 7

Enumeration Reference

Common

```

enum DEVICE_TYPE{
    DEVICE_TYPE_UNKNOWN=0,
    IDT_DEVICE_AUGUSTA,
    AUGUSTA_KB,
    IDT_DEVICE_SPECTRUM_PRO,
    IDT_DEVICE_MINISMART_II,
    IDT_DEVICE_UNIPAY,
    IDT_DEVICE_UNIPAY_I_V,
    IDT_DEVICE_UNIPAY_III,
    IDT_DEVICE_L100,
};

enum DEVICE_INTERFACE{
    DEVICE_INTERFACE_UNKNOWN=0,
    PROTOCOL_USBHID,
    PROTOCOL_USBBK,
    PROTOCOL_SEIRAL,
    PROTOCOL_BLUETOOTH,
};

enum DEVICE_PROTOCOL{
    DEVICE_PROTOCOL_UNKNOWN=0,
    PROTOCOL_NGA,
    PROTOCOL_IDG,
    PROTOCOL_IIP,
};

enum DEVICE_STATUS{
    DEVICE_DISCONNECT=0,
    DEVICE_CONNECTED,
};

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,
};

enum MSR_callBack_type
{
    MSR_callBack_type_ERR=1,
    MSR_callBack_type_RETURN_CODE,
    MSR_callBack_type_TIMEOUT
};

enum PIN_callBack_type
{
    PIN_callBack_type_ERR=1,
    PIN_callBack_type_RETURN_CODE,
    PIN_callBack_type_TIMEOUT
};

```

```

enum CAPTURE_ENCODE_TYPE
{
    CAPTURE_ENCODE_TYPE_ISOABA,
    CAPTURE_ENCODE_TYPE_AAMVA,
    CAPTURE_ENCODE_TYPE_Other,
    CAPTURE_ENCODE_TYPE_Raw,
    CAPTURE_ENCODE_TYPE_Jis_II,
    CAPTURE_ENCODE_TYPE_Jis_I,
    CAPTURE_ENCODE_TYPE_Jis_II_Security,
    CAPTURE_ENCODE_TYPE_Contactless_Visa_Kernel1,
    CAPTURE_ENCODE_TYPE_Contactless_MasterCard,
    CAPTURE_ENCODE_TYPE_Contactless_Visa_Kernel3,
    CAPTURE_ENCODE_TYPE_Contactless_AmericanExpress,
    CAPTURE_ENCODE_TYPE_Contactless_JCB,
    CAPTURE_ENCODE_TYPE_Contactless_Discover,
    CAPTURE_ENCODE_TYPE_Contactless_UnionPay,
    CAPTURE_ENCODE_TYPE_Contactless_Others,
    CAPTURE_ENCODE_TYPE_Manual_Entry_Enhanced_Mode,
    CAPTURE_ENCODE_TYPE_JisI_II
};

enum CAPTURE_ENCRYPT_TYPE
{
    CAPTURE_ENCRYPT_TYPE_TDES, CAPTURE_ENCRYPT_TYPE_AES, CAPTURE_ENCRYPT_TYPE_NONE
};

enum EMV_PIN_MODE
{
    EMV_PIN_MODE_CANCEL=0, EMV_PIN_MODE_ONLINE_DUKPT=1, EMV_PIN_MODE_ONLINE_MKSK=2, EMV_PIN_MODE_OFFLINE=3,
};

enum EMV_CALLBACK_TYPE
{
    EMV_CALLBACK_TYPE_LCD=1, EMV_CALLBACK_TYPE_PINPAD=2, EMV_CALLBACK_MSR=3,
    EMV_callBack_type_ERR,
    EMV_callBack_type_RETURN_CODE,
};

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

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_TRANSACTION_CANCELED = 0x0012,
    EMV_RESULT_CODE_SWIPE_NON_ICC = 0x11,
    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 = 0xFF,
    EMV_RESULT_CODE_FILE_ARG_INVALID = 0x1001,
    EMV_RESULT_CODE_FILE_OPEN_FAILED = 0x1002,
    EMV_RESULT_CODE_FILE_OPERATION_FAILED = 0x1003,
    EMV_RESULT_CODE_MEMORY_NOT_ENOUGH = 0x2001,
    EMV_RESULT_CODE_SMARTCARD_OK = 0x3001,
    EMV_RESULT_CODE_SMARTCARD_FAIL = 0x3002,
    EMV_RESULT_CODE_SMARTCARD_INIT_FAILED = 0x3003,
    EMV_RESULT_CODE_FALLBACK_SITUATION = 0x3004,
    EMV_RESULT_CODE_SMARTCARD_ABSENT = 0x3005,
    EMV_RESULT_CODE_SMARTCARD_TIMEOUT = 0x3006,
    EMV_RESULT_CODE_MSR_CARD_ERROR = 0x3007,
    EMV_RESULT_CODE_PARSING_TAGS_FAILED = 0x5001,
    EMV_RESULT_CODE_CARD_DATA_ELEMENT_DUPLICATE = 0x5002,
    EMV_RESULT_CODE_DATA_FORMAT_INCORRECT = 0x5003,
    EMV_RESULT_CODE_APP_NO_TERM = 0x5004,
    EMV_RESULT_CODE_APP_NO_MATCHING = 0x5005,
    EMV_RESULT_CODE_MANDATORY_OBJECT_MISSING = 0x5006,
    EMV_RESULT_CODE_APP_SELECTION_RETRY = 0x5007,
    EMV_RESULT_CODE_AMOUNT_ERROR_GET = 0x5008,
    EMV_RESULT_CODE_CARD_REJECTED = 0x5009,
    EMV_RESULT_CODE_AIP_NOT_RECEIVED = 0x5010,
};

```

```

    EMV_RESULT_CODE_AFL_NOT_RECEIVED = 0x5011,
    EMV_RESULT_CODE_AFL_LEN_OUT_OF_RANGE = 0x5012,
    EMV_RESULT_CODE_SFI_OUT_OF_RANGE = 0x5013,
    EMV_RESULT_CODE_AFL_INCORRECT = 0x5014,
    EMV_RESULT_CODE_EXP_DATE_INCORRECT = 0x5015,
    EMV_RESULT_CODE_EFF_DATE_INCORRECT = 0x5016,
    EMV_RESULT_CODE_ISS_COD_TBL_OUT_OF_RANGE = 0x5017,
    EMV_RESULT_CODE_CRYPTOGAM_TYPE_INCORRECT = 0x5018,
    EMV_RESULT_CODE_PSE_BY_CARD_NOT_SUPPORTED = 0x5019,
    EMV_RESULT_CODE_USER_LANGUAGE_SELECTED = 0x5020,
    EMV_RESULT_CODE_SERVICE_NOT_ALLOWED = 0x5021,
    EMV_RESULT_CODE_NO_TAG_FOUND = 0x5022,
    EMV_RESULT_CODE_CARD_BLOCKED = 0x5023,
    EMV_RESULT_CODE_LEN_INCORRECT = 0x5024,
    EMV_RESULT_CODE_CARD_COM_ERROR = 0x5025,
    EMV_RESULT_CODE_TSC_NOT_INCREASED = 0x5026,
    EMV_RESULT_CODE_HASH_INCORRECT = 0x5027,
    EMV_RESULT_CODE_ARC_NOT_PRESENCE = 0x5028,
    EMV_RESULT_CODE_ARC_INVALID = 0x5029,
    EMV_RESULT_CODE_COMM_NO_ONLINE = 0x5030,
    EMV_RESULT_CODE_TRAN_TYPE_INCORRECT = 0x5031,
    EMV_RESULT_CODE_APP_NO_SUPPORT = 0x5032,
    EMV_RESULT_CODE_APP_NOT_SELECT = 0x5033,
    EMV_RESULT_CODE_LANG_NOT_SELECT = 0x5034,
    EMV_RESULT_CODE_TERM_DATA_NOT_PRESENCE = 0x5035,
    EMV_RESULT_CODE_CVM_TYPE_UNKNOWN = 0x6001,
    EMV_RESULT_CODE_CVM_AIP_NOT_SUPPORTED = 0x6002,
    EMV_RESULT_CODE_CVM_TAG_8E_MISSING = 0x6003,
    EMV_RESULT_CODE_CVM_TAG_8E_FORMAT_ERROR = 0x6004,
    EMV_RESULT_CODE_CVM_CODE_IS_NOT_SUPPORTED = 0x6005,
    EMV_RESULT_CODE_CVM_COND_CODE_IS_NOT_SUPPORTED = 0x6006,
    EMV_RESULT_CODE_CVM_NO_MORE = 0x6007,
    EMV_RESULT_CODE_PIN_BYPASSED_BEFORE = 0x6008,
    EMV_RESULT_CODE_UNKONWN = 0xffff,
};

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
};

enum EVENT_NOTIFICATION_Types
{
    EVENT_NOTIFICATION_UNKNOWN,
    EVENT_NOTIFICATION_ICC_Card_Not_Seated,
    EVENT_NOTIFICATION_ICC_Card_Seated,
    EVENT_NOTIFICATION_MSR_Swipe_Card,
};

enum CTLS_APPLICATION
{
    CTLS_APPLICATION_NONE = 0,
    CTLS_APPLICATION_MASTERCARD = 1,
    CTLS_APPLICATION_VISA = 2,
    CTLS_APPLICATION_AMEX = 3,
    CTLS_APPLICATION_DISCOVER = 4,
    CTLS_APPLICATION_SPEEDPASS = 5,
    CTLS_APPLICATION_GIFT_CARD = 6,
    CTLS_APPLICATION_DINERS_CLUB = 7,
    CTLS_APPLICATION_EN_ROUTE = 8,
    CTLS_APPLICATION_JCB = 9,
    CTLS_APPLICATION_VIVO_DIAGNOSTIC = 10,
    CTLS_APPLICATION_HID = 11,
    CTLS_APPLICATION_MSR_SWIPE = 12,
    CTLS_APPLICATION_RESERVED = 13,
    CTLS_APPLICATION_DES_FIRE_TRACK_DATA = 14,
    CTLS_APPLICATION_DES_FIRE_RAW_DATA = 15,
    CTLS_APPLICATION_RBS = 17,
    CTLS_APPLICATION_VIVO_COMM = 20,
};

enum DeviceState
{
    TransactionData,
    Connected,
    ConnectionFailed,
    DataReceived,
    DataSent,
    Disconnected,
    SwipeCanceled,
};

```

```

        ToConnect,
        ToSwipe,
        ToTap,
        CommandTimeout,
        SwipeTimeout,
        DeviceTimeout,
        EMVCallback,
        TransactionCancelled,
        TransactionFailed,
        MSRDecodeError,
        DefaultDeviceTypeChange,
        Notification,
        PINpadKeypress,
        PINCancelled,
        PINTimeout
    };

enum RETURN_CODE
{
    RETURN_CODE_DO_SUCCESS = 0,
    RETURN_CODE_ERR_DISCONNECT,
    RETURN_CODE_ERR_CMD_RESPONSE,
    RETURN_CODE_ERR_TIMEDOUT,
    RETURN_CODE_ERR_INVALID_PARAMETER,
    RETURN_CODE_SDK_BUSY_MSR,
    RETURN_CODE_SDK_BUSY_PINPAD,
    RETURN_CODE_SDK_BUSY_CTLs,
    RETURN_CODE_SDK_BUSY_EMV,
    RETURN_CODE_ERR_OTHER,
    RETURN_CODE_FAILED,
    RETURN_CODE_NOT_ATTACHED,
    RETURN_CODE_MONO_AUDIO,
    RETURN_CODE_CONNECTED,
    RETURN_CODE_LOW_VOLUME,
    RETURN_CODE_CANCELED,
    RETURN_CODE_INVALID_STR,
    RETURN_CODE_NO_FILE,
    RETURN_CODE_INVALID_FILE,
    RETURN_CODE_HOST_UNREACHABLE,
    RETURN_CODE_RKI_FAILURE,
    RETURN_CODE_MISSING_DLL,
    RETURN_CODE_ERR_BUF_NOT_ENOUGH,
    RETURN_CODE_P1_INCORRECT_FRAME_TAG = 0X0C01,
    RETURN_CODE_P1_INCORRECT_FRAME_TYPE = 0X0C02,
    RETURN_CODE_P1_UNKNOWN_FRAME_TYPE = 0X0C03,
    RETURN_CODE_P1_UNKNOWN_COMMAND = 0X0C04,
    RETURN_CODE_P1_UNKNOWN_SUB_COMMAND = 0X0C05,
    RETURN_CODE_P1_CRC_ERROR = 0X0C06,
    RETURN_CODE_P1_FAILED = 0X0C07,
    RETURN_CODE_P1_TIMEOUT = 0X0C08,
    RETURN_CODE_P1_INCORRECT_PARAMETER = 0X0C0A,
    RETURN_CODE_P1_COMMAND_NOT_SUPPORTED = 0X0C0B,
    RETURN_CODE_P1_SUB_COMMAND_NOT_SUPPORTED = 0X0C0C,
    RETURN_CODE_P1_STATUS_ABORT_COMMAND = 0X0C0D,
    RETURN_CODE_P1_COMMAND_NOT_ALLOWED = 0X0C0F,
    RETURN_CODE_P2_ = 0X0D01,
    RETURN_CODE_P2_UNKNOWN_COMMAND = 0X0D02,
    RETURN_CODE_P2_UNKNOWN_SUB_COMMAND = 0X0D03,
    RETURN_CODE_P2_CRC_ERROR = 0X0D04,
    RETURN_CODE_P2_INCORRECT_PARAMETER = 0X0D05,
    RETURN_CODE_P2_PARAMETER_NOT_SUPPORTED = 0X0D06,
    RETURN_CODE_P2_MAL_FORMATTED_DATA = 0X0D07,
    RETURN_CODE_P2_TIMEOUT = 0X0D08,
    RETURN_CODE_P2_FAILED = 0X0D0A,
    RETURN_CODE_P2_COMMAND_NOT_ALLOWED = 0X0D0B,
    RETURN_CODE_P2_SUB_COMMAND_NOT_ALLOWED = 0X0D0C,
    RETURN_CODE_P2_BUFFER_OVERFLOW = 0X0D0D,
    RETURN_CODE_P2_USER_INTERFACE_EVENT = 0X0D0E,
    RETURN_CODE_P2_COMM_TYPE_NOT_SUPPORTED = 0X0D11,
    RETURN_CODE_P2_SECURE_INTERFACE_NOT_FUNCTIONAL = 0X0D12,
    RETURN_CODE_P2_DATA_FIELD_NOT_MOD_8 = 0X0D13,
    RETURN_CODE_P2_PADDING_UNEXPECTED = 0X0D14,
    RETURN_CODE_P2_KEY_TYPE_INVALID = 0X0D15,
    RETURN_CODE_P2_COULD_NOT_RETRIEVE_KEY = 0X0D16,
    RETURN_CODE_P2_HASH_CODE_ERROR = 0X0D17,
    RETURN_CODE_P2_COUNDTOT_STORE_KEY = 0X0D18,
    RETURN_CODE_P2_FRAME_TOO_LARGE = 0X0D19,
    RETURN_CODE_P2_RESEND_INITSECURECOMM_COMMAND = 0X0D1A,
    RETURN_CODE_P2_EEPROM_NOT_INITIALIZED = 0X0D1B,
    RETURN_CODE_P2_APDU_ENCODING_ERROR = 0X0D1C,
    RETURN_CODE_P2_SAM_COMM_ERROR = 0X0D20,
    RETURN_CODE_P2_SEQUENCE_COUNTER_ERROR = 0X0D21,
    RETURN_CODE_P2_IMPROPER_BITMAP = 0X0D22,
    RETURN_CODE_P2_REQUEST_ONLINE_AUTHORIZATION = 0X0D23,
    RETURN_CODE_P2_RAW_DATA_READ_SUCCESSFUL = 0X0D24,
    RETURN_CODE_P2_MESSAGE_INDEX_NOT_AVAILABLE = 0X0D25,

```

```

RETURN_CODE_P2_VERSION_INFORMATION_MISMATCH = 0X0D26,
RETURN_CODE_P2_INCORRECT_MESSAGE_INDEX = 0X0D27,
RETURN_CODE_P2_NEXT_MESSAGE_NOT_RECEIVED = 0X0D28,
RETURN_CODE_P2_ILM_LANGUAGE_NOT_AVAILABLE = 0X0D29,
RETURN_CODE_P2_OTHER_LANGUAGE_NOT_SUPPORTED = 0X0D2A,
RETURN_CODE_UNKNOWN_ERROR_FROM_SAM = 0X0D41,
RETURN_CODE_INVALID_DATA_DETECTED_BY_SAM = 0X0D42,
RETURN_CODE_INCOMPLETE_DATA_DETECTED_BY_SAM = 0X0D43,
RETURN_CODE_RESERVED = 0X0D44,
RETURN_CODE_INVALID_KEY_HASH_ALGORITHM = 0X0D45,
RETURN_CODE_INVALID_KEY_ENCRYPTION_ALGORITHM = 0X0D46,
RETURN_CODE_INVALID_MODULUS_LENGTH = 0X0D47,
RETURN_CODE_INVALID_EXPONENT = 0X0D48,
RETURN_CODE_KEY_ALREADY_EXISTS = 0X0D49,
RETURN_CODE_NO_SPACE_FOR_NEW_RID = 0X0D4A,
RETURN_CODE_KEY_NOT_FOUND = 0X0D4B,
RETURN_CODE_CRYPTOTIMEOUT = 0X0D4C,
RETURN_CODE_CRYPTOCOMMUNICATION_ERROR = 0X0D4D,
RETURN_CODE_P2_KEY_MANAGER_ERROR_4E = 0X0D4E,
RETURN_CODE_ALL_KEY_SLOTS_FULL = 0X0D4F,
RETURN_CODE_P2_AUTO_SWITCH_OK = 0X0D50,
RETURN_CODE_P2_AUTO_SWITCH_FAILED = 0X0D51,
RETURN_CODE_P2_DATA_DOES_NOT_EXIST = 0X0D60,
RETURN_CODE_P2_DATA_FULL = 0X0D61,
RETURN_CODE_P2_WRITE_FLASH_ERROR = 0X0D62,
RETURN_CODE_P2_OK_AND_HAVE_NEXT_COMMAND = 0X0D63,
RETURN_CODE_P2_ACCOUNT_DUKPT_KEY_NOT_EXIST = 0X0D90,
RETURN_CODE_P2_ACCOUNT_DUKPT_KEY_KSN_EXHAUSTED = 0X0D91,

RETURN_CODE_UNKNOWN_COMMAND = 0X9031, // Unknown command
RETURN_CODE_WRONG_PARAMETER = 0X9032, // Wrong parameter (such as the length of the command is
    incorrect)

RETURN_CODE_WAIT = 0X9038,
RETURN_CODE_BUSY = 0X9039,
RETURN_CODE_RETRIES_OVER_LIMIT = 0X903A,
RETURN_CODE_TIMEOUT = 0X8100,

RETURN_CODE_INVALID_MAN_SYSTEM_DATA = 0X9040,
RETURN_CODE_NOT_AUTHENTICATED = 0X9041,
RETURN_CODE_INVALID_MASTER_DUKPT_KEY = 0X9042,
RETURN_CODE_INVALID_MAC_KEY = 0X9043,
RETURN_CODE_RESERVED_FOR_FUTURE_USE = 0X9044,
RETURN_CODE_RESERVED_FOR_FUTURE_USE_2 = 0X9045,
RETURN_CODE_INVALID_DATA_DUKPT_KEY = 0X9046,
RETURN_CODE_INVALID_PIN_PAIRING_DUKPT_KEY = 0X9047,
RETURN_CODE_INVALID_DATA_PAIRING_DUKPT_KEY = 0X9048,
RETURN_CODE_NO_NONCE_GENERATED = 0X9049,
RETURN_CODE_NO_GUID_AVAILABLE = 0X9949,
RETURN_CODE_NO_MAC_CALCULATION = 0X9950,

RETURN_CODE_NOT_READY = 0X904A,
RETURN_CODE_MSR_DATA_FAILED = 0X904B,

RETURN_CODE_INVALID_CERTIFICATE = 0X9050,
RETURN_CODE_DUPLICATE_KEY_DETECTED = 0X9051,
RETURN_CODE_AT_CHECKS_FAILED = 0X9052,
RETURN_CODE_TR34_CHECKS_FAILED = 0X9053,
RETURN_CODE_TR31_CHECKS_FAILED = 0X9054,
RETURN_CODE_AMC_CHECKS_FAILED = 0X9055,
RETURN_CODE_FIRMWARE_DOWNLOAD_FAILED = 0X9056,

RETURN_CODE_LOG_IS_FULL = 0X9060,
RETURN_CODE_REMOVAL_SENSOR_UNENGAGED = 0X9061,
RETURN_CODE_HARDWARE_PROBLEM = 0X9062,

RETURN_CODE_ICC_COMMUNICATION_TIMEOUT = 0X9070,
RETURN_CODE_IFF_DATA_ERROR = 0X9071,
RETURN_CODE_SMART_CARD_NOT_POWERED_UP = 0X9072,
RETURN_CODE_NO_AID = 0xF200,
RETURN_CODE_NO_TERMINAL_DATA = 0xF201,
RETURN_CODE_WRONG_TLV_FORMAT = 0xF202,
RETURN_CODE_AID_LIST_FULL = 0xF203,
RETURN_CODE_NO_CA_KEY = 0xF204,
RETURN_CODE_NO_CA_KEY_RID = 0xF205,
RETURN_CODE_NO_CA_KEY_INDEX = 0xF206,
RETURN_CODE_CA_KEY_LIST_FULL = 0xF207,
RETURN_CODE_CA_KEY_HASH_ERROR = 0xF208,
RETURN_CODE_COMMAND_FORMAT_ERROR = 0xF209,
RETURN_CODE_UNEXPECTED_COMMAND = 0xF20A,
RETURN_CODE_NO_CRL = 0xF20B,
RETURN_CODE_CRL_LIST_FULL = 0xF20C,
RETURN_CODE_MISSING_REQUIRED_PARAMETERS = 0xF20D,
RETURN_CODE_CA_INCORRECT_HASH_ALGORITHM = 0xF20E,

RETURN_CODE_EMV_AUTHORIZATION_ACCEPTED = 0x0E00,

```

```

RETURN_CODE_EMV_AUTHORIZATION_UNABLE_TO_GO_ONLINE = 0x0E01,
RETURN_CODE_EMV_AUTHORIZATION_TECHNICAL_ISSUE = 0x0E02,
RETURN_CODE_EMV_AUTHORIZATION_DECLINED = 0x0E03,
RETURN_CODE_EMV_AUTHORIZATION_ISSUER_REFERRAL = 0x0E04,
RETURN_CODE_EMV_APPROVED = 0x0F00,
RETURN_CODE_EMV_DECLINED = 0x0F01,
RETURN_CODE_EMV_GO_ONLINE = 0x0F02,
RETURN_CODE_EMV_FAILED = 0x0F03,
RETURN_CODE_EMV_SYSTEM_ERROR = 0x0F05,
RETURN_CODE_EMV_NOT_ACCEPTED = 0x0F07,
RETURN_CODE_EMV_FALLBACK = 0x0F0A,
RETURN_CODE_EMV_CANCEL = 0x0F0C,
RETURN_CODE_EMV_TIMEOUT = 0x0F0D,
RETURN_CODE_EMV_OTHER_ERROR = 0x0F0F,
RETURN_CODE_EMV_OFFLINE_APPROVED = 0x0F10,
RETURN_CODE_EMV_OFFLINE_DECLINED = 0x0F11,
RETURN_CODE_EMV_NEW_SELECTION = 0x0F21,
RETURN_CODE_EMV_NO_AVAILABLE_APPS = 0x0F22,
RETURN_CODE_EMV_NO_TERMINAL_FILE = 0x0F23,
RETURN_CODE_EMV_NO_CAPK_FILE = 0x0F24,
RETURN_CODE_EMV_NO_CRL_ENTRY = 0x0F25,
RETURN_CODE_BLOCKING_DISABLED = 0x0FFE,
RETURN_CODE_CM100_WITHOUT_ERROR = 0xBBE0,
RETURN_CODE_CM100_PARAMETER = 0xBBE1,
RETURN_CODE_CM100_LOWOUTBUFFER = 0xBBE2,
RETURN_CODE_CM100_CARD_NOT_FOUND = 0xBBE3,
RETURN_CODE_CM100_COLLISION_CARD_EXIST = 0xBBE4,
RETURN_CODE_CM100_TOO_MANY_CARDS_EXIST = 0xBBE5,
RETURN_CODE_CM100_SAVED_DATA_NOT_EXIST = 0xBBE6,
RETURN_CODE_CM100_NO_DATA_AVAILABLE = 0xBBE8,
RETURN_CODE_CM100_INVALID_CID_RETURNED = 0xBBE9,
RETURN_CODE_CM100_INVALID_CARD_EXIST = 0xBBEA,
RETURN_CODE_CM100_COMMAND_UNSUPPORTED = 0xBBEC,
RETURN_CODE_CM100_COMMAND_PROCESS = 0xBBED,
RETURN_CODE_CM100_INVALID_COMMAND = 0xBBEE,
RETURN_CODE_COMMAND_UNAVAILABLE = 0xFFFF
};

```


Chapter 8

EMV Callback

During an EMV transaction, ID TECH devices without a built-in LCD display return LCD Display messages as an EMV Callback.

```
void EMV_callBack(int device_type, int device_state, unsigned char * data, int dataLen, IDTTransactionData*
    cardData, EMV_Callback* emvCallback, int transactionResultCode)
```

There is an **DeviceState** enum in the SDK's EMV Callback that is returned for **device_state**. When this **DeviceState** is received as **EMVCallback**, it returns an **EMV_Callback()** class pointer. **EMV_Callback.callbackType** specifies the type of callback: **EMV_CALLBACK_TYPE_LCD**, **EMV_CALLBACK_TYPE_PINPAD**, or **EMV_CALLBACK_TYPE_MSR**. The device only uses the **EMV_CALLBACK_TYPE_LCD**.

The callback type for LCD display messages is **EMV_CALLBACK_TYPE_LCD**. To determine the type of LCD message, 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

The **LCD_DISPLAY_MODE_MESSAGE** and **LCD_DISPLAY_MODE_CLEAR_SCREEN** 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 the EMV transaction pauses until a response is sent to [emv_callbackResponseLCD\(\)](#).

The message to display is **byte[] lcd_messages**. This contains either a message string or a message ID according to the LCD Foreign Language Mapping Table ([Foreign Language Mapping Table](#)).

Chapter 9

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	TLV Exclusion 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 Status Information
DFEE27	MSR Control
DFEF1A	SmartTap Delimiter
DFEF1E	Encrypted Sensitive Tags
DFEF1F	Auto Authenticate

Tag	Description
DFEF20	MAC option in reponse data
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 Verification KSN for SRED
DFEF43	Local TZ/DST information.
DFEF44	Combination Options
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

Tag	Description
DFE↔ F5B	Mask for Tag5A
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

Tag	Description
FFE6	AID Disabled
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
Byte 2								
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
Byte 3								
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

Byte 1								
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
Byte 2								
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
Byte 3								
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
Byte 4								
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
Byte 5								
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	x	1	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 10

File Index

10.1 File List

Here is a list of all documented files with brief descriptions:

Source_C/libIDT_Augusta.h	
Augusta API	44
Source_C/libIDT_Device.h	
Windows C++ API	103
Source_C/libIDT_KioskIII.h	
KioskIII API	297
Source_C/libIDT_L100.h	
L100 API	325
Source_C/libIDT_L80.h	
L80 API	358
Source_C/libIDT_MiniSmartII.h	
MiniSmartII API	391
Source_C/libIDT_NEO2.h	
NEO2 API	442
Source_C/libIDT_PipReader.h	
PipReader API	566
Source_C/libIDT_SpectrumPro.h	
SpectrumPro API	593
Source_C/libIDT_SREDKey2.h	
SREDKey2 API	641
Source_C/libIDT_UniPayI_V.h	
UniPay 1.5 API	662
Source_C/libIDT_Vendi.h	
Vendi API	691
Source_C/libIDT_VP3300_AJ.h	
VP3300 AJ API	718
Source_C/libIDT_VP3300_BT.h	
VP3300 BT API	770
Source_C/libIDT_VP3300_COM.h	
VP3300 COM API	822
Source_C/libIDT_VP3300_USB.h	
VP3300 USB API	874
Source_C/libIDT_VP8800.h	
VP8800 API	925

Chapter 11

File Documentation

11.1 Source_C/libIDT_Augusta.h File Reference

Augusta API.

```
#include "IDTDef.h"
```

Macros

- `#define IN`
- `#define OUT`
- `#define IN_OUT`

Typedefs

- `typedef void(* pMessageHotplug) (int, int)`
- `typedef void(* pSendDataLog) (BYTE *, int)`
- `typedef void(* pReadDataLog) (BYTE *, int)`
- `typedef void(* pEMV_callBack) (int, int, BYTE *, int, IDTTransactionData *, EMV_Callback *, int)`
- `typedef void(* pMSR_callBack) (int, IDTMSRData)`
- `typedef void(* pMSR_callBackp) (int, IDTMSRData *)`
- `typedef void(* pPIN_callBack) (int, IDTPINData *)`
- `typedef void(* pCMR_callBack) (int, IDTCMRData *)`
- `typedef void(* pCSFS_callBack) (BYTE status)`
- `typedef void(* pFW_callBack) (int, int, int, int, int)`
- `typedef void(* ftpComm_callBack) (int, int, int)`
- `typedef void(* httpComm_callBack) (BYTE *, int)`
- `typedef void(* v4Comm_callBack) (BYTE, BYTE, BYTE *, int)`

Functions

- `void registerHotplugCallBk (pMessageHotplug pMsgHotplug)`
- `void registerLogCallBk (pSendDataLog pFSend, pReadDataLog pFRead)`
- `void emv_registerCallBk (pEMV_callBack pEMVf)`
- `void msr_registerCallBk (pMSR_callBack pMSRf)`
- `void msr_registerCallBkp (pMSR_callBackp pMSRf)`
- `void pin_registerCallBk (pPIN_callBack pPINf)`
- `void device_registerCameraCallBk (pCMR_callBack pCMRf)`

- void [device_registerCardStatusFrontSwitchCallBk](#) (pCSFS_callBack pCSFSf)
- void [device_registerFWCallBk](#) (pFW_callBack pFWf)
- char * [SDK_Version](#) ()
- int [setAbsoluteLibraryPath](#) (const char *absoluteLibraryPath)
- int [device_init](#) ()
- int [device_setCurrentDevice](#) (int deviceType)
- int [device_close](#) ()
- void [device_getResponseCodeString](#) (IN int returnCode, OUT char *despcriton)
- int [device_isConnected](#) ()
- int [device_isAttached](#) (int deviceType)
- int [device_getFirmwareVersion](#) (OUT char *firmwareVersion)
- int [device_getFirmwareVersion_Len](#) (OUT char *firmwareVersion, IN_OUT int *firmwareVersionLen)
- int [device_getCurrentDeviceType](#) ()
- int [device_SendDataCommand](#) (IN BYTE *cmd, IN int cmdLen, IN BYTE *data, IN int dataLen, OUT BYTE *response, IN_OUT int *respLen)
- int [device_updateFirmware](#) (IN BYTE *firmwareData, IN int firmwareDataLen, IN char *firmwareName, IN int encryptionType, IN BYTE *keyBlob, IN int keyBlobLen)
- int [device_rebootDevice](#) ()
- int [device_controlLED](#) (byte indexLED, byte control, int intervalOn, int intervalOff)
- int [device_controlLED_ICC](#) (int controlMode, int interval)
- int [device_controlLED_MSR](#) (byte control, int intervalOn, int intervalOff)
- int [device_controlBeep](#) (int index, int frequency, int duration)
- int [device_getDRS](#) (BYTE *codeDRS, int *codeDRSLen)
- int [device_getKeyStatus](#) (int *newFormat, BYTE *status, int *statusLen)
- int [device_getSDKWaitTime](#) ()
- void [device_setSDKWaitTime](#) (int waitTime)
- int [device_getThreadStackSize](#) ()
- void [device_setThreadStackSize](#) (int threadSize)
- int [config_getModelNumber](#) (OUT char *sNumber)
- int [config_getModelNumber_Len](#) (OUT char *sNumber, IN_OUT int *sNumberLen)
- int [config_getSerialNumber](#) (OUT char *sNumber)
- int [config_getSerialNumber_Len](#) (OUT char *sNumber, IN_OUT int *sNumberLen)
- int [config_setLEDController](#) (IN int firmwareControlMSRLED, IN int firmwareControlICCLED)
- int [config_getLEDController](#) (OUT int *firmwareControlMSRLED, OUT int *firmwareControlICCLED)
- int [config_setBeeperController](#) (IN int firmwareControlBeeper)
- int [config_getBeeperController](#) (OUT int *firmwareControlBeeper)
- int [config_setEncryptionControl](#) (IN int msr, IN int icc)
- int [config_getEncryptionControl](#) (OUT int *msr, OUT int *icc)
- int [icc_enable](#) (IN int withNotification)
- int [icc_disable](#) ()
- int [icc_powerOnICC](#) (OUT BYTE *ATR, IN_OUT int *inLen)
- int [icc_powerOffICC](#) ()
- int [icc_exchangeAPDU](#) (IN BYTE *c_APDU, IN int cLen, OUT BYTE *reData, IN_OUT int *reLen)
- int [icc_exchangeEncryptedAPDU](#) (IN BYTE *c_APDU, IN int cLen, OUT BYTE *reData, IN_OUT int *reLen)
- int [icc_getAPDU_KSN](#) (OUT BYTE *KSN, IN_OUT int *inLen)
- int [icc_getFunctionStatus](#) (OUT int *enabled, OUT int *withNotification)
- int [icc_getICCReaderStatus](#) (OUT BYTE *status)
- int [icc_getKeyFormatForICCDUKPT](#) (OUT BYTE *format)
- int [icc_getKeyTypeForICCDUKPT](#) (OUT BYTE *type)
- int [emv_getEMVKernelVersion](#) (OUT char *version)
- int [emv_getEMVKernelVersion_Len](#) (OUT char *version, IN_OUT int *versionLen)
- int [emv_getEMVKernelCheckValue](#) (OUT BYTE *checkValue, IN_OUT int *checkValueLen)
- int [emv_getEMVConfigurationCheckValue](#) (OUT BYTE *checkValue, IN_OUT int *checkValueLen)
- void [emv_allowFallback](#) (IN int allow)
- void [emv_setAutoAuthenticateTransaction](#) (IN int authenticate)

- void `emv_setAutoCompleteTransaction` (IN int complete)
- int `emv_getAutoAuthenticateTransaction` ()
- int `emv_getAutoCompleteTransaction` ()
- int `emv_startTransaction` (IN double amount, IN double amtOther, IN int exponent, IN int type, IN int timeout, IN BYTE *tags, IN int tagsLen, IN int forceOnline)
- int `emv_activateTransaction` (IN int timeout, IN BYTE *tags, IN int tagsLen, IN int forceOnline)
- void `device_setTransactionExponent` (int exponent)
- int `device_startTransaction` (IN double amount, IN double amtOther, IN int type, IN const int _timeout, IN BYTE *tags, IN int tagsLen)
- int `emv_authenticateTransaction` (IN BYTE *updatedTLV, IN int updatedTLVLen)
- int `emv_authenticateTransactionWithTimeout` (IN int timeout, IN BYTE *updatedTLV, IN int updatedTLVLen)
- int `emv_completeTransaction` (IN int commError, IN BYTE *authCode, IN int authCodeLen, IN BYTE *iad, IN int iadLen, IN BYTE *tlvScripts, IN int tlvScriptsLen, IN BYTE *tlv, IN int tlvLen)
- int `emv_cancelTransaction` ()
- int `device_cancelTransaction` ()
- int `emv_retrieveTransactionResult` (IN BYTE *tags, IN int tagsLen, IDTTransactionData *cardData)
- int `emv_callbackResponseLCD` (IN int type, byte selection)
- int `emv_callbackResponseMSR` (IN BYTE *MSR, IN_OUT int MSRLen)
- int `emv_retrieveApplicationData` (IN BYTE *AID, IN int AIDLen, OUT BYTE *tlv, IN_OUT int *tlvLen)
- int `emv_setApplicationData` (IN BYTE *name, IN int nameLen, IN BYTE *tlv, IN int tlvLen)
- int `emv_removeApplicationData` (IN BYTE *AID, IN int AIDLen)
- int `emv_removeAllApplicationData` ()
- int `emv_retrieveAIDList` (OUT BYTE *AIDList, IN_OUT int *AIDListLen)
- int `emv_retrieveTerminalData` (OUT BYTE *tlv, IN_OUT int *tlvLen)
- int `emv_setTerminalData` (IN BYTE *tlv, IN int tlvLen)
- int `emv_removeTerminalData` ()
- int `emv_retrieveCAPK` (IN BYTE *capk, IN int capkLen, OUT BYTE *key, IN_OUT int *keyLen)
- int `emv_setCAPK` (IN BYTE *capk, IN int capkLen)
- int `emv_removeCAPK` (IN BYTE *capk, IN int capkLen)
- int `emv_removeAllCAPK` ()
- int `emv_retrieveCAPKList` (OUT BYTE *keys, IN_OUT int *keysLen)
- int `emv_retrieveTerminalID` (OUT char *terminalID)
- int `emv_retrieveTerminalID_Len` (OUT char *terminalID, IN_OUT int *terminalIDLen)
- int `emv_setTerminalID` (IN char *terminalID)
- int `emv_retrieveCRL` (OUT BYTE *list, IN_OUT int *lssLen)
- int `emv_setCRL` (IN BYTE *list, IN int lsLen)
- int `emv_removeCRL` (IN BYTE *list, IN int lsLen)
- int `emv_removeAllCRL` ()
- int `msr_getMSRData` (OUT BYTE *reData, IN_OUT int *reLen)
- int `msr_cancelMSRSwipe` ()
- int `msr_startMSRSwipe` (IN int _timeout)
- void `parseMSRData` (IN BYTE *resData, IN int resLen, IN_OUT IDTMSRData *cardData)
- int `msr_getKeyFormatForICCDUKPT` (OUT BYTE *format)
- int `msr_getKeyTypeForICCDUKPT` (OUT BYTE *type)
- int `msr_setKeyFormatForICCDUKPT` (IN BYTE format)
- int `msr_setKeyTypeForICCDUKPT` (IN BYTE type)
- int `msr_captureMode` (IN int isBufferMode, IN int withNotification)
- int `msr_setSetting` (IN BYTE setting, IN BYTE *val, IN int valLen)
- int `msr_getSetting` (IN byte setting, OUT BYTE *value, IN_OUT int *valueLen)
- int `msr_setSwipeForcedEncryptionOption` (IN int track1, IN int track2, IN int track3, IN int track3card0)
- int `msr_getSwipeForcedEncryptionOption` (OUT BYTE *option)
- int `msr_setSwipeMaskOption` (IN int track1, IN int track2, IN int track3)
- int `msr_getSwipeMaskOption` (OUT BYTE *option)
- int `msr_setExpirationMask` (IN int mask)
- int `msr_getExpirationMask` (OUT BYTE *value)

- int [msr_setClearPANID](#) (IN BYTE val)
- int [msr_getClearPANID](#) (OUT BYTE *value)
- int [msr_disable](#) ()
- int [pin_cancelPINEntry](#) ()

11.1.1 Detailed Description

Augusta API.

Augusta Global API methods.

11.1.2 Macro Definition Documentation

11.1.2.1 IN

```
#define IN
```

INPUT parameter.

11.1.2.2 IN_OUT

```
#define IN_OUT
```

INPUT / OUTPUT PARAMETER.

11.1.2.3 OUT

```
#define OUT
```

OUTPUT parameter.

11.1.3 Typedef Documentation

11.1.3.1 ftpComm_callBack

```
typedef void(* ftpComm_callBack)(int, int, int)
```

Define the comm callback function to get FTP file transfer status

It should be passed as a parameter in a FTP request, Signature (int, int, int) = response code, current block, total blocks
 RESPONSE CODES: 100 = FILE DOWNLOAD STARTED 101 = FILE BLOCK XX OF XX RECEIVED 102 = FILE DOWNLOAD COMPLETED 103 = FILE DOWNLOAD TERMINATED PREMATURELY

11.1.3.2 httpComm_callBack

```
typedef void(* httpComm_callBack) (BYTE *, int)
```

Define the comm callback function to get the async url data

It should be registered using the comm_registerHTTPCallback

11.1.3.3 pCMR_callback

```
typedef void(* pCMR_callback) (int, IDTCMRData *)
```

Define the camera callback function to get the image data

It should be registered using the device_registerCameraCallBk,

11.1.3.4 pCSFS_callback

```
typedef void(* pCSFS_callback) (BYTE status)
```

Define the card status and front switch callback function to get card and front switch status

It should be registered using the device_registerCardStatusFrontSwitchCallBk,

11.1.3.5 pEMV_callback

```
typedef void(* pEMV_callback) (int, int, BYTE *, int, IDTTransactionData *, EMV_Callback *,
int)
```

Define the EMV callback function to get the transaction message/data/result.

It should be registered using the emv_registerCallBk,

11.1.3.6 pFW_callback

```
typedef void(* pFW_callback) (int, int, int, int, int)
```

Define the firmware update callback function to get the status of firmware update

It should be registered using the device_registerFWCallBk,

11.1.3.7 pMessageHotplug

```
typedef void(* pMessageHotplug) (int, int)
```

Define the USB hot-plug callback function to monitor the info when plug in/out the reader.

It should be registered using the registerHotplugCallBk, The first integer parameter is device type, and the second integer parameter is either 0: Device Plugged Out or 1: Device Plugged In

11.1.3.8 pMSR_callback

```
typedef void(* pMSR_callback) (int, IDTMSRData)
```

Define the MSR callback function to get the MSR card data

It should be registered using the msr_registerCallBk, this callback function is for backward compatibility

11.1.3.9 pMSR_callbackp

```
typedef void(* pMSR_callbackp) (int, IDTMSRData *)
```

Define the MSR callback function to get pointer to the MSR card data

It should be registered using the msr_registerCallBk, this callback function is recommended instead of pMSR_callback

11.1.3.10 pPIN_callback

```
typedef void(* pPIN_callback) (int, IDTPINData *)
```

Define the PINPad callback function to get the input PIN Pad data
It should be registered using the pin_registerCallBk,

11.1.3.11 pReadDataLog

```
typedef void(* pReadDataLog) (BYTE *, int)
```

Define the read response callback function to monitor the reading response from the reader.
It should be registered using the registerLogCallBk,

11.1.3.12 pSendDataLog

```
typedef void(* pSendDataLog) (BYTE *, int)
```

Define the send command callback function to monitor the sending command into the reader.
It should be registered using the registerLogCallBk,

11.1.3.13 v4Comm_callBack

```
typedef void(* v4Comm_callBack) (BYTE, BYTE, BYTE *, int)
```

Define the comm callback function to receive the V4 Protocol packets received by the device from an external source (IP/USB/RS-232) It should be registered using the comm_registerV4Callback, Data callback will contain command, sub-command, and data from V4 packet

11.1.4 Function Documentation

11.1.4.1 config_getBeeperController()

```
int config_getBeeperController (
    OUT int * firmwareControlBeeper )
```

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

Parameters

<i>firmwareControlBeeper</i>	1 means firmware control the beeper, 0 means software control beeper.
------------------------------	---

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString

11.1.4.2 config_getEncryptionControl()

```
int config_getEncryptionControl (
    OUT int * msr,
    OUT int * icc )
```

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"> • 1: enabled MSR with Encryption, • 0: disabled MSR with Encryption,
<i>icc</i>	<ul style="list-style-type: none"> • 1: enabled ICC with Encryption, • 0: disabled ICC with Encryption,

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString

11.1.4.3 config_getLEDController()

```
int config_getLEDController (
    OUT int * firmwareControlMSRLED,
    OUT int * firmwareControlICCLED )
```

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

Parameters

<i>firmwareControlMSRLED</i>	<ul style="list-style-type: none"> • 1: firmware control the MSR LED • 0: software control the MSR LED
<i>firmwareControlICCLED</i>	<ul style="list-style-type: none"> • 1: firmware control the ICC LED • 0: software control the ICC LED

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString

11.1.4.4 config_getModelNumber()

```
int config_getModelNumber (
    OUT char * sNumber )
```

Polls device for Model Number

Parameters

<i>sNumber</i>	Returns Model Number; needs to have at least 64 bytes of memory
----------------	---

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString

11.1.4.5 config_getModelNumber_Len()

```
int config_getModelNumber_Len (
    OUT char * sNumber,
    IN_OUT int * sNumberLen )
```

DEPRECATED : please use [config_getModelNumber_Len\(OUT char* sNumber, IN_OUT int *sNumberLen\)](#)

Polls device for Model Number

Parameters

<i>sNumber</i>	Returns Model Number
<i>sNumber</i>	length of Model Number

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString

11.1.4.6 config_getSerialNumber()

```
int config_getSerialNumber (
    OUT char * sNumber )
```

DEPRECATED : please use [config_getSerialNumber_Len\(OUT char* sNumber, IN_OUT int *sNumberLen\)](#)

Polls device for Serial Number

Parameters

<i>sNumber</i>	Returns Serial Number; needs to have at least 64 bytes of memory
----------------	--

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString

11.1.4.7 config_getSerialNumber_Len()

```
int config_getSerialNumber_Len (
```

```
OUT char * sNumber,
IN_OUT int * sNumberLen )
```

Polls device for Serial Number

Parameters

<i>sNumber</i>	Returns Serial Number
<i>sNumberLen</i>	Length of Serial Number

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString

11.1.4.8 config_setBeeperController()

```
int config_setBeeperController (
    IN int firmwareControlBeeper )
```

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

Parameters

<i>firmwareControlBeeper</i>	1 means firmware control the beeper, 0 means software control beeper.
------------------------------	---

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString

11.1.4.9 config_setEncryptionControl()

```
int config_setEncryptionControl (
    IN int msr,
    IN int icc )
```

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"> • 1: enable MSR with Encryption, • 0: disable MSR with Encryption,
------------	---

Parameters

<i>icc</i>	<ul style="list-style-type: none"> • 1: enable ICC with Encryption, • 0: disable ICC with Encryption,
------------	---

Returns

RETURN_CODE: Values can be parsed with `device_getResponseCodeString`

11.1.4.10 config_setLEDController()

```
int config_setLEDController (
    IN int firmwareControlMSRLED,
    IN int firmwareControlICCLED )
```

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"> • 1: firmware control the MSR LED • 0: software control the MSR LED
<i>firmwareControlICCLED</i>	<ul style="list-style-type: none"> • 1: firmware control the ICC LED • 0: software control the ICC LED

Returns

RETURN_CODE: Values can be parsed with `device_getResponseCodeString`

11.1.4.11 device_cancelTransaction()

```
int device_cancelTransaction ( )
```

Cancel Transaction request.

Returns

RETURN_CODE: Values can be parsed with [device_getResponseCodeString\(\)](#)

11.1.4.12 device_close()

```
int device_close ( )
```

Close the device

Returns

RETURN_CODE: 0: success, 0x0A: failed

11.1.4.13 device_controlBeep()

```
int device_controlBeep (
    int index,
    int frequency,
    int duration )
```

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

11.1.4.14 device_controlLED()

```
int device_controlLED (
    byte indexLED,
    byte control,
    int intervalOn,
    int intervalOff )
```

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

11.1.4.15 device_controlLED_ICC()

```
int device_controlLED_ICC (  
    int controlMode,  
    int interval )
```

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

11.1.4.16 device_controlLED_MSR()

```
int device_controlLED_MSR (  
    byte control,  
    int intervalOn,  
    int intervalOff )
```

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

11.1.4.17 device_getCurrentDeviceType()

```
int device_getCurrentDeviceType ( )
```

Get current active device type

Returns

: return the device type defined as DEVICE_TYPE in the IDTDef.h

11.1.4.18 device_getDRS()

```
int device_getDRS (
    BYTE * codeDRS,
    int * codeDRSLen )
```

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

Parameters

<i>codeDRSLen</i>	the length of codeDRS
-------------------	-----------------------

Returns

RETURN_CODE: Values can be parsed with [device_getResponseCodeString\(\)](#) Note: Only support TTK devices

11.1.4.19 device_getFirmwareVersion()

```
int device_getFirmwareVersion (
    OUT char * firmwareVersion )
```

Polls device for Firmware Version

Parameters

<i>firmwareVersion</i>	Response returned of Firmware Version; needs to have at least 128 bytes of memory
------------------------	---

Returns

RETURN_CODE: Values can be parsed with [device_getResponseCodeString\(\)](#)

11.1.4.20 device_getFirmwareVersion_Len()

```
int device_getFirmwareVersion_Len (
    OUT char * firmwareVersion,
    IN_OUT int * firmwareVersionLen )
```

DEPRECATED : please use [device_getFirmwareVersion_Len\(OUT char* firmwareVersion, IN_OUT int *firmwareVersionLen\)](#)

Polls device for Firmware Version

Parameters

<i>firmwareVersion</i>	Response returned of Firmware Version
<i>firmwareVersionLen</i>	Length of Firmware Version

Returns

RETURN_CODE: Values can be parsed with [device_getResponseCodeString\(\)](#)

11.1.4.21 device_getKeyStatus()

```
int device_getKeyStatus (
    int * newFormat,
    BYTE * status,
    int * statusLen )
```

Get Key Status

Gets the status of loaded keys

Parameters

<i>status</i>	newFormat for Augusta and miniSmartII only 1: new format of key status 0: reserved format for support previous device
<i>status</i>	For L80, L100, Augusta and miniSmartII: When the newFormat is 0, data format as follows. For Augusta and miniSmartII: 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 For L80 and L100: byte 0: PIN DUKPT Key byte 1: PIN Master Key byte 2: Standard PIN Session Key byte 3: Desjardins PIN Session Key byte 4: Account/MSR DUKPT Key, 1 Exists, 0 None, 0xFF STOP, Does not support, always 0 byte 5: Account/ICC DUKPT Key, 1 Exists, 0 None, 0xFF STOP, Does not support, always 0 byte 6: Admin DUKPT Key, 1 Exists, 0 None, 0xFF STOP byte 7: Data DUKPT Key, 1 Exists, 0 None, 0xFF STOP byte 8: MAC DUKPT Key, 1 Exists, 0 None, 0xFF STOP

when the newFormat is 1, 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) For NEO2 and SREDKey2: Each unit of three bytes represents one key's parameters (index and slot). Key Name Index (1 byte): 0x14 - LCL-KEK 0x01 - Pin encryption Key (NEO2 only) 0x02 - Data encryption Key 0x05 - MAC DUKPT Key 0x0A - PCI Pairing Key (NEO2 only) Key Slot (2 bytes): Indicate different slots of a certain Key Name Example: slot =5 (0x00 0x05), slot=300 (0x01 0x2C) For BTPay380, slot is always 0 For example, 0x14 0x00 0x00 0x02 0x00 0x00 0x0A 0x00 0x00 will represent [KeyNameIndex=0x14,KeySlot=0x0000], [KeyNameIndex=0x02,KeySlot=0x0000] and [KeyNameIndex=0x0A,KeySlot=0x0000]

Parameters

<i>statusLen</i>	the length of status
------------------	----------------------

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString

11.1.4.22 device_getResponseCodeString()

```
void device_getResponseCodeString (
    IN int returnCode,
    OUT char * description )
```

Review the return code description.

Parameters

<i>returnCode</i>	the response result.
<i>description</i>	

Return values

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

- 0: "no error, beginning task";
- 1: "no response from reader";
- 2: "invalid response data";
- 3: "time out for task or CMD";
- 4: "wrong parameter";
- 5: "SDK is doing MSR or ICC task";
- 6: "SDK is doing PINPad task";
- 7: "SDK is doing CTLS task";
- 8: "SDK is doing EMV task";
- 9: "SDK is doing Other task";
- 10: "err response or data";
- 11: "no reader attached";
- 12: "mono audio is enabled";
- 13: "did connection";
- 14: "audio volume is too low";
- 15: "task or CMD be canceled";
- 16: "UF wrong string format";
- 17: "UF file not found";
- 18: "UF wrong file format";
- 19: "Attempt to contact online host failed";
- 20: "Attempt to perform RKI failed";
- 22: "Buffer size is not enough";
- 0X300: "Key Type(TDES) of Session Key is not same as the related Master Key.";
- 0X400: "Related Key was not loaded.";
- 0X500: "Key Same.";
- 0X501: "Key is all zero";
- 0X502: "TR-31 format error";
- 0X702: "PAN is Error Key.";

- 0X705: "No Internal MSR PAN (or Internal MSR PAN is erased timeout)";
- 0X0C01: "Incorrect Frame Tag";
- 0X0C02: "Incorrect Frame Type";
- 0X0C03: "Unknown Frame Type";
- 0X0C04: "Unknown Command";
- 0X0C05: "Unknown Sub-Command";
- 0X0C06: "CRC Error";
- 0X0C07: "Failed";
- 0X0C08: "Timeout";
- 0X0C0A: "Incorrect Parameter";
- 0X0C0B: "Command Not Supported";
- 0X0C0C: "Sub-Command Not Supported";
- 0X0C0D: "Parameter Not Supported / Status Abort Command";
- 0X0C0F: "Sub-Command Not Allowed";
- 0X0D01: "Incorrect Header Tag";
- 0X0D02: "Unknown Command";
- 0X0D03: "Unknown Sub-Command";
- 0X0D04: "CRC Error in Frame";
- 0X0D05: "Incorrect Parameter";
- 0X0D06: "Parameter Not Supported";
- 0X0D07: "Mal-formatted Data";
- 0X0D08: "Timeout";
- 0X0D0A: "Failed / NACK";
- 0X0D0B: "Command not Allowed";
- 0X0D0C: "Sub-Command not Allowed";
- 0X0D0D: "Buffer Overflow (Data Length too large for reader buffer)";
- 0X0D0E: "User Interface Event";
- 0X0D11: "Communication type not supported, VT-1, burst, etc.";
- 0X0D12: "Secure interface is not functional or is in an intermediate state.";
- 0X0D13: "Data field is not mod 8";
- 0X0D14: "Pad - 0X80 not found where expected";
- 0X0D15: "Specified key type is invalid";
- 0X0D1: "Could not retrieve key from the SAM(InitSecureComm)";
- 0X0D17: "Hash code problem";
- 0X0D18: "Could not store the key into the SAM(InstallKey)";
- 0X0D19: "Frame is too large";

- 0X0D1A: "Unit powered up in authentication state but POS must resend the InitSecureComm command";
- 0X0D1B: "The EEPROM may not be initialized because SecCommInterface does not make sense";
- 0X0D1C: "Problem encoding APDU";
- 0X0D20: "Unsupported Index(ILM) SAM Transceiver error - problem communicating with the SAM(Key Mgr)";
- 0X0D2: "Unexpected Sequence Counter in multiple frames for single bitmap(ILM) Length error in data returned from the SAM(Key Mgr)";
- 0X0D22: "Improper bit map(ILM)";
- 0X0D23: "Request Online Authorization";
- 0X0D24: "ViVOCard3 raw data read successful";
- 0X0D25: "Message index not available(ILM) ViVOcomm activate transaction card type(ViVOcomm)";
- 0X0D26: "Version Information Mismatch(ILM)";
- 0X0D27: "Not sending commands in correct index message index(ILM)";
- 0X0D28: "Time out or next expected message not received(ILM)";
- 0X0D29: "ILM languages not available for viewing(ILM)";
- 0X0D2A: "Other language not supported(ILM)";
- 0X0D41: "Unknown Error from SAM";
- 0X0D42: "Invalid data detected by SAM";
- 0X0D43: "Incomplete data detected by SAM";
- 0X0D44: "Reserved";
- 0X0D45: "Invalid key hash algorithm";
- 0X0D46: "Invalid key encryption algorithm";
- 0X0D47: "Invalid modulus length";
- 0X0D48: "Invalid exponent";
- 0X0D49: "Key already exists";
- 0X0D4A: "No space for new RID";
- 0X0D4B: "Key not found";
- 0X0D4C: "Crypto not responding";
- 0X0D4D: "Crypto communication error";
- 0X0D4E: "Module-specific error for Key Manager";
- 0X0D4F: "All key slots are full (maximum number of keys has been installed)";
- 0X0D50: "Auto-Switch OK";
- 0X0D51: "Auto-Switch failed";
- 0X0D90: "Account DUKPT Key not exist";
- 0X0D91: "Account DUKPT Key KSN exhausted";
- 0X0D00: "This Key had been loaded.";
- 0X0E00: "Base Time was loaded.";

- 0X0F00: "Encryption Or Decryption Failed.";
- 0X1000: "Battery Low Warning (It is High Priority Response while Battery is Low.)";
- 0X1800: "Send 'Cancel Command' after send 'Get Encrypted PIN' & 'Get Numeric' & 'Get Amount'; - 0↵
X1900: "Press 'Cancel' key after send 'Get Encrypted PIN' & 'Get Numeric' & 'Get Amount'";
- 0X30FF: "Security Chip is not connect";
- 0X3000: "Security Chip is deactivation & Device is In Removal Legally State.";
- 0X3101: "Security Chip is activation & Device is In Removal Legally State.";
- 0X5500: "No Admin DUKPT Key.";
- 0X5501: "Admin DUKPT Key STOP.";
- 0X5502: "Admin DUKPT Key KSN is Error.";
- 0X5503: "Get Authentication Code1 Failed.";
- 0X5504: "Validate Authentication Code Error.";
- 0X5505: "Encrypt or Decrypt data failed.";
- 0X5506: "Not Support the New Key Type.";
- 0X5507: "New Key Index is Error.";
- 0X5508: "Step Error.";
- 0X5509: "KSN Error";
- 0X550A: "MAC Error.";
- 0X550B: "Key Usage Error.";
- 0X550C: "Mode Of Use Error.";
- 0X550F: "Other Error.";
- 0X6000: "Save or Config Failed / Or Read Config Error.";
- 0X6200: "No Serial Number.";
- 0X6900: "Invalid Command - Protocol is right, but task ID is invalid.";
- 0X6A01: "Unsupported Command - Protocol and task ID are right, but command is invalid - In this State";
- 0X6A00: "Unsupported Command - Protocol and task ID are right, but command is invalid.";
- 0X6B00: "Unknown parameter in command - Protocol task ID and command are right, but parameter is invalid.";
- 0X6C00: "Unknown parameter in command - Protocol task ID and command are right, but length is out of the requirement.";
- 0X7200: "Device is suspend (MKSK suspend or press password suspend).";
- 0X7300: "PIN DUKPT is STOP (21 bit 1).";
- 0X7400: "Device is Busy.";
- 0XE100: "Can not enter sleep mode";
- 0XE200: "File has existed";
- 0XE300: "File has not existed";
- 0XE313: "IO line low -- Card error after session start";

- 0XE400: "Open File Error";
- 0XE500: "SmartCard Error";
- 0XE600: "Get MSR Card data is error";
- 0XE700: "Command time out";
- 0XE800: "File read or write is error";
- 0XE900: "Active 1850 error!";
- 0XEA00: "Load bootloader error";
- 0XEF00: "Protocol Error- STX or ETX or check error.";
- 0XEB00: "Picture is not exist";
- 0X2C02: "No Microprocessor ICC seated";
- 0X2C06: "no card seated to request ATR";
- 0X2D01: "Card Not Supported,";
- 0X2D03: "Card Not Supported, wants CRC";
- 0X690D: "Command not supported on reader without ICC support";
- 0X8100: "ICC error time out on power-up";
- 0X8200: "invalid TS character received - Wrong operation step";
- 0X8300: "Decode MSR Error";
- 0X8400: "TriMagII no Response";
- 0X8500: "No Swipe MSR Card";
- 0X8510: "No Financial Card";
- 0X8600: "Unsupported F, D, or combination of F and D";
- 0X8700: "protocol not supported EMV TD1 out of range";
- 0X8800: "power not at proper level";
- 0X8900: "ATR length too long";
- 0X8B01: "EMV invalid TA1 byte value";
- 0X8B02: "EMV TB1 required";
- 0X8B03: "EMV Unsupported TB1 only 00 allowed";
- 0X8B04: "EMV Card Error, invalid BWI or CWI";
- 0X8B06: "EMV TB2 not allowed in ATR";
- 0X8B07: "EMV TC2 out of range";
- 0X8B08: "EMV TC2 out of range";
- 0X8B09: "per EMV96 TA3 must be > - 0XF";
- 0X8B10: "ICC error on power-up";
- 0X8B11: "EMV T=1 then TB3 required";
- 0X8B12: "Card Error, invalid BWI or CWI";
- 0X8B13: "Card Error, invalid BWI or CWI";

- 0X8B17: "EMV TC1/TB3 conflict-";
- 0X8B20: "EMV TD2 out of range must be T=1";
- 0X8C00: "TCK error";
- 0XA304: "connector has no voltage setting";
- 0XA305: "ICC error on power-up invalid (SBLK(IFSD) exchange";
- 0XE301: "ICC error after session start";
- 0XFF00: "Request to go online";
- 0XFF01: "EMV: Accept the offline transaction";
- 0XFF02: "EMV: Decline the offline transaction";
- 0XFF03: "EMV: Accept the online transaction";
- 0XFF04: "EMV: Decline the online transaction";
- 0XFF05: "EMV: Application may fallback to magstripe technology";
- 0XFF06: "EMV: ICC detected tah the conditions of use are not satisfied";
- 0XFF07: "EMV: ICC didn't accept transaction";
- 0XFF08: "EMV: Transaction was cancelled";
- 0XFF09: "EMV: Application was not selected by kernel or ICC format error or ICC missing data error";
- 0XFF0A: "EMV: Transaction is terminated";
- 0XFF0B: "EMV: Other EMV Error";
- 0XFFFF: "NO RESPONSE";
- 0XF002: "ICC communication timeout";
- 0XF003: "ICC communication Error";
- 0XF00F: "ICC Card Seated and Highest Priority, disable MSR work request";
- 0XF200: "AID List / Application Data is not exist";
- 0XF201: "Terminal Data is not exist";
- 0XF202: "TLV format is error";
- 0XF203: "AID List is full";
- 0XF204: "Any CA Key is not exist";
- 0XF205: "CA Key RID is not exist";
- 0XF206: "CA Key Index it not exist";
- 0XF207: "CA Key is full";
- 0XF208: "CA Key Hash Value is Error";
- 0XF209: "Transaction format error";
- 0XF20A: "The command will not be processing";
- 0XF20B: "CRL is not exist";
- 0XF20C: "CRL number exceed max number";
- 0XF20D: "Amount,Other Amount,Trasaction Type are missing";

- 0XF20E: "The Identification of algorithm is mistake";
- 0XF20F: "No Financial Card";
- 0XF210: "In Encrypt Result state, TLV total Length is greater than Max Length";
- 0X1001: "INVALID ARG";
- 0X1002: "FILE_OPEN_FAILED";
- 0X1003: "FILE OPERATION_FAILED";
- 0X2001: "MEMORY_NOT_ENOUGH";
- 0X3002: "SMARTCARD_FAIL";
- 0X3003: "SMARTCARD_INIT_FAILED";
- 0X3004: "FALLBACK_SITUATION";
- 0X3005: "SMARTCARD_ABSENT";
- 0X3006: "SMARTCARD_TIMEOUT";
- 0X3012: "EMV_RESULT_CODE_MSR_CARD_ERROR_FALLBACK";
- 0X5001: "EMV_PARSING_TAGS_FAILED";
- 0X5002: "EMV_DUPLICATE_CARD_DATA_ELEMENT";
- 0X5003: "EMV_DATA_FORMAT_INCORRECT";
- 0X5004: "EMV_NO_TERM_APP";
- 0X5005: "EMV_NO_MATCHING_APP";
- 0X5006: "EMV_MISSING_MANDATORY_OBJECT";
- 0X5007: "EMV_APP_SELECTION_RETRY";
- 0X5008: "EMV_GET_AMOUNT_ERROR";
- 0X5009: "EMV_CARD_REJECTED";
- 0X5010: "EMV_AIP_NOT_RECEIVED";
- 0X5011: "EMV_AFL_NOT_RECEIVED";
- 0X5012: "EMV_AFL_LEN_OUT_OF_RANGE";
- 0X5013: "EMV_SFI_OUT_OF_RANGE";
- 0X5014: "EMV_AFL_INCORRECT";
- 0X5015: "EMV_EXP_DATE_INCORRECT";
- 0X5016: "EMV_EFF_DATE_INCORRECT";
- 0X5017: "EMV_ISS_COD_TBL_OUT_OF_RANGE";
- 0X5018: "EMV_CRYPTOGram_TYPE_INCORRECT";
- 0X5019: "EMV_PSE_NOT_SUPPORTED_BY_CARD";
- 0X5020: "EMV_USER_SELECTED_LANGUAGE";
- 0X5021: "EMV_SERVICE_NOT_ALLOWED";
- 0X5022: "EMV_NO_TAG_FOUND";
- 0X5023: "EMV_CARD_BLOCKED";

- 0X5024: "EMV_LEN_INCORRECT";
- 0X5025: "CARD_COM_ERROR";
- 0X5026: "EMV_TSC_NOT_INCREASED";
- 0X5027: "EMV_HASH_INCORRECT";
- 0X5028: "EMV_NO_ARC";
- 0X5029: "EMV_INVALID_ARC";
- 0X5030: "EMV_NO_ONLINE_COMM";
- 0X5031: "TRAN_TYPE_INCORRECT";
- 0X5032: "EMV_APP_NO_SUPPORT";
- 0X5033: "EMV_APP_NOT_SELECT";
- 0X5034: "EMV_LANG_NOT_SELECT";
- 0X5035: "EMV_NO_TERM_DATA";
- 0X5039: "EMV_PIN_ENTRY_TIMEOUT";
- 0X6001: "CVM_TYPE_UNKNOWN";
- 0X6002: "CVM_AIP_NOT_SUPPORTED";
- 0X6003: "CVM_TAG_8E_MISSING";
- 0X6004: "CVM_TAG_8E_FORMAT_ERROR";
- 0X6005: "CVM_CODE_IS_NOT_SUPPORTED";
- 0X6006: "CVM_COND_CODE_IS_NOT_SUPPORTED";
- 0X6007: "NO_MORE_CVM";
- 0X6008: "PIN_BYPASSED_BEFORE";
- 0X7001: "PK_BUFFER_SIZE_TOO_BIG";
- 0X7002: "PK_FILE_WRITE_ERROR";
- 0X7003: "PK_HASH_ERROR";
- 0X8001: "NO_CARD_HOLDER_CONFIRMATION";
- 0X8002: "GET_ONLINE_PIN";
- 0XD000: "Data not exist";
- 0XD001: "Data access error";
- 0XD100: "RID not exist";
- 0XD101: "RID existed";
- 0XD102: "Index not exist";
- 0XD200: "Maximum exceeded";
- 0XD201: "Hash error";
- 0XD205: "System Busy";
- 0X0E01: "Unable to go online";
- 0X0E02: "Technical Issue";

- 0X0E03: "Declined";
- 0X0E04: "Issuer Referral transaction";
- 0X0F01: "Decline the online transaction";
- 0X0F02: "Request to go online";
- 0X0F03: "Transaction is terminated";
- 0X0F05: "Application was not selected by kernel or ICC format error or ICC missing data error";
- 0X0F07: "ICC didn't accept transaction";
- 0X0F0A: "Application may fallback to magstripe technology";
- 0X0F0C: "Transaction was cancelled";
- 0X0F0D: "Timeout";
- 0X0F0F: "Other EMV Error";
- 0X0F10: "Accept the offline transaction";
- 0X0F11: "Decline the offline transaction";
- 0X0F21: "ICC detected tah the conditions of use are not satisfied";
- 0X0F22: "No app were found on card matching terminal configuration";
- 0X0F23: "Terminal file does not exist";
- 0X0F24: "CAPK file does not exist";
- 0X0F25: "CRL Entry does not exist";
- 0X0FFE: "code when blocking is disabled";
- 0X0FFF: "code when command is not applicable on the selected device";
- 0XF005: "ICC Encrypted C-APDU Data Structure Length Error Or Format Error.";
- 0XBBE0: "CM100 Success";
- 0XBBE1: "CM100 Parameter Error";
- 0XBBE2: "CM100 Low Output Buffer";
- 0XBBE3: "CM100 Card Not Found";
- 0XBBE4: "CM100 Collision Card Exists";
- 0XBBE5: "CM100 Too Many Cards Exist";
- 0XBBE6: "CM100 Saved Data Does Not Exist";
- 0XBBE8: "CM100 No Data Available";
- 0XBBE9: "CM100 Invalid CID Returned";
- 0XBBEA: "CM100 Invalid Card Exists";
- 0XBBEC: "CM100 Command Unsupported";
- 0XBBED: "CM100 Error In Command Process";
- 0XBBEE: "CM100 Invalid Command";
- 0X9031: "Unknown command";
- 0X9032: "Wrong parameter (such as the length of the command is incorrect)";

- 0X9038: "Wait (the command couldnt be finished in BWT)";
- 0X9039: "Busy (a previously command has not been finished)";
- 0X903A: "Number of retries over limit";
- 0X9040: "Invalid Manufacturing system data";
- 0X9041: "Not authenticated";
- 0X9042: "Invalid Master DUKPT Key";
- 0X9043: "Invalid MAC Key";
- 0X9044: "Reserved for future use";
- 0X9045: "Reserved for future use";
- 0X9046: "Invalid DATA DUKPT Key";
- 0X9047: "Invalid PIN Pairing DUKPT Key";
- 0X9048: "Invalid DATA Pairing DUKPT Key";
- 0X9049: "No nonce generated";
- 0X9949: "No GUID available. Perform getVersion first.";
- 0X9950: "MAC Calculation unsuccessful. Check BDK value.";
- 0X904A: "Not ready";
- 0X904B: "Not MAC data";
- 0X9050: "Invalid Certificate";
- 0X9051: "Duplicate key detected";
- 0X9052: "AT checks failed";
- 0X9053: "TR34 checks failed";
- 0X9054: "TR31 checks failed";
- 0X9055: "MAC checks failed";
- 0X9056: "Firmware download failed";
- 0X9060: "Log is full";
- 0X9061: "Removal sensor unengaged";
- 0X9062: "Any hardware problems";
- 0X9070: "ICC communication timeout";
- 0X9071: "ICC data error (such check sum error)";
- 0X9072: "Smart Card not powered up";

11.1.4.23 device_getSDKWaitTime()

```
int device_getSDKWaitTime ( )
```

Get SDK Wait Time

Get the SDK wait time for transactions

Returns

SDK wait time in seconds

11.1.4.24 device_getThreadStackSize()

```
int device_getThreadStackSize ( )
```

Get Thread Stack Size

Get the stack size setting for newly created threads

Returns

Thread Stack Size

11.1.4.25 device_init()

```
int device_init ( )
```

Initial the device by USB

It will detect the device and trying connect.

The connect status can be checked by [device_isConnected\(\)](#).

Note: after the function returns success, the function [device_setCurrentDevice\(\)](#) has to be called to set the device type.

Returns

RETURN_CODE: Values can be parsed with [device_getResponseCodeString\(\)](#)

11.1.4.26 device_isAttached()

```
int device_isAttached (
    int deviceType )
```

Check if the device is attached to the USB port The function [device_init\(\)](#) must be called before this function.

Parameters

<i>deviceType, the</i>	device type of the USB device
------------------------	-------------------------------

Returns

1 if the device is attached, or 0 if the device is not attached

11.1.4.27 device_isConnected()

```
int device_isConnected ( )
```

Check the device connected status

Returns

DEVICE_DISCONNECT=0, or DEVICE_CONNECTED = 1

11.1.4.28 device_rebootDevice()

```
int 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\(\)](#)

11.1.4.29 device_registerCameraCallBk()

```
void device_registerCameraCallBk (
    pCMR_callback pCMRf )
```

To register the camera callback function to get the image data. (Pass NULL to disable the callback.)

11.1.4.30 device_registerCardStatusFrontSwitchCallBk()

```
void device_registerCardStatusFrontSwitchCallBk (
    pCSFS_callback pCSFSf )
```

To register the card status and front switch callback function to get status. (Pass NULL to disable the callback.)

11.1.4.31 device_registerFWCallBk()

```
void device_registerFWCallBk (
    pFW_callback pFWf )
```

To register the firmware update callback function to get the status of firmware update. (Pass NULL to disable the callback.)

11.1.4.32 `device_SendDataCommand()`

```
int device_SendDataCommand (
    IN BYTE * cmd,
    IN int cmdLen,
    IN BYTE * data,
    IN int dataLen,
    OUT BYTE * response,
    IN_OUT int * respLen )
```

Send a Command to device

Sends a command to the device .

Parameters

<i>cmd</i>	buffer of command to execute.
<i>cmdLen,the</i>	length of the buffer cmd.
<i>data</i>	buffer of IDG command data.
<i>dataLen,the</i>	length of the buffer data.
<i>response</i>	Response data
<i>respLen,the</i>	length of Response data

Returns

RETURN_CODE: Values can be parsed with [device_getResponseCodeString\(\)](#)

11.1.4.33 `device_setCurrentDevice()`

```
int device_setCurrentDevice (
    int deviceType )
```

Sets the current device to talk to

The connect status can be checked by [device_isConnected\(\)](#).

Parameters

<i>deviceType</i>	Device to connect to <pre>enum DEVICE_TYPE { IDT_DEVICE_UNKNOWN=0, IDT_DEVICE_AUGUSTA_HID, IDT_DEVICE_AUGUSTA_KB, IDT_DEVICE_AUGUSTA_S_HID, IDT_DEVICE_AUGUSTA_S_KB, IDT_DEVICE_AUGUSTA_S_TTK_HID, IDT_DEVICE_SPECTRUM_PRO, IDT_DEVICE_MINISMART_II, IDT_DEVICE_L80, IDT_DEVICE_L100, IDT_DEVICE_UNIPAY, IDT_DEVICE_UNIPAY_I_V, IDT_DEVICE_VP3300_AJ, IDT_DEVICE_KIOSK_III, IDT_DEVICE_KIOSK_III_S, IDT_DEVICE_PIP_READER, IDT_DEVICE_VENDI, IDT_DEVICE_VP3300_USB, IDT_DEVICE_UNIPAY_I_V_TTK, IDT_DEVICE_VP3300_BT, IDT_DEVICE_VP8800, IDT_DEVICE_SREDKEY2_HID, IDT_DEVICE_SREDKEY2_KB, IDT_DEVICE_NEO2, IDT_DEVICE_MINISMART_II_COM = IDT_DEVICE_NEO2+5, IDT_DEVICE_SPECTRUM_PRO_COM, IDT_DEVICE_KIOSK_III_COM, IDT_DEVICE_KIOSK_III_S_COM, IDT_DEVICE_PIP_READER_COM, IDT_DEVICE_VP3300_COM, IDT_DEVICE_NEO2_COM, IDT_DEVICE_MAX_DEVICES = IDT_DEVICE_NEO2_COM+5 };</pre>
-------------------	---

Returns

RETURN_CODE: 1: success, 0: failed

11.1.4.34 device_setSDKWaitTime()

```
void device_setSDKWaitTime (
    int waitTime )
```

Set SDK Wait Time

Set the SDK wait time for transactions

Parameters

<i>waitTime</i>	The SDK wait time for transaction in seconds. The Maximum is 2147483 seconds.
-----------------	---

11.1.4.35 device_setThreadStackSize()

```
void device_setThreadStackSize (
    int threadSize )
```

Set Thread Stack Size

Set the stack size setting for newly created threads

11.1.4.36 device_setTransactionExponent()

```
void device_setTransactionExponent (
    int exponent )
```

Sets the transaction exponent to be used with device_startTransaction. Default value is 2

Parameters

<i>exponent, The</i>	exponent to use when calling device_startTransaction
----------------------	--

11.1.4.37 device_startTransaction()

```
int device_startTransaction (
    IN double amount,
    IN double amtOther,
    IN int type,
    IN const int _timeout,
    IN BYTE * tags,
    IN int tagsLen )
```

Start Transaction Request

The tags will be returned in the callback routine.

Parameters

<i>amount</i>	Transaction amount value (tag value 9F02) • SEE IMPORTANT NOTE BELOW
---------------	---

Parameters

<i>amtOther</i>	Other amount value, if any (tag value 9F03) <ul style="list-style-type: none">• SEE IMPORTANT NOTE BELOW
<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. 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>tagsLen</i>	The length of tags data buffer.

>>>>>IMPORTANT: parameters for amount and amtOther MUST BE PASSED AS A DOUBLE VALUE WITH DECIMAL POINT. Example, do not pass 1, but instead pass 1.0 or 1.00. Otherwise, results will be unpredictable

Returns

RETURN_CODE: Values can be parsed with [device_getResponseCodeString\(\)](#)

11.1.4.38 device_updateFirmware()

```
int device_updateFirmware (
    IN BYTE * firmwareData,
    IN int firmwareDataLen,
    IN char * firmwareName,
    IN int encryptionType,
    IN BYTE * keyBlob,
    IN int keyBlobLen )
```

Update Firmware Updates the firmware of Augusta.

Parameters

<i>firmwareData</i>	Signed binary data of a firmware file provided by IDTech
<i>firmwareDataLen</i>	Length of firmwareData
<i>firmwareName</i>	Firmware name. <ul style="list-style-type: none">• For example "Augusta_S_TTK_V1.00.002.fm"
<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)
<i>keyBlobLen</i>	Length of keyBlob

Returns

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

Firmware update status is returned in the callback with the following values: sender = AUGUSTA state = Device↔ State.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

11.1.4.39 emv_activateTransaction()

```
int emv_activateTransaction (
    IN int timeout,
    IN BYTE * tags,
    IN int tagsLen,
    IN int forceOnline )
```

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>	Tags to be included in the request. Passed as a TLV stream. Example, tag 9F0C with amount 0x000000000100 would be 0x9F0C06000000000100
<i>tagsLen</i>	Length of tags
<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

Returns

RETURN_CODE: Values can be parsed with `device_getResponseCodeString` >>>>IMPORTANT: parameters for amount and amtOther MUST BE PASSED AS A DOUBLE VALUE WITH DECIMAL POINT. Example, do not pass 1, but instead pass 1.0 or 1.00. Otherwise, results will be unpredictable

11.1.4.40 emv_allowFallback()

```
void emv_allowFallback (
    IN int allow )
```

Allow fallback for EMV transactions. Default is TRUE

Parameters

<i>allow</i>	TRUE = allow fallback, FALSE = don't allow fallback
--------------	---

11.1.4.41 emv_authenticateTransaction()

```
int emv_authenticateTransaction (
    IN BYTE * updatedTLV,
    IN int updatedTLVLen )
```

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 = DFEE1A079F029F36959F37
<i>updatedTLVLen</i>	

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString

11.1.4.42 emv_authenticateTransactionWithTimeout()

```
int emv_authenticateTransactionWithTimeout (
    IN int timeout,
    IN BYTE * updatedTLV,
    IN int updatedTLVLen )
```

Authenticate EMV Transaction Request with Timeout

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>timeout</i>	Timeout value in seconds.
----------------	---------------------------

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 = DFEE1A079F029F36959F37
<i>updatedTLVLen</i>	

Returns

RETURN_CODE: Values can be parsed with `device_getResponseCodeString`

11.1.4.43 `emv_callbackResponseLCD()`

```
int emv_callbackResponseLCD (
    IN int type,
    byte selection )
```

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 <code>EMV_LCD_DISPLAY_MODE_CANCEL</code> . Otherwise, value can be <code>EMV_LCD_DISPLAY_MODE_MENU</code> , <code>EMV_LCD_DISPLAY_MODE_PROMPT</code> , or <code>EMV_LCD_DISPLAY_MODE_LANGUAGE_SELECT</code>
<i>selection</i>	If <code>type = EMV_LCD_DISPLAY_MODE_MENU</code> or <code>EMV_LCD_DISPLAY_MODE_LANGUAGE_SELECT</code> , provide the selection ID line number. Otherwise, if <code>type = EMV_LCD_DISPLAY_MODE_PROMPT</code> supply either 0x43 ('C') for Cancel, or 0x45 ('E') for Enter/accept

Returns

RETURN_CODE: Values can be parsed with `device_getResponseCodeString()`

11.1.4.44 `emv_callbackResponseMSR()`

```
int emv_callbackResponseMSR (
    IN BYTE * MSR,
    IN_OUT int MSRlen )
```

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>MSRLen</i>	the length of Swiped track data

Returns

RETURN_CODE: Values can be parsed with [device_getResponseCodeString\(\)](#)

11.1.4.45 emv_cancelTransaction()

```
int emv_cancelTransaction ( )
```

Cancel EMV Transaction

Cancels the currently executing EMV transaction.

Returns

RETURN_CODE: Values can be parsed with [device_getResponseCodeString\(\)](#)

11.1.4.46 emv_completeTransaction()

```
int emv_completeTransaction (
    IN int commError,
    IN BYTE * authCode,
    IN int authCodeLen,
    IN BYTE * iad,
    IN int iadLen,
    IN BYTE * tlvScripts,
    IN int tlvScriptsLen,
    IN BYTE * tlv,
    IN int tlvLen )
```

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(1) if host was unreachable, or FALSE(0) 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>authCodeLen</i>	the length of authCode
<i>iad</i>	Issuer Authentication Data, if any. Example 0x11223344556677883030 (tag value 91).
<i>iadLen</i>	the length of iadLen
<i>tlvScripts</i>	71/72 scripts, if any

Parameters

<i>tlvScriptsLen</i>	the length of tlvScriptsLen
<i>tlv</i>	Additional TLV data to return with transaction results (if any)
<i>tlvLen</i>	the length of tlv

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString

11.1.4.47 emv_getAutoAuthenticateTransaction()

```
int emv_getAutoAuthenticateTransaction ( )
```

Gets auto authenticate value for EMV transactions.

Returns

RETURN_CODE: TRUE = auto authenticate, FALSE = manually authenticate

11.1.4.48 emv_getAutoCompleteTransaction()

```
int emv_getAutoCompleteTransaction ( )
```

Gets auto complete value for EMV transactions.

Returns

RETURN_CODE: TRUE = auto complete, FALSE = manually complete

11.1.4.49 emv_getEMVConfigurationCheckValue()

```
int emv_getEMVConfigurationCheckValue (
    OUT BYTE * checkValue,
    IN_OUT int * checkValueLen )
```

Get EMV Kernel configuration check value info

Parameters

<i>checkValue</i>	Response returned of Kernel configuration check value info
<i>checkValueLen</i>	the length of checkValue

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString

11.1.4.50 `emv_getEMVKernelCheckValue()`

```
int emv_getEMVKernelCheckValue (
    OUT BYTE * checkValue,
    IN_OUT int * checkValueLen )
```

Get EMV Kernel check value info

Parameters

<i>checkValue</i>	Response returned of Kernel check value info
<i>checkValueLen</i>	the length of <i>checkValue</i>

Returns

RETURN_CODE: Values can be parsed with `device_getResponseCodeString`

11.1.4.51 `emv_getEMVKernelVersion()`

```
int emv_getEMVKernelVersion (
    OUT char * version )
```

DEPRECATED : please use `emv_getEMVKernelVersion_Len(OUT char* version, IN_OUT int *versionLen)`

Polls device for EMV Kernel Version

Parameters

<i>version</i>	Response returned of Kernel Version; needs to have at least 128 bytes of memory.
----------------	--

Returns

RETURN_CODE: Values can be parsed with `device_getResponseCodeString()`

11.1.4.52 `emv_getEMVKernelVersion_Len()`

```
int emv_getEMVKernelVersion_Len (
    OUT char * version,
    IN_OUT int * versionLen )
```

Polls device for EMV Kernel Version

Parameters

<i>version</i>	Response returned of Kernel Version
<i>versionLen</i>	Length of <i>version</i>

Returns

RETURN_CODE: Values can be parsed with `device_getResponseCodeString()`

11.1.4.53 `emv_registerCallBk()`

```
void emv_registerCallBk (
    pEMV_callback pEMVf )
```

To register the emv callback function to get the EMV processing response. (Pass NULL to disable the callback.)

11.1.4.54 `emv_removeAllApplicationData()`

```
int emv_removeAllApplicationData ( )
```

Remove All Application Data

Removes all the Application Data

Returns

RETURN_CODE: Values can be parsed with [device_getResponseCodeString\(\)](#)

11.1.4.55 `emv_removeAllCAPK()`

```
int emv_removeAllCAPK ( )
```

Remove All Certificate Authority Public Key

Removes all the CAPK

Returns

RETURN_CODE: Values can be parsed with [device_getResponseCodeString\(\)](#)

11.1.4.56 `emv_removeAllCRL()`

```
int emv_removeAllCRL ( )
```

Remove All Certificate Revocation List Entries

Removes all CRLentry entries

Returns

RETURN_CODE: Values can be parsed with [device_getResponseCodeString\(\)](#)

11.1.4.57 `emv_removeApplicationData()`

```
int emv_removeApplicationData (
    IN BYTE * AID,
    IN int AIDLen )
```

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>AIDLen</i>	the length of AID data buffer

Returns

RETURN_CODE: Values can be parsed with [device_getResponseCodeString\(\)](#)

11.1.4.58 emv_removeCAPK()

```
int emv_removeCAPK (
    IN BYTE * capk,
    IN int capkLen )
```

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>capkLen</i>	the length of capk data buffer

Returns

RETURN_CODE: Values can be parsed with [device_getResponseCodeString\(\)](#)

11.1.4.59 emv_removeCRL()

```
int emv_removeCRL (
    IN BYTE * list,
    IN int lsLen )
```

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>lsLen</i>	the length of list data buffer

Returns

RETURN_CODE: Values can be parsed with [device_getResponseCodeString\(\)](#)

11.1.4.60 `emv_removeTerminalData()`

```
int emv_removeTerminalData ( )
```

Remove Terminal Data

Removes the Terminal Data

Returns

RETURN_CODE: Values can be parsed with [device_getResponseCodeString\(\)](#)

11.1.4.61 `emv_retrieveAIDList()`

```
int emv_retrieveAIDList (
    OUT BYTE * AIDList,
    IN_OUT int * AIDListLen )
```

Retrieve AID list

Returns all the AID names installed on the terminal.

Parameters

<i>AIDList</i>	array of AID name byte arrays
<i>AIDListLen</i>	the length of AIDList array buffer

Returns

RETURN_CODE: Values can be parsed with [device_getResponseCodeString\(\)](#)

11.1.4.62 `emv_retrieveApplicationData()`

```
int emv_retrieveApplicationData (
    IN BYTE * AID,
    IN int AIDLen,
    OUT BYTE * tlv,
    IN_OUT int * tlvLen )
```

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>AIDLen</i>	the length of AID data buffer.
<i>tlv</i>	The TLV elements of the requested AID
<i>tlvLen</i>	the length of tlv data buffer.

Returns

RETURN_CODE: Values can be parsed with [device_getResponseCodeString\(\)](#)

11.1.4.63 emv_retrieveCAPK()

```
int emv_retrieveCAPK (
    IN BYTE * capk,
    IN int capkLen,
    OUT BYTE * key,
    IN_OUT int * keyLen )
```

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>capkLen</i>	the length of capk data buffer
<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>keyLen</i>	<p>the length of key data buffer</p> <ul style="list-style-type: none"> •

Returns

RETURN_CODE: Values can be parsed with [device_getResponseCodeString\(\)](#)

11.1.4.64 emv_retrieveCAPKList()

```
int emv_retrieveCAPKList (
    OUT BYTE * keys,
    IN_OUT int * keysLen )
```

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>keysLen</i>	the length of keys data buffer

Returns

RETURN_CODE: Values can be parsed with [device_getResponseCodeString\(\)](#)

11.1.4.65 emv_retrieveCRL()

```
int emv_retrieveCRL (
    OUT BYTE * list,
    IN_OUT int * lssLen )
```

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>lssLen</i>	the length of list data buffer

Returns

RETURN_CODE: Values can be parsed with [device_getResponseCodeString\(\)](#)

11.1.4.66 emv_retrieveTerminalData()

```
int emv_retrieveTerminalData (
    OUT BYTE * tlv,
    IN_OUT int * tlvLen )
```

Retrieve Terminal Data

Retrieves the Terminal Data.

Parameters

<i>tlv</i>	Response returned as a TLV
<i>tlvLen</i>	the length of tlv data buffer

Returns

RETURN_CODE: Values can be parsed with [device_getResponseCodeString\(\)](#)

11.1.4.67 `emv_retrieveTerminalID()`

```
int emv_retrieveTerminalID (
    OUT char * terminalID )
```

DEPRECATED : please use `emv_retrieveTerminalID_Len(OUT char* terminalID, IN_OUT int *terminalIDLen)`

Gets the terminal ID as printable characters .

Parameters

<i>terminalID</i>	Terminal ID string; needs to have at least 30 bytes of memory
-------------------	---

Returns

RETURN_CODE: Values can be parsed with `device_getResponseCodeString()`

11.1.4.68 `emv_retrieveTerminalID_Len()`

```
int emv_retrieveTerminalID_Len (
    OUT char * terminalID,
    IN_OUT int * terminalIDLen )
```

Gets the terminal ID as printable characters .

Parameters

<i>terminalID</i>	Terminal ID string
<i>terminalIDLen</i>	Length of terminalID

Returns

RETURN_CODE: Values can be parsed with `device_getResponseCodeString()`

11.1.4.69 `emv_retrieveTransactionResult()`

```
int emv_retrieveTransactionResult (
    IN BYTE * tags,
    IN int tagsLen,
    IDTTransactionData * cardData )
```

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>tagsLen</i>	Length of tag list
<i>cardData</i>	All requested tags returned as unencrypted, encrypted and masked TLV data in IDTTransactionData object

Returns

RETURN_CODE: Values can be parsed with [device_getResponseCodeString\(\)](#)

11.1.4.70 emv_setApplicationData()

```
int emv_setApplicationData (
    IN BYTE * name,
    IN int nameLen,
    IN BYTE * tlv,
    IN int tlvLen )
```

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>nameLen</i>	the length of name data buffer of Application name,
<i>tlv</i>	Application data in TLV format
<i>tlvLen</i>	the length of tlv data buffer

Returns

RETURN_CODE: Values can be parsed with [device_getResponseCodeString\(\)](#)

11.1.4.71 emv_setAutoAuthenticateTransaction()

```
void emv_setAutoAuthenticateTransaction (
    IN int authenticate )
```

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

Parameters

<i>authenticate</i>	TRUE = auto authenticate, FALSE = manually authenticate
---------------------	---

11.1.4.72 emv_setAutoCompleteTransaction()

```
void emv_setAutoCompleteTransaction (
    IN int complete )
```

Enables complete for EMV transactions. If a `emv_authenticateTransaction` results in code 0x0004 (go online), then `emv_completeTransaction` can automatically execute if parameter is set to TRUE

Parameters

<i>complete</i>	TRUE = auto complete, FALSE = manually complete
-----------------	---

11.1.4.73 `emv_setCAPK()`

```
int emv_setCAPK (
    IN BYTE * capk,
    IN int capkLen )
```

Set Certificate Authority Public Key

Sets the CAPK as specified by the CAKey structure

Parameters

<i>capk</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>capkLen</i>	the length of capk data buffer

Returns

RETURN_CODE: Values can be parsed with [device_getResponseCodeString\(\)](#)

11.1.4.74 `emv_setCRL()`

```
int emv_setCRL (
    IN BYTE * list,
    IN int lsLen )
```

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>lsLen</i>	the length of list data buffer

Returns

RETURN_CODE: Values can be parsed with [device_getResponseCodeString\(\)](#)

11.1.4.75 emv_setTerminalData()

```
int emv_setTerminalData (
    IN BYTE * tlv,
    IN int tlvLen )
```

Set Terminal Data

Sets the Terminal Data as specified by the TerminalData structure passed as a parameter

Parameters

<i>tlv</i>	TerminalData configuration file
<i>tlvLen</i>	the length of tlv data buffer

Return values

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

11.1.4.76 emv_setTerminalID()

```
int emv_setTerminalID (
    IN char * terminalID )
```

Sets the terminal ID as printable characters .

Parameters

<i>terminalID</i>	Terminal ID to set
-------------------	--------------------

Returns

RETURN_CODE: Values can be parsed with [device_getResponseCodeString\(\)](#)

11.1.4.77 emv_startTransaction()

```
int emv_startTransaction (
    IN double amount,
    IN double amtOther,
    IN int exponent,
    IN int type,
    IN int timeout,
    IN BYTE * tags,
    IN int tagsLen,
    IN int forceOnline )
```

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) - SEE IMPORTANT NOTE BELOW
<i>amtOther</i>	Other amount value, if any (tag value 9F03) - SEE IMPORTANT NOTE BELOW
<i>exponent</i>	Number of characters after decimal 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 stream. Example, tag 9F0C with amount 0x000000000100 would be 0x9F0C06000000000100 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>tagsLen</i>	Length of tags
<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 = DFEE1A079F029F36959F37

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString >>>>IMPORTANT: parameters for amount and amtOther MUST BE PASSED AS A DOUBLE VALUE WITH DECIMAL POINT. Example, do not pass 1, but instead pass 1.0 or 1.00. Otherwise, results will be unpredictable

11.1.4.78 `icc_disable()`

```
int icc_disable ( )
```

ICC Function enable/disable Disable ICC function

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString

11.1.4.79 `icc_enable()`

```
int icc_enable (
    IN int withNotification )
```

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

Parameters

<i>withNotification</i>	<ul style="list-style-type: none"> • 1: with notification when ICC seated status changed, • 0: without notification.
-------------------------	--

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString

11.1.4.80 icc_exchangeAPDU()

```
int icc_exchangeAPDU (
    IN BYTE * c_APDU,
    IN int cLen,
    OUT BYTE * reData,
    IN_OUT int * reLen )
```

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>cLen</i>	APDU data packet length
<i>reData</i>	Unencrypted APDU response
<i>reLen</i>	Unencrypted APDU response data length

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString

11.1.4.81 icc_exchangeEncryptedAPDU()

```
int icc_exchangeEncryptedAPDU (
    IN BYTE * c_APDU,
    IN int cLen,
    OUT BYTE * reData,
    IN_OUT int * reLen )
```

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>cLen</i>	data packet length
-------------	--------------------

Parameters

<i>reData</i>	response encrypted APDU response. Can be three options:
---------------	---

[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]

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.

Parameters

<i>reLen</i>	encrypted APDU response data length
--------------	-------------------------------------

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString

11.1.4.82 `icc_getAPDU_KSN()`

```
int icc_getAPDU_KSN (
    OUT BYTE * KSN,
    IN_OUT int * inLen )
```

Get APDU KSN

Retrieves the KSN used in ICC Encrypted APDU usage

Parameters

<i>KSN</i>	Returns the encrypted APDU packet KSN
<i>inLen</i>	KSN data length

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString

11.1.4.83 `icc_getFunctionStatus()`

```
int icc_getFunctionStatus (
    OUT int * enabled,
    OUT int * withNotification )
```

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"> • 1: ICC Function enabled, • 0: means disabled.
<i>withNotification</i>	1 means with notification when ICC seated status changed. 0 means without notification.

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString

11.1.4.84 icc_getICCReaderStatus()

```
int icc_getICCReaderStatus (
    OUT BYTE * status )
```

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
---------------	--

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString

11.1.4.85 icc_getKeyFormatForICCDUKPT()

```
int icc_getKeyFormatForICCDUKPT (
    OUT BYTE * format )
```

Get Key Format For DUKPT

Specifies how data will be encrypted with Data Key or PIN key (if DUKPT key loaded). This applies to both MSR and ICC

Parameters

<i>format</i>	<p>Response returned from method:</p> <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.
---------------	---

Returns

RETURN_CODE: Values can be parsed with `device_getResponseCodeString`

11.1.4.86 `icc_getKeyTypeForICCDUKPT()`

```
int icc_getKeyTypeForICCDUKPT (
    OUT BYTE * type )
```

Get Key Type for DUKPT

Specifies the key type used for ICC DUKPT encryption. This applies to both MSR and ICC.

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.
-------------	---

Returns

RETURN_CODE: Values can be parsed with `device_getResponseCodeString`

11.1.4.87 `icc_powerOffICC()`

```
int icc_powerOffICC ( )
```

Power Off ICC

Powers down the ICC

Returns

RETURN_CODE: Values can be parsed with `device_getResponseCodeString()`

If Success, empty If Failure, ASCII encoded data of error string

11.1.4.88 `icc_powerOnICC()`

```
int icc_powerOnICC (
    OUT BYTE * ATR,
    IN_OUT int * inLen )
```

Power On ICC

Power up the currently selected microprocessor card in the ICC reader

Parameters

<i>ATR,the</i>	ATR data response when succeeded power on ICC,
<i>inLen,the</i>	length of ATR data

Returns

RETURN_CODE: Values can be parsed with [device_getResponseCodeString\(\)](#)

11.1.4.89 msr_cancelMSRSwipe()

```
int msr_cancelMSRSwipe ( )
```

Disable MSR Swipe Cancels MSR swipe request.

Returns

RETURN_CODE: Values can be parsed with [device_getResponseCodeString\(\)](#)

11.1.4.90 msr_captureMode()

```
int msr_captureMode (
    IN int isBufferMode,
    IN int withNotification )
```

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">• 1: Enter into Buffer mode.• 0: Enter into Auto mode. KB mode only. Swipes automatically captured and sent to keyboard buffer
<i>withNotification</i>	<ul style="list-style-type: none">• 1: with notification when swiped MSR data.• 0: without notification when swiped MSR data.

Returns

RETURN_CODE: Values can be parsed with [device_getResponseCodeString](#)

11.1.4.91 msr_disable()

```
int msr_disable ( )
```

Disable MSR Disable MSR functions.

Returns

RETURN_CODE: Values can be parsed with [device_getResponseCodeString\(\)](#)

11.1.4.92 msr_getClearPANID()

```
int msr_getClearPANID (
    OUT BYTE * value )
```

Get Clear PAN ID.

Returns the number of digits that begin the PAN that will be in the clear

Parameters

<i>value</i>	4901 <Setting value>="">. setting Value: Number of digits in clear. Values are char '0' - '6'
--------------	---

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString

11.1.4.93 msr_getExpirationMask()

```
int msr_getExpirationMask (
    OUT BYTE * value )
```

Get MSR expiration date mask.

Parameters

<i>value</i>	5001 <Setting value>="">. setting Value: '0' = masked, '1' = not-masked
--------------	---

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString

11.1.4.94 msr_getKeyFormatForICCDUKPT()

```
int msr_getKeyFormatForICCDUKPT (
    OUT BYTE * format )
```

Get Key Format For DUKPT

Specifies how data will be encrypted with Data Key or PIN key (if DUKPT key loaded). This applies to both MSR and ICC

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.
---------------	--

Returns

RETURN_CODE: Values can be parsed with `device_getResponseCodeString`

11.1.4.95 msr_getKeyTypeForICCDUKPT()

```
int msr_getKeyTypeForICCDUKPT (
    OUT BYTE * type )
```

Get Key Type for DUKPT

Specifies the key type used for ICC DUKPT encryption. This applies to both MSR and ICC.

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.

Returns

RETURN_CODE: Values can be parsed with `device_getResponseCodeString`

11.1.4.96 msr_getMSRData()

```
int msr_getMSRData (
    OUT BYTE * reData,
    IN_OUT int * reLen )
```

Get MSR Data Reads the MSR Data buffer**Parameters**

<i>reData</i>	Card data
<i>reLen</i>	Card data length

Returns

RETURN_CODE: Values can be parsed with [device_getResponseCodeString\(\)](#)

11.1.4.97 msr_getSetting()

```
int msr_getSetting (
    IN byte setting,
    OUT BYTE * value,
    IN_OUT int * valueLen )
```

Get Single MSR Setting value

Returns the encryption used for swipe data

Parameters

<i>setting</i>	the msr setting to retrieve.
<i>value</i>	MSR Setting value.
<i>valueLen</i>	the length of value.

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString

11.1.4.98 msr_getSwipeForcedEncryptionOption()

```
int msr_getSwipeForcedEncryptionOption (
    OUT BYTE * option )
```

Get MSR Swipe Forced Encryption Option.

Parameters

<i>option</i>	8401 <Setting value>="". Setting Value 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
---------------	---

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString

11.1.4.99 msr_getSwipeMaskOption()

```
int msr_getSwipeMaskOption (
    OUT BYTE * option )
```

Get MSR Swipe Mask Option.

Gets the swipe mask/clear data sending option

Parameters

<i>option</i>	8601 <Setting value>="". Setting Value 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
---------------	---

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString

11.1.4.100 msr_registerCallBk()

```
void msr_registerCallBk (
    pMSR_callback pMSRf )
```

To register the msr callback function to get the MSR card data. (Pass NULL to disable the callback.)

11.1.4.101 msr_registerCallBkp()

```
void msr_registerCallBkp (
    pMSR_callback pMSRF )
```

To register the msr callback function to get the MSR card data pointer. (Pass NULL to disable the callback.)

11.1.4.102 msr_setClearPANID()

```
int msr_setClearPANID (
    IN BYTE val )
```

Set Clear PAN ID.

Parameters

<i>val</i>	Set Clear PAN ID to value: Number of digits to show in clear. Range 0-6.
------------	--

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString

11.1.4.103 msr_setExpirationMask()

```
int msr_setExpirationMask (
    IN int mask )
```

Set Expiration Masking

Sets the flag to mask the expiration date

Parameters

<i>mask</i>	TRUE = mask expiration
-------------	------------------------

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString

11.1.4.104 msr_setKeyFormatForICCDUKPT()

```
int msr_setKeyFormatForICCDUKPT (
    IN BYTE format )
```

Set Key Format for DUKPT

Sets how data will be encrypted, with either TDES or AES (if DUKPT key loaded) This applies to both MSR and ICC

Parameters

<i>format</i>	encryption Encryption Type <ul style="list-style-type: none">• 00: Encrypt with TDES• 01: Encrypt with AES
---------------	---

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString

11.1.4.105 msr_setKeyTypeForICCDUKPT()

```
int msr_setKeyTypeForICCDUKPT (
    IN BYTE type )
```

Set Key Type for DUKPT Key

Sets which key the data will be encrypted with, with either Data Key or PIN key (if DUKPT key loaded) This applies to both MSR and ICC

Parameters

<i>type</i>	Encryption Type <ul style="list-style-type: none">• 00: Encrypt with Data Key• 01: Encrypt with PIN Key
-------------	--

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString

11.1.4.106 msr_setSetting()

```
int msr_setSetting (
    IN BYTE setting,
    IN BYTE * val,
    IN int valLen )
```

Set MSR settings.**Parameters**

<i>setting</i>	the msr setting to set.
<i>val</i>	the value to set to the msr setting.
<i>valLen</i>	the length of val.

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString

11.1.4.107 msr_setSwipeForcedEncryptionOption()

```
int msr_setSwipeForcedEncryptionOption (
    IN int track1,
    IN int track2,
    IN int track3,
    IN int track3card0 )
```

Set MSR Swipe Forced Encryption Option.

Parameters

<i>track1</i>	Set track1 encryption to true or false.
<i>track2</i>	Set track2 encryption to true or false.
<i>track3</i>	Set track3 encryption to true or false.
<i>track3card0</i>	Set track3 card0 encryption to true or false.

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString

11.1.4.108 msr_setSwipeMaskOption()

```
int msr_setSwipeMaskOption (
    IN int track1,
    IN int track2,
    IN int track3 )
```

Set MSR Swipe Mask Option.

Sets the swipe mask/clear data sending option

Parameters

<i>track1</i>	Set track1 mask to true or false.
<i>track2</i>	Set track2 mask to true or false.
<i>track3</i>	Set track3 mask to true or false.

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString

11.1.4.109 msr_startMSRSwipe()

```
int msr_startMSRSwipe (
    IN int _timeout )
```

Start 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
----------------	---------------------

Returns

RETURN_CODE: Values can be parsed with [device_getResponseCodeString\(\)](#)

11.1.4.110 parseMSRData()

```
void parseMSRData (
    IN BYTE * resData,
    IN int resLen,
    IN_OUT IDTMSRData * cardData )
```

Parser the MSR data from the buffer into IDTMSTData structure

Parameters

<i>resData</i>	MSR card data buffer
<i>resLen</i>	the length of resData
<i>cardData</i>	the parser result with IDTMSTData structure

11.1.4.111 pin_cancelPINEntry()

```
int pin_cancelPINEntry ( )
```

Cancel PIN Entry

Cancels PIN entry request

Returns

RETURN_CODE: Values can be parsed with [device_getResponseCodeString\(\)](#)

11.1.4.112 pin_registerCallBk()

```
void pin_registerCallBk (
    pPIN_callBack pPINf )
```

To register the pin callback function to get the PINPad data. (Pass NULL to disable the callback.)

11.1.4.113 registerHotplugCallBk()

```
void registerHotplugCallBk (
    pMessageHotplug pMsgHotplug )
```

To register the USB HID hot-plug callback function which implemented in the application to monitor the hotplug message from the SDK.

11.1.4.114 registerLogCallBk()

```
void registerLogCallBk (
    pSendDataLog pFSend,
    pReadDataLog pFRead )
```

To register the log callback function which implemented in the application to monitor sending/reading data between application and reader.

11.1.4.115 SDK_Version()

```
char* SDK_Version ( )
```

To Get SDK version

Returns

return the SDK version string

11.1.4.116 setAbsoluteLibraryPath()

```
int setAbsoluteLibraryPath (
    const char * absoluteLibraryPath )
```

Set the path to use when searching for ID TECH's libraries. If this is not set, the libraries will be searched for with the system's default procedures.

Parameters

<i>absoluteLibraryPath</i>	The absolute path to ID TECH's libraries.
----------------------------	---

Returns

RETURN_CODE: Values can be parsed with [device_getResponseCodeString\(\)](#)

11.2 Source_C/libIDT_Device.h File Reference

Windows C++ API.

```
#include "IDTDef.h"
```

Macros

- #define [IN](#)
- #define [OUT](#)
- #define [IN_OUT](#)

Typedefs

- typedef void(* [pMessageHotplug](#)) (int, int)
- typedef void(* [pSendDataLog](#)) (BYTE *, int)
- typedef void(* [pReadDataLog](#)) (BYTE *, int)
- typedef void(* [pEMV_callBack](#)) (int, int, BYTE *, int, IDTTransactionData *, EMV_Callback *, int)
- typedef void(* [pWP_callBack](#)) (char *, int, int)
- typedef void(* [pWN_callBack](#)) (char *, int, int)
- typedef void(* [pMSR_callBack](#)) (int, IDTMSRData)
- typedef void(* [pMSR_callBackp](#)) (int, IDTMSRData *)
- typedef void(* [pPIN_callBack](#)) (int, IDTPINData *)
- typedef void(* [pLCD_callBack](#)) (int, IDTLCDItem *)
- typedef void(* [pCMR_callBack](#)) (int, IDTCMRData *)
- typedef void(* [pCSFS_callBack](#)) (BYTE status)
- typedef void(* [pFW_callBack](#)) (int, int, int, int, int)
- typedef void(* [pRKI_callBack](#)) (int, char *)
- typedef void(* [httpComm_callBack](#)) (BYTE *, int)
- typedef void(* [v4Comm_callBack](#)) (BYTE, BYTE, BYTE *, int)
- typedef void(* [ftpComm_callBack](#)) (int, int, int)
- typedef void(* [pLog_callback](#)) (BYTE, char *)

Functions

- void [registerHotplugCallBk](#) ([pMessageHotplug](#) pMsgHotplug)
- void [registerLogCallBk](#) ([pSendDataLog](#) pFSend, [pReadDataLog](#) pFRead)
- void [emv_registerCallBk](#) ([pEMV_callBack](#) pEMVf)
- void [loyalty_registerCallBk](#) ([pEMV_callBack](#) pEMVf)
- void [msr_registerCallBk](#) ([pMSR_callBack](#) pMSRf)
- void [msr_registerCallBkp](#) ([pMSR_callBackp](#) pMSRf)
- void [ctls_registerCallBkp](#) ([pMSR_callBackp](#) pMSRf)
- void [pin_registerCallBk](#) ([pPIN_callBack](#) pPINf)
- void [lcd_registerCallBk](#) ([pLCD_callBack](#) pLCDf)
- void [device_registerCameraCallBk](#) ([pCMR_callBack](#) pCMRf)
- void [device_registerCardStatusFrontSwitchCallBk](#) ([pCSFS_callBack](#) pCSFSf)
- void [device_registerFWCallBk](#) ([pFW_callBack](#) pFWf)
- void [device_registerRKICallBk](#) ([pRKI_callBack](#) pRKIf)
- char * [SDK_Version](#) ()
- int [setAbsoluteLibraryPath](#) (const char *absoluteLibraryPath)
- int [device_setConfigPath](#) (const char *path)
- int [device_setNEO2DevicesConfigs](#) (IN const char *configs, IN int len)
- int [device_init](#) ()
- void [device_setNEOGen](#) (int gen)
- void [device_setNEOAltDevice](#) (int alt)
- int [device_getNEOAltDevice](#) ()
- int [rs232_device_init](#) (int deviceType, int port_number, int brate)
- void [set_open_com_port_timeout](#) (int timeout)
- int [device_setCurrentDevice](#) (int deviceType)
- int [device_close](#) ()
- void [device_getResponseCodeString](#) (IN int returnCode, OUT char *despcrition)
- void [device_getIDGStatusCodeString](#) (IN int returnCode, OUT char *despcrition)
- int [device_isConnected](#) ()
- int [device_isAttached](#) (int deviceType)
- int [device_startTransaction](#) (IN double amount, IN double amtOther, IN int type, IN const int _timeout, IN BYTE *tags, IN int tagsLen)

- int [loyalty_startTransaction](#) (IN double amount, IN double amtOther, IN int type, IN const int _timeout, IN BYTE *tags, IN int tagsLen, IN const int cardType, IN const int iccReadType)
- void [device_setTransactionExponent](#) (int exponent)
- int [device_activateTransaction](#) (IN const int _timeout, IN BYTE *tags, IN int tagsLen)
- int [device_cancelTransaction](#) ()
- int [loyalty_cancelTransaction](#) ()
- int [device_setCancelTransactionMode](#) (int mode)
- int [device_cancelTransactionSilent](#) (int enable)
- int [loyalty_cancelTransactionSilent](#) (int enable)
- int [device_getDriveFreeSpace](#) (OUT int *free, OUT int *used)
- int [device_listDirectory](#) (IN char *directoryName, IN int directoryNameLen, IN int recursive, IN int onSD, OUT char *directory, IN_OUT int *directoryLen)
- int [device_createDirectory](#) (IN char *directoryName, IN int directoryNameLen)
- int [device_deleteDirectory](#) (IN char *dirName, IN int dirNameLen)
- int [device_transferFile](#) (IN char *fileName, IN int fileNameLen, IN BYTE *file, IN int fileLen)
- int [device_deleteFile](#) (IN char *fileName, IN int fileNameLen)
- int [device_queryFile](#) (IN char *directoryName, IN int directoryNameLen, IN char *fileName, IN int fileNameLen, OUT int *isExist, OUT BYTE *timeStamp, IN_OUT int *timeStampLen, OUT char *fileSize, IN_OUT int *fileSizeLen)
- int [device_readFileFromSD](#) (IN char *directoryName, IN int directoryNameLen, IN char *fileName, IN int fileNameLen, IN int startingOffset, IN int numBytes, OUT BYTE *fileData, IN_OUT int *fileDataLen)
- int [device_startListenNotifications](#) ()
- int [device_stopListenNotifications](#) ()
- int [device_getFirmwareVersion](#) (OUT char *firmwareVersion)
- int [device_getFirmwareVersion_Len](#) (OUT char *firmwareVersion, IN_OUT int *firmwareVersionLen)
- int [device_getDeviceTreeVersion](#) (OUT char *deviceTreeVersion, IN_OUT int *deviceTreeVersionLen)
- int [device_getDateTime](#) (OUT BYTE *dateTime)
- int [device_getDateTime_Len](#) (OUT BYTE *dateTime, IN_OUT int *dateTimeLen)
- int [device_controlLED](#) (byte indexLED, byte control, int intervalOn, int intervalOff)
- int [device_controlLED_ICC](#) (int controlMode, int interval)
- int [device_controlLED_MSR](#) (byte control, int intervalOn, int intervalOff)
- int [device_controlBeep](#) (int index, int frequency, int duration)
- int [device_getDRS](#) (BYTE *codeDRS, int *codeDRSLen)
- int [device_setCoreDumpLogFile](#) (IN char *filename, IN int filenameLen)
- int [device_outputLog](#) (IN char *filename, IN int filenameLen)
- int [device_getKeyStatus](#) (int *newFormat, BYTE *status, int *statusLen)
- int [device_enterStopMode](#) ()
- int [device_setSleepModeTime](#) (int time)
- int [device_verifyBackdoorKey](#) ()
- int [device_selfCheck](#) ()
- int [device_pingDevice](#) ()
- int [device_controlUserInterface](#) (IN BYTE *values)
- int [device_controlIndicator](#) (IN int indicator, IN int enable)
- int [device_getCurrentDeviceType](#) ()
- int [device_SendDataCommandNEO](#) (IN int cmd, IN int subCmd, IN BYTE *data, IN int dataLen, OUT BYTE *response, IN_OUT int *respLen)
- int [device_SendDataCommand](#) (IN BYTE *cmd, IN int cmdLen, IN BYTE *data, IN int dataLen, OUT BYTE *response, IN_OUT int *respLen)
- int [device_SendDataCommandITP](#) (IN BYTE *cmd, IN int cmdLen, IN BYTE *data, IN int dataLen, OUT BYTE *response, IN_OUT int *respLen)
- int [device_rebootDevice](#) ()
- int [device_updateFirmware](#) (IN BYTE *firmwareData, IN int firmwareDataLen, IN char *firmwareName, IN int encryptionType, IN BYTE *keyBlob, IN int keyBlobLen)
- int [device_getDeviceMemoryUsageInfo](#) (OUT int *freeHeapSize, OUT int *notFreedBlockCnt, OUT int *minEverFreeHeapSize)

- int [device_loadCertCA](#) (IN BYTE CertType, IN BYTE *CACertData, IN int CACertDataLen)
- int [device_rrcDownloadApp](#) (IN char *zipFileName, IN int zipFileNameLen, IN char *appName, IN int appNameLen, IN int appDataLen)
- int [device_rrcUninstallApp](#) (IN char *appName, IN int appNameLen)
- int [device_rrcInstallApp](#) (IN char *appName, IN int appNameLen)
- int [device_rrcRunApp](#) (IN char *appName, IN int appNameLen)
- int [device_rrcConnect](#) ()
- int [device_rrcDisconnect](#) ()
- int [felica_authentication](#) (IN BYTE *key, IN int keyLen)
- int [felica_readWithMac](#) (IN int blockCnt, IN BYTE *blockList, IN int blockListLen, OUT BYTE *blockData, OUT int *blockDataLen)
- int [felica_writeWithMac](#) (IN BYTE blockNum, IN BYTE *blockData, IN int blockDataLen)
- int [felica_read](#) (IN BYTE *serviceCodeList, IN int serviceCodeListLen, IN int blockCnt, IN BYTE *blockList, IN int blockListLen, OUT BYTE *blockData, OUT int *blockDataLen)
- int [felica_write](#) (IN BYTE *serviceCodeList, IN int serviceCodeListLen, IN int blockCnt, IN BYTE *blockList, IN int blockListLen, IN BYTE *blockData, IN int blockDataLen, OUT BYTE *statusFlag, OUT int *statusFlagLen)
- int [felica_poll](#) (IN BYTE *systemCode, IN int systemCodeLen, OUT BYTE *respData, OUT int *respDataLen)
- int [felica_SendCommand](#) (IN BYTE *command, IN int commandLen, OUT BYTE *respData, OUT int *respDataLen)
- int [felica_requestService](#) (IN BYTE *nodeCode, IN int nodeCodeLen, OUT BYTE *respData, OUT int *respDataLen)
- int [felica_getCode](#) ()
- int [felica_cancelCodeEntry](#) ()
- int [config_getModelNumber](#) (OUT char *sNumber)
- int [config_getModelNumber_Len](#) (OUT char *sNumber, IN_OUT int *sNumberLen)
- int [device_setSystemLanguage](#) (char *language)
- int [config_getSerialNumber](#) (OUT char *sNumber)
- int [config_getSerialNumber_Len](#) (OUT char *sNumber, IN_OUT int *sNumberLen)
- int [config_setCmdTimeOutDuration](#) (IN int millisecond)
- int [config_setLEDController](#) (int firmwareControlMSRLED, int firmwareControlICCLEDD)
- int [config_getLEDController](#) (int *firmwareControlMSRLED, int *firmwareControlICCLEDD)
- int [config_setBeeperController](#) (int firmwareControlBeeper)
- int [config_getBeeperController](#) (int *firmwareControlBeeper)
- int [config_setEncryptionControl](#) (int msr, int icc)
- int [config_getEncryptionControl](#) (int *msr, int *icc)
- int [device_startRKI](#) (IN const char *caPath, IN int isProduction)
- void [device_setRKI_URL](#) (IN char *rkiURL, IN int rkiURLLen)
- int [device_enablePassThrough](#) (int enablePassThrough)
- int [device_enableL80PassThrough](#) (int enableL80PassThrough)
- int [device_enableL100PassThrough](#) (int enableL100PassThrough)
- int [device_getL80PassThroughMode](#) ()
- int [device_getL100PassThroughMode](#) ()
- int [device_enhancedPassthrough](#) (IN BYTE *data, IN int dataLen)
- int [device_setBurstMode](#) (IN BYTE mode)
- int [device_setPollMode](#) (IN BYTE mode)
- int [device_pollForToken](#) (IN int timeout, OUT BYTE *respData, IN_OUT int *respDataLen)
- int [device_setMerchantRecord](#) (int index, int enabled, char *merchantID, char *merchantURL)
- int [device_getMerchantRecord](#) (IN int index, OUT BYTE *record)
- int [device_getMerchantRecord_Len](#) (IN int index, OUT BYTE *record, IN_OUT int *recordLen)
- int [device_pollCardReader](#) (OUT BYTE *status)
- int [device_pollCardReader_Len](#) (OUT BYTE *status, IN_OUT int *statusLen)
- int [device_getSpectrumProKSN](#) (IN int type, OUT BYTE *KSN)
- int [device_getSpectrumProKSN_Len](#) (IN int type, OUT BYTE *KSN, IN_OUT int *KSNLen)
- int [device_calibrateParameters](#) (BYTE delta)
- int [device_getRTCTime](#) (IN BYTE *dateTime, IN_OUT int *dateTimeLen)

- int [device_setRTCDateTime](#) (IN BYTE *dateTime, IN int dateTimeLen)
- int [device_configureButtons](#) (IN BYTE done, IN BYTE swipe, IN BYTE delay)
- int [device_getButtonConfiguration](#) (OUT BYTE *done, OUT BYTE *swipe, OUT BYTE *delay)
- int [device_disableBlueLED](#) ()
- int [device_enableBlueLED](#) (IN BYTE *data, IN int dataLen)
- int [device_lcdDisplayClear](#) ()
- int [device_enableExternalLCDMessages](#) (IN int enableExtLCDMsg)
- int [device_enableRFAntenna](#) (IN int enableAntenna)
- int [device_turnOffYellowLED](#) ()
- int [device_turnOnYellowLED](#) ()
- int [device_buzzerOnOff](#) ()
- int [device_lcdDisplayLine1Message](#) (IN BYTE *message, IN int messageLen)
- int [device_lcdDisplayLine2Message](#) (IN BYTE *message, IN int messageLen)
- int [device_startQRCodeScan](#) (IN int _timeout)
- int [device_startQRCodeScanWithDisplayWindowInfo](#) (IN int _timeout, IN int x, IN int y, IN int width, IN int height)
- int [device_stopQRCodeScan](#) ()
- int [device_startTakingPhoto](#) (IN int _timeout)
- int [device_stopTakingPhoto](#) ()
- int [device_stopAudio](#) ()
- int [device_playAudio](#) (IN char *fileName, IN int fileNameLen, IN int onSD)
- int [device_getAudioVolume](#) (OUT BYTE *volume)
- int [device_setAudioVolume](#) (IN BYTE volume)
- int [device_getCameraParameters](#) (OUT BYTE *isAutoFocus, OUT BYTE *focalLength)
- int [device_setCameraParameters](#) (IN BYTE isAutoFocus, IN BYTE focalLength)
- int [device_getSDKWaitTime](#) ()
- void [device_setSDKWaitTime](#) (int waitTime)
- int [device_getThreadStackSize](#) ()
- void [device_setThreadStackSize](#) (int threadSize)
- void [device_toSDCard](#) (int forSDCard)
- int [device_getTamperStatus](#) (OUT int *isTampered)
- int [icc_enable](#) (IN int withNotification)
- int [icc_disable](#) ()
- int [icc_powerOnICC](#) (OUT BYTE *ATR, IN_OUT int *inLen)
- int [icc_powerOffICC](#) ()
- int [icc_exchangeAPDU](#) (IN BYTE *c_APDU, IN int cLen, OUT BYTE *reData, IN_OUT int *reLen)
- int [icc_exchangeEncryptedAPDU](#) (IN BYTE *c_APDU, IN int cLen, OUT BYTE *reData, IN_OUT int *reLen)
- int [icc_getAPDU_KSN](#) (OUT BYTE *KSN, IN_OUT int *inLen)
- int [icc_getFunctionStatus](#) (OUT int *enabled, OUT int *withNotification)
- int [icc_getICCRReaderStatus](#) (OUT BYTE *status)
- int [icc_getKeyFormatForICCDUKPT](#) (OUT BYTE *format)
- int [icc_getKeyTypeForICCDUKPT](#) (OUT BYTE *type)
- int [icc_setKeyFormatForICCDUKPT](#) (IN BYTE format)
- int [icc_setKeyTypeForICCDUKPT](#) (IN BYTE type)
- int [iso8583_get1987Handler](#) (OUT DL_ISO8583_HANDLER *ISOHandler)
- int [iso8583_get1993Handler](#) (OUT DL_ISO8583_HANDLER *ISOHandler)
- int [iso8583_get2003Handler](#) (OUT DL_ISO8583_HANDLER *ISOHandler)
- int [iso8583_getField](#) (IN DL_UINT16 dataField, IN DL_ISO8583_HANDLER *ISOHandler, OUT DL_ISO8583_FIELD_DEF *field)
- int [iso8583_initializeMessage](#) (OUT DL_ISO8583_MSG *ISOMessage)
- int [iso8583_getMessageField](#) (IN DL_UINT16 dataField, IN DL_ISO8583_MSG *ISOMessage, OUT DL_ISO8583_MSG_FIELD *messageField)
- int [iso8583_setMessageField](#) (IN DL_UINT16 dataField, IN const DL_UINT8 *data, OUT DL_ISO8583_MSG *ISOMessage)
- int [iso8583_removeMessageField](#) (IN DL_UINT16 dataField, OUT DL_ISO8583_MSG *ISOMessage)

- int [iso8583_packMessage](#) (IN const DL_ISO8583_HANDLER *ISOHandler, IN const DL_ISO8583_MSG *ISOMessage, OUT DL_UINT8 *packedData, OUT DL_UINT16 *packedDataLength)
- int [iso8583_unpackMessage](#) (IN const DL_ISO8583_HANDLER *ISOHandler, IN const DL_UINT8 *packedData, IN DL_UINT16 packedDataLength, OUT DL_ISO8583_MSG *ISOMessage)
- int [iso8583_freeMessage](#) (IN DL_ISO8583_MSG *ISOMessage)
- int [iso8583_serializeToXML](#) (IN DL_ISO8583_HANDLER *ISOHandler, IN DL_ISO8583_MSG *ISOMessage, OUT BYTE *serializedMessage, OUT int *serializedMessageLength)
- int [iso8583_deserializeFromXML](#) (IN BYTE *serializedMessage, IN int serializedMessageLength, OUT DL_ISO8583_HANDLER *ISOHandler, OUT DL_ISO8583_MSG *ISOMessage)
- int [iso8583_displayMessage](#) (IN DL_ISO8583_HANDLER *ISOHandler, IN DL_ISO8583_MSG *ISOMessage)
- int [lcd_resetInitialState](#) ()
- int [lcd_customDisplayMode](#) (IN int enable)
- int [lcd_setForeBackColor](#) (IN BYTE *foreRGB, IN int foreRGBLen, IN BYTE *backRGB, IN int backRGBLen)
- int [lcd_clearDisplay](#) (IN BYTE control)
- int [lcd_captureSignature](#) (IN int timeout)
- int [lcd_startSlideShow](#) (IN char *files, IN int filesLen, IN int posX, IN int posY, IN int posMode, IN int touchEnable, IN int recursion, IN int touchTerminate, IN int delay, IN int loops, IN int clearScreen)
- int [lcd_cancelSlideShow](#) (OUT BYTE *statusCode, IN_OUT int *statusCodeLen)
- int [lcd_setDisplayImage](#) (IN char *file, IN int fileLen, IN int posX, IN int posY, IN int posMode, IN int touchEnable, IN int clearScreen)
- int [lcd_setBackgroundImage](#) (IN char *file, IN int fileLen, IN int enable)
- int [lcd_displayText](#) (IN int posX, IN int posY, IN int displayWidth, IN int displayHeight, IN int fontDesignation, IN int fontID, IN int screenPosition, IN char *displayText, OUT BYTE *graphicsID)
- int [lcd_displayText_Len](#) (IN int posX, IN int posY, IN int displayWidth, IN int displayHeight, IN int fontDesignation, IN int fontID, IN int screenPosition, IN char *displayText, OUT BYTE *graphicsID, IN_OUT int *graphicsIDLen)
- int [lcd_displayParagraph](#) (IN int posX, IN int posY, IN int displayWidth, IN int displayHeight, IN int fontDesignation, IN int fontID, IN int displayProperties, IN char *displayText)
- int [lcd_displayButton](#) (IN int posX, IN int posY, IN int buttonWidth, IN int buttonHeight, IN int fontDesignation, IN int fontID, IN int displayPosition, IN char *buttonLabel, IN int buttonTextColorR, IN int buttonTextColorG, IN int buttonTextColorB, IN int buttonBackgroundColorR, IN int buttonBackgroundColorG, IN int buttonBackgroundColorB, OUT BYTE *graphicsID)
- int [lcd_displayButton_Len](#) (IN int posX, IN int posY, IN int buttonWidth, IN int buttonHeight, IN int fontDesignation, IN int fontID, IN int displayPosition, IN char *buttonLabel, IN int buttonTextColorR, IN int buttonTextColorG, IN int buttonTextColorB, IN int buttonBackgroundColorR, IN int buttonBackgroundColorG, IN int buttonBackgroundColorB, OUT BYTE *graphicsID, IN_OUT int *graphicsIDLen)
- int [lcd_createList](#) (IN int posX, IN int posY, IN int numColumns, IN int numRows, IN int fontDesignation, IN int fontID, IN int verticalScrollArrowsVisible, IN int borderedListItems, IN int borderedScrollArrows, IN int touchSensitive, IN int automaticScrolling, OUT BYTE *graphicsID)
- int [lcd_createList_Len](#) (IN int posX, IN int posY, IN int numColumns, IN int numRows, IN int fontDesignation, IN int fontID, IN int verticalScrollArrowsVisible, IN int borderedListItems, IN int borderedScrollArrows, IN int touchSensitive, IN int automaticScrolling, OUT BYTE *graphicsID, IN_OUT int *graphicsIDLen)
- int [lcd_addItemToList](#) (IN BYTE *listGraphicsID, IN char *itemName, IN char *itemID, IN int selected)
- int [lcd_getSelectedListItem](#) (IN BYTE *listGraphicsID, OUT char *itemID)
- int [lcd_getSelectedListItem_Len](#) (IN BYTE *listGraphicsID, OUT char *itemID, IN_OUT int *itemIDLen)
- int [lcd_clearEventQueue](#) ()
- int [lcd_getInputEvent](#) (IN int timeout, OUT int *dataReceived, OUT BYTE *eventType, OUT BYTE *graphicsID, OUT BYTE *eventData)
- int [lcd_getInputEvent_Len](#) (IN int timeout, OUT int *dataReceived, OUT BYTE *eventType, IN_OUT int *eventTypeLen, OUT BYTE *graphicsID, IN_OUT int *graphicsIDLen, OUT BYTE *eventData, IN_OUT int *eventDataLen)
- int [lcd_createInputField](#) (IN BYTE *specs, IN int specsLen, OUT BYTE *graphicId)
- int [lcd_createInputField_Len](#) (IN BYTE *specs, IN int specsLen, OUT BYTE *graphicId, IN_OUT int *graphicIdLen)
- int [lcd_getInputFieldValue](#) (IN BYTE *graphicId, OUT BYTE *retData, IN_OUT int *retDataLen)

- int `lcd_createScreen` (IN char *screenName, IN int screenNameLen, OUT int *ScreenID)
- int `lcd_destroyScreen` (IN char *screenName, IN int screenNameLen)
- int `lcd_getActiveScreen` (OUT char *screenName, IN_OUT int *screenNameLen)
- int `lcd_showScreen` (IN char *screenName, IN int screenNameLen)
- int `lcd_getButtonEvent` (OUT int *screenID, OUT int *objectID, OUT char *screenName, IN_OUT int *screenNameLen, OUT char *objectName, IN_OUT int *objectNameLen, OUT int *isLongPress)
- int `lcd_addButton` (IN char *screenName, IN int screenNameLen, IN char *buttonName, IN int buttonNameLen, IN BYTE type, IN BYTE alignment, IN int xCord, IN int yCord, IN char *label, IN int labelLen, OUT IDTLCDItem *returnItem)
- int `lcd_addEthernet` (IN char *screenName, IN int screenNameLen, IN char *objectName, IN int objectNameLen, IN BYTE alignment, IN int xCord, IN int yCord, OUT IDTLCDItem *returnItem)
- int `lcd_addLED` (IN char *screenName, IN int screenNameLen, IN char *objectName, IN int objectNameLen, IN BYTE alignment, IN int xCord, IN int yCord, OUT IDTLCDItem *returnItem, IN BYTE *LED, IN int LEDLen)
- int `lcd_addText` (IN char *screenName, IN int screenNameLen, IN char *objectName, IN int objectNameLen, IN BYTE alignment, IN int xCord, IN int yCord, IN int width, IN int height, IN BYTE fontID, IN BYTE *color, IN int colorLen, IN char *label, IN int labelLen, OUT IDTLCDItem *returnItem)
- int `lcd_addImage` (IN char *screenName, IN int screenNameLen, IN char *objectName, IN int objectNameLen, IN BYTE alignment, IN int xCord, IN int yCord, IN char *filename, IN int filenameLen, OUT IDTLCDItem *returnItem)
- int `lcd_addVideo` (IN char *screenName, IN int screenNameLen, IN char *objectName, IN int objectNameLen, IN BYTE alignment, IN int xCord, IN int yCord, IN char *filename, IN int filenameLen, OUT IDTLCDItem *returnItem)
- int `lcd_addExtVideo` (IN char *screenName, IN int screenNameLen, IN char *objectName, IN int objectNameLen, IN BYTE alignment, IN int xCord, IN int yCord, IN BYTE loop, IN BYTE numVideos, IN char *filenames, IN int filenamesLen, OUT IDTLCDItem *returnItem)
- int `lcd_cloneScreen` (IN char *screenName, IN int screenNameLen, IN char *cloneName, IN int cloneNameLen, OUT int *cloneID)
- int `lcd_updateLabel` (IN char *screenName, IN int screenNameLen, IN char *objectName, IN int objectNameLen, IN char *label, IN int labelLen)
- int `lcd_updateColor` (IN char *screenName, IN int screenNameLen, IN char *objectName, IN int objectNameLen, IN BYTE *color, IN int colorLen)
- int `lcd_updatePosition` (IN char *screenName, IN int screenNameLen, IN char *objectName, IN int objectNameLen, IN BYTE alignment, IN int new_xCord, IN int new_yCord)
- int `lcd_removeItem` (IN char *screenName, IN int screenNameLen, IN char *objectName, IN int objectNameLen)
- int `lcd_storeScreenInfo` ()
- int `lcd_loadScreenInfo` ()
- int `lcd_clearScreenInfo` ()
- int `lcd_getAllScreens` (IN_OUT int *screenNumbers, OUT IDTScreenInfo *screenInfo)
- int `lcd_getAllObjects` (IN char *screenName, IN int screenNameLen, IN_OUT int *objectNumbers, OUT IDTObjectInfo *objectInfo)
- int `lcd_queryScreenByName` (IN char *screenName, IN int screenNameLen, OUT int *result)
- int `lcd_linkUIWithTransactionMessageId` (IN BYTE MessageId, IN char *screenName, IN int screenNameLen)
- int `lcd_queryObjectByName` (IN char *objectName, IN int objectNameLen, IN_OUT int *objectNumbers, OUT IDTScreenInfo *screenInfo)
- int `lcd_queryScreenbyID` (IN int screenID, OUT int *result, OUT int *screenName, IN_OUT int *screenNameLen)
- int `lcd_queryObjectbyID` (IN int objectID, OUT int *objectNumbers, OUT IDTScreenInfo *screenInfo)
- int `lcd_setBacklight` (IN BYTE isBacklightOn, IN BYTE backlightVal)
- int `emv_getEMVKernelVersion` (OUT char *version)
- int `emv_getEMVKernelVersion_Len` (OUT char *version, IN_OUT int *versionLen)
- int `emv_getEMVKernelCheckValue` (OUT BYTE *checkValue, IN_OUT int *checkValueLen)
- int `emv_getEMVConfigurationCheckValue` (OUT BYTE *checkValue, IN_OUT int *checkValueLen)
- void `emv_setAutoAuthenticateTransaction` (IN int authenticate)
- void `emv_setAutoCompleteTransaction` (IN int complete)

- int [emv_getAutoAuthenticateTransaction](#) ()
- int [emv_getAutoCompleteTransaction](#) ()
- void [emv_allowFallback](#) (IN int allow)
- void [emv_setTransactionParameters](#) (IN double amount, IN double amtOther, IN int type, IN int timeout, IN BYTE *tags, IN int tagsLen)
- int [emv_startTransaction](#) (IN double amount, IN double amtOther, IN int exponent, IN int type, IN int timeout, IN BYTE *tags, IN int tagsLen, IN int forceOnline)
- int [emv_activateTransaction](#) (IN int timeout, IN BYTE *tags, IN int tagsLen, IN int forceOnline)
- int [emv_authenticateTransaction](#) (IN BYTE *updatedTLV, IN int updatedTLVLen)
- int [emv_authenticateTransactionWithTimeout](#) (IN int timeout, IN BYTE *updatedTLV, IN int updatedTLVLen)
- int [emv_completeTransaction](#) (IN int commError, IN BYTE *authCode, IN int authCodeLen, IN BYTE *iad, IN int iadLen, IN BYTE *tlvScripts, IN int tlvScriptsLen, IN BYTE *tlv, IN int tlvLen)
- int [emv_cancelTransaction](#) ()
- int [emv_retrieveTransactionResult](#) (IN BYTE *tags, IN int tagsLen, IDTTTransactionData *cardData)
- int [emv_callbackResponseLCD](#) (IN int type, byte selection)
- int [emv_callbackResponseMSR](#) (IN BYTE *MSR, IN_OUT int MSRLen)
- int [emv_retrieveApplicationData](#) (IN BYTE *AID, IN int AIDLen, OUT BYTE *tlv, IN_OUT int *tlvLen)
- int [emv_setApplicationData](#) (IN BYTE *name, IN int nameLen, IN BYTE *tlv, IN int tlvLen)
- int [emv_setApplicationDataTLV](#) (IN BYTE *tlv, IN int tlvLen)
- int [emv_removeApplicationData](#) (IN BYTE *AID, IN int AIDLen)
- int [emv_removeAllApplicationData](#) ()
- int [emv_retrieveAIDList](#) (OUT BYTE *AIDList, IN_OUT int *AIDListLen)
- int [emv_retrieveTerminalData](#) (OUT BYTE *tlv, IN_OUT int *tlvLen)
- int [emv_setTerminalData](#) (IN BYTE *tlv, IN int tlvLen)
- int [emv_setTerminalMajorConfiguration](#) (IN int configuration)
- int [emv_removeTerminalData](#) ()
- int [emv_retrieveCAPK](#) (IN BYTE *capk, IN int capkLen, OUT BYTE *key, IN_OUT int *keyLen)
- int [emv_setCAPK](#) (IN BYTE *capk, IN int capkLen)
- int [emv_removeCAPK](#) (IN BYTE *capk, IN int capkLen)
- int [emv_removeAllCAPK](#) ()
- int [emv_retrieveCAPKList](#) (OUT BYTE *keys, IN_OUT int *keysLen)
- int [emv_retrieveTerminalID](#) (OUT char *terminalID)
- int [emv_retrieveTerminalID_Len](#) (OUT char *terminalID, IN_OUT int *terminalIDLen)
- int [emv_setTerminalID](#) (IN char *terminalID)
- int [emv_retrieveCRL](#) (OUT BYTE *list, IN_OUT int *lssLen)
- int [emv_setCRL](#) (IN BYTE *list, IN int lssLen)
- int [emv_removeCRL](#) (IN BYTE *list, IN int lssLen)
- int [emv_removeAllCRL](#) ()
- int [msr_clearMSRData](#) ()
- int [msr_getMSRData](#) (OUT BYTE *reData, IN_OUT int *reLen)
- int [msr_cancelMSRSwipe](#) ()
- int [msr_startMSRSwipe](#) (IN int _timeout)
- int [executeTransaction](#) (WorldPayData *data, pWP_callback wpCallback, int requestOnly)
- int [forwardTransaction](#) (IN pWP_callback wpCallback, IN char *forwardID, IN int forwardIDLen, IN char *password, IN int passwordLen, IN int bypassProcessing)
- int [cancelWorldPay](#) ()
- int [executeTransaction_WorldNet](#) (WorldNetData *data, pWN_callback wnCallback, int requestOnly)
- int [forwardTransaction_WorldNet](#) (IN char *apiKey, IN int apiKeyLen, IN pWN_callback wnCallback, IN char *forwardID, IN int forwardIDLen, IN char *password, IN int passwordLen, IN int bypassProcessing)
- int [cancelWorldNet](#) ()
- void [printfChar](#) (BYTE *data, int dataLen)
- void [parseMSRData](#) (IN BYTE *resData, IN int resLen, IN_OUT IDTMSRData *cardData)
- int [msr_getKeyFormatForICCDUKPT](#) (OUT BYTE *format)
- int [msr_getKeyTypeForICCDUKPT](#) (OUT BYTE *type)
- int [msr_setKeyFormatForICCDUKPT](#) (IN BYTE format)

- int [msr_setKeyTypeForICCDUKPT](#) (IN BYTE type)
- int [msr_captureMode](#) (int isBufferMode, int withNotification)
- int [msr_flushTrackData](#) ()
- int [msr_setExpirationMask](#) (int mask)
- int [msr_getExpirationMask](#) (BYTE *value)
- int [msr_setClearPANID](#) (BYTE val)
- int [msr_getClearPANID](#) (BYTE *value)
- int [msr_setSwipeForcedEncryptionOption](#) (int track1, int track2, int track3, int track3card0)
- int [msr_getSwipeForcedEncryptionOption](#) (BYTE *option)
- int [msr_setSwipeMaskOption](#) (int track1, int track2, int track3)
- int [msr_getSwipeMaskOption](#) (BYTE *option)
- int [msr_disable](#) ()
- int [msr_getFunctionStatus](#) (int *enable, int *isBufferMode, int *withNotification)
- int [pin_getPIN](#) (IN int mode, IN int PANSource, IN char *iccPAN, IN int iccPANLen, IN int startTimeout, IN int entryTimeout, IN char *language, IN int languageLen)
- int [pin_cancelPINEntry](#) ()
- int [pin_setKeyValues](#) (int mode)
- int [pin_getEncryptedOnlinePIN](#) (IN int keyType, IN int timeout)
- int [pin_getPAN](#) (IN int getCSC, IN int timeout)
- int [pin_promptCreditDebit](#) (IN char *currencySymbol, IN int currencySymbolLen, IN char *displayAmount, IN int displayAmountLen, IN int timeout, OUT BYTE *retData, IN_OUT int *retDataLen)
- int [pin_getEncryptedPIN](#) (int keyType, char *PAN, int PANLen, char *message, int messageLen, int timeout)
- int [pin_promptForKeyInput](#) (int messageID, int languageID, int maskInput, int minLen, int maxLen, int timeout)
- int [pin_promptForAmountInput](#) (int messageID, int languageID, int minLen, int maxLen, int timeout)
- int [pin_getFunctionKey](#) (int timeout)
- int [pin_sendBeep](#) (int frequency, int duration)
- int [pin_capturePin](#) (IN int timeout, IN int type, IN char *PAN, IN int PANLen, IN int minPIN, IN int maxPIN, IN char *message, IN int messageLen)
- int [pin_capturePinExt](#) (IN int type, IN char *PAN, IN int PANLen, IN int minPIN, IN int maxPIN, IN char *message, IN int messageLen, IN char *verify, IN int verifyLen)
- int [pin_promptForNumericKeyWithSwipe](#) (IN int timeout, IN BYTE function, IN int minLen, IN int maxLen, IN char *line1, IN int line1Len, IN char *line2, IN int line2Len, BYTE *signature, IN int signatureLen)
- int [pin_promptForNumericKey](#) (IN int timeout, IN int maskInput, IN int minLen, IN int maxLen, IN char *message, IN int messageLen, BYTE *signature, IN int signatureLen)
- int [pin_inputFromPrompt](#) (BYTE mask, BYTE preClearText, BYTE postClearText, int minLen, int maxLen, char *lang, BYTE promptID, char *defaultResponse, int defaultResponseLen, int timeout)
- int [pin_promptForAmount](#) (IN int timeout, IN int minLen, IN int maxLen, IN char *message, IN int messageLen, BYTE *signature, IN int signatureLen)
- int [pin_getPanEntry](#) (IN int csc, IN int expDate, IN int ADR, IN int ZIP, IN int mod10CK, IN int timeout, IN int encPANOnly)
- int [lcd_savePrompt](#) (int promptNumber, char *prompt, int promptLen)
- int [lcd_displayPrompt](#) (int promptNumber, int lineNumber)
- int [lcd_displayMessage](#) (int lineNumber, char *message, int messageLen)
- int [lcd_enableBacklight](#) (int enable)
- int [lcd_getBacklightStatus](#) (int *enabled)
- int [ws_requestCSR](#) (OUT RequestCSR *csr)
- int [ws_loadSSLCert](#) (IN char *name, IN int nameLen, IN char *dataDER, IN int dataDERLen)
- int [ws_revokeSSLCert](#) (IN char *name, IN int nameLen)
- int [ws_deleteSSLCert](#) (IN char *name, IN int nameLen)
- int [ws_getCertChainType](#) (OUT int *type)
- int [ws_updateRootCertificate](#) (IN char *name, IN int nameLen, IN char *dataDER, IN int dataDERLen, IN char *signature, IN int signatureLen)
- int [ctls_startTransaction](#) (IN double amount, IN double amtOther, IN int type, IN const int _timeout, IN BYTE *tags, IN int tagsLen)
- int [ctls_activateTransaction](#) (IN const int _timeout, IN BYTE *tags, IN int tagsLen)

- int [ctls_cancelTransaction](#) ()
- int [ctls_retrieveApplicationData](#) (IN BYTE *AID, IN int AIDLen, OUT BYTE *tlv, IN_OUT int *tlvLen)
- int [ctls_setApplicationData](#) (IN BYTE *tlv, IN int tlvLen)
- int [ctls_removeApplicationData](#) (IN BYTE *AID, IN int AIDLen)
- int [ctls_removeAllApplicationData](#) ()
- int [ctls_retrieveAIDList](#) (OUT BYTE *AIDList, IN_OUT int *AIDListLen)
- int [ctls_retrieveTerminalData](#) (OUT BYTE *tlv, IN_OUT int *tlvLen)
- int [ctls_setTerminalData](#) (IN BYTE *tlv, IN int tlvLen)
- int [ctls_retrieveCAPK](#) (IN BYTE *capk, IN int capkLen, OUT BYTE *key, IN_OUT int *keyLen)
- int [ctls_setCAPK](#) (IN BYTE *capk, IN int capkLen)
- int [ctls_removeCAPK](#) (IN BYTE *capk, IN int capkLen)
- int [ctls_removeAllCAPK](#) ()
- int [ctls_retrieveCAPKList](#) (OUT BYTE *keys, IN_OUT int *keysLen)
- int [ctls_setConfigurationGroup](#) (IN BYTE *tlv, IN int tlvLen)
- int [ctls_getConfigurationGroup](#) (IN int group, OUT BYTE *tlv, OUT int *tlvLen)
- int [ctls_getAllConfigurationGroups](#) (OUT BYTE *tlv, IN_OUT int *tlvLen)
- int [ctls_removeConfigurationGroup](#) (int group)
- int [ctls_displayOnlineAuthResult](#) (IN int statusCode, IN BYTE *TLV, IN int TLVLen)
- void [parsePINBlockData](#) (IN BYTE *resData, IN int resLen, IN_OUT IDTPINData *cardData)
- void [parsePINData](#) (IN BYTE *resData, IN int resLen, IN_OUT IDTPINData *cardData)

11.2.1 Detailed Description

Windows C++ API.

Windows C++ Global API methods.

11.2.2 Macro Definition Documentation

11.2.2.1 IN

```
#define IN
```

INPUT parameter.

11.2.2.2 IN_OUT

```
#define IN_OUT
```

INPUT / OUTPUT PARAMETER.

11.2.2.3 OUT

```
#define OUT
```

OUTPUT parameter.

11.2.3 Typedef Documentation

11.2.3.1 ftpComm_callBack

```
typedef void(* ftpComm_callBack) (int, int, int)
```

Define the comm callback function to get FTP file transfer status

It should be passed as a parameter in a FTP request, Signature (int, int, int) = response code, current block, total blocks
RESPONSE CODES: 100 = FILE DOWNLOAD STARTED 101 = FILE BLOCK XX OF XX RECEIVED 102 = FILE DOWNLOAD COMPLETED 103 = FILE DOWNLOAD TERMINATED PREMATURELY

11.2.3.2 httpComm_callBack

```
typedef void(* httpComm_callBack) (BYTE *, int)
```

Define the comm callback function to get the async url data

It should be registered using the comm_registerHTTPCallback,

11.2.3.3 pCMR_callBack

```
typedef void(* pCMR_callBack) (int, IDTCMRData *)
```

Define the camera callback function to get the image data

It should be registered using the device_registerCameraCallbk,

11.2.3.4 pCSFS_callBack

```
typedef void(* pCSFS_callBack) (BYTE status)
```

Define the card status and front switch callback function to get card and front switch status

It should be registered using the device_registerCardStatusFrontSwitchCallbk,

11.2.3.5 pEMV_callBack

```
typedef void(* pEMV_callBack) (int, int, BYTE *, int, IDTTransactionData *, EMV_Callback *, int)
```

Define the EMV callback function to get the transaction message/data/result.

It should be registered using the emv_registerCallbk,

11.2.3.6 pFW_callBack

```
typedef void(* pFW_callBack) (int, int, int, int, int)
```

Define the FW callback function to get the status of the firmware update

It should be registered using the device_registerFWCallbk,

11.2.3.7 pLCD_callBack

```
typedef void(* pLCD_callBack) (int, IDTLCDItem *)
```

Define the LCD callback function to get the input LCDItem

It should be registered using the lcd_registerCallbk,

11.2.3.8 pLog_callback

```
typedef void(* pLog_callback) (BYTE, char *)
```

Define the log callback function to receive log messages.

11.2.3.9 pMessageHotplug

```
typedef void(* pMessageHotplug) (int, int)
```

Define the USB hot-plug callback function to monitor the info when plug in/out the reader.

It should be registered using the registerHotplugCallBk, The first integer parameter is device type, and the second integer parameter is either 0: Device Plugged Out or 1: Device Plugged In

11.2.3.10 pMSR_callBack

```
typedef void(* pMSR_callBack) (int, IDTMSRData)
```

Define the MSR callback function to get the MSR card data

It should be registered using the msr_registerCallBk, this callback function is for backward compatibility

11.2.3.11 pMSR_callBackp

```
typedef void(* pMSR_callBackp) (int, IDTMSRData *)
```

Define the MSR callback function to get pointer to the MSR card data

It should be registered using the msr_registerCallBk, this callback function is recommended instead of pMSR_callBack

11.2.3.12 pPIN_callBack

```
typedef void(* pPIN_callBack) (int, IDTPINData *)
```

Define the PINPad callback function to get the input PIN Pad data

It should be registered using the pin_registerCallBk,

11.2.3.13 pReadDataLog

```
typedef void(* pReadDataLog) (BYTE *, int)
```

Define the read response callback function to monitor the reading response from the reader.

It should be registered using the registerLogCallBk,

11.2.3.14 pRKI_callBack

```
typedef void(* pRKI_callBack) (int, char *)
```

Define the RKI callback function to get the status of the RKI

It should be registered using the device_registerRKICallBk,

11.2.3.15 pSendDataLog

```
typedef void(* pSendDataLog) (BYTE *, int)
```

Define the send command callback function to monitor the sending command into the reader.

It should be registered using the registerLogCallBk,

11.2.3.16 pWN_callBack

```
typedef void(* pWN_callBack) (char *, int, int)
```

Define the Worldnet callback function to get the transaction message/data/result.

11.2.3.17 pWP_callBack

```
typedef void(* pWP_callBack) (char *, int, int)
```

Define the Worldpay callback function to get the transaction message/data/result.

11.2.3.18 v4Comm_callBack

```
typedef void(* v4Comm_callBack) (BYTE, BYTE, BYTE *, int)
```

Define the comm callback function to receive the V4 Protocol packets received by the device from an external source (IP/USB/RS-232) It should be registered using the `comm_registerV4Callback`, Data callback will contain command, sub-command, and data from V4 packet

11.2.4 Function Documentation

11.2.4.1 cancelWorldNet()

```
int cancelWorldNet ( )
```

Cancels WorldNet transaction.

Returns

RETURN_CODE: Values can be parsed with [device_getIDGStatusCodeString\(\)](#)

11.2.4.2 cancelWorldPay()

```
int cancelWorldPay ( )
```

Cancels WorldPay transaction.

Returns

RETURN_CODE: Values can be parsed with [device_getIDGStatusCodeString\(\)](#)

11.2.4.3 config_getBeeperController()

```
int config_getBeeperController (
    int * firmwareControlBeeper )
```

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

Parameters

<i>firmwareControlBeeper</i>	1 means firmware control the beeper, 0 means software control beeper.
------------------------------	---

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString

11.2.4.4 config_getEncryptionControl()

```
int config_getEncryptionControl (
    int * msr,
    int * icc )
```

Get Encryption Control - AUGUSTA

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"> • 1: enabled MSR with Encryption, • 0: disabled MSR with Encryption,
<i>icc</i>	<ul style="list-style-type: none"> • 1: enabled ICC with Encryption, • 0: disabled ICC with Encryption,

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString

11.2.4.5 config_getLEDController()

```
int config_getLEDController (
    int * firmwareControlMSRLED,
    int * firmwareControlICCLED )
```

Get the LED Controller Status - AUGUSTA Get the MSR / ICC LED controlled status by software or firmware NOTE: The ICC LED always controlled by software.

Parameters

<i>firmwareControlMSRLED</i>	<ul style="list-style-type: none"> • 1: firmware control the MSR LED • 0: software control the MSR LED
------------------------------	--

Parameters

<i>firmwareControlICCLED</i>	<ul style="list-style-type: none"> • 1: firmware control the ICC LED • 0: software control the ICC LED
------------------------------	--

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString

11.2.4.6 config_getModelNumber()

```
int config_getModelNumber (
    OUT char * sNumber )
```

DEPRECATED : please use [config_getModelNumber_Len\(OUT char* sNumber, IN_OUT int *sNumberLen\)](#)

Polls device for Model Number

Parameters

<i>sNumber</i>	Returns Model Number; needs to have at least 64 bytes of memory
----------------	---

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString

11.2.4.7 config_getModelNumber_Len()

```
int config_getModelNumber_Len (
    OUT char * sNumber,
    IN_OUT int * sNumberLen )
```

Polls device for Model Number

Parameters

<i>sNumber</i>	Returns Model Number
<i>sNumber</i>	length of Model Number

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString

11.2.4.8 config_getSerialNumber()

```
int config_getSerialNumber (
    OUT char * sNumber )
```

DEPRECATED : please use `config_getSerialNumber_Len(OUT char* sNumber, IN_OUT int *sNumberLen)`

Polls device for Serial Number

Parameters

<i>sNumber</i>	Returns Serial Number; needs to have at least 64 bytes of memory
----------------	--

Returns

RETURN_CODE: Values can be parsed with `device_getResponseCodeString`

11.2.4.9 config_getSerialNumber_Len()

```
int config_getSerialNumber_Len (
    OUT char * sNumber,
    IN_OUT int * sNumberLen )
```

Polls device for Serial Number

Parameters

<i>sNumber</i>	Returns Serial Number
<i>sNumberLen</i>	Length of Serial Number

Returns

RETURN_CODE: Values can be parsed with `device_getResponseCodeString`

11.2.4.10 config_setBeeperController()

```
int config_setBeeperController (
    int firmwareControlBeeper )
```

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

Parameters

<i>firmwareControlBeeper</i>	1 means firmware control the beeper, 0 means software control beeper.
------------------------------	---

Returns

RETURN_CODE: Values can be parsed with `device_getResponseCodeString`

11.2.4.11 config_setCmdTimeOutDuration()

```
int config_setCmdTimeOutDuration (
    IN int millisecond )
```

Set the timeout duration for regular commands. The new timeout value will affect all the functions actually send (sync) commands that doesn't need to wait for a callback function, such as [device_getFirmwareVersion\(\)](#), [device_pingDevice\(\)](#), [device_SendDataCommandNEO\(\)](#), [device_enablePassThrough\(\)](#), [device_setBurstMode\(\)](#), [device_setPollMode\(\)](#), [device_updateFirmware\(\)](#) ... etc.

Parameters

<i>millisecond</i>	timeout value in milliseconds. Please consult the firmware team for the proper value.
--------------------	---

Returns

RETURN_CODE: Values can be parsed with [device_getIDGStatusCodeString](#)

11.2.4.12 config_setEncryptionControl()

```
int config_setEncryptionControl (
    int msr,
    int icc )
```

Set Encryption Control - AUGUSTA

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"> • 1: enable MSR with Encryption, • 0: disable MSR with Encryption,
<i>icc</i>	<ul style="list-style-type: none"> • 1: enable ICC with Encryption, • 0: disable ICC with Encryption,

Returns

RETURN_CODE: Values can be parsed with [device_getResponseCodeString](#)

11.2.4.13 config_setLEDController()

```
int config_setLEDController (
    int firmwareControlMSRLED,
    int firmwareControlICCLEDD )
```

Set the LED Controller - AUGUSTA 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"> • 1: firmware control the MSR LED • 0: software control the MSR LED
<i>firmwareControlICCLED</i>	<ul style="list-style-type: none"> • 1: firmware control the ICC LED • 0: software control the ICC LED

Returns

RETURN_CODE: Values can be parsed with `device_getResponseCodeString`

11.2.4.14 `ctls_activateTransaction()`

```
int ctls_activateTransaction (
    IN const int _timeout,
    IN BYTE * tags,
    IN int tagsLen )
```

Start CTLS Transaction Request

Authorizes the CTLS 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 stream. Example, tag 9F02 with amount 0x000000000100 would be 0x9F0206000000000100
<i>tagsLen</i>	The length of tags data buffer.

>>>>>IMPORTANT: parameters for amount and amtOther MUST BE PASSED AS A DOUBLE VALUE WITH DECIMAL POINT. Example, do not pass 1, but instead pass 1.0 or 1.00. Otherwise, results will be unpredictable

Returns

RETURN_CODE: Values can be parsed with [device_getIDGStatusCodeString\(\)](#) Note: if auto poll is on, it will return the error IDG_P2_STATUS_CODE_COMMAND_NOT_ALLOWED

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

11.2.4.15 `ctls_cancelTransaction()`

```
int ctls_cancelTransaction ( )
```

Cancel EMV Transaction

Cancels the currently executing EMV transaction.

Returns

RETURN_CODE: Values can be parsed with [device_getIDGStatusCodeString\(\)](#)

11.2.4.16 ctls_displayOnlineAuthResult()

```
int ctls_displayOnlineAuthResult (
    IN int statusCode,
    IN BYTE * TLV,
    IN int TLVLen )
```

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. The SDK timeout of the command is set to 7 seconds.

Parameters

<i>statusCode</i>	1 = OK, 0 = NOT OK, 2 = ARC response 89 for Interac
<i>TLV</i>	Optional TLV for AOSA
<i>TLVLen</i>	TLV Length

Returns

RETURN_CODE: Values can be parsed with [device_getIDGStatusCodeString\(\)](#)

11.2.4.17 ctls_getAllConfigurationGroups()

```
int ctls_getAllConfigurationGroups (
    OUT BYTE * tlv,
    IN_OUT int * tlvLen )
```

Retrieve All Configuration Groups

Returns all the Configuration Groups installed on the terminal for CTLS

Parameters

<i>tlv</i>	The TLV elements data
<i>tlvLen</i>	the length of tlv data buffer.

Returns

RETURN_CODE: Values can be parsed with [device_getIDGStatusCodeString\(\)](#)

11.2.4.18 ctls_getConfigurationGroup()

```
int ctls_getConfigurationGroup (
    IN int group,
    OUT BYTE * tlv,
    OUT int * tlvLen )
```

Get Configuration Group

Retrieves the Configuration for the specified Group.

Parameters

<i>group</i>	Configuration Group
--------------	---------------------

Parameters

<i>tlv</i>	return data
<i>tlvLen</i>	the length of tlv data buffer

Returns

RETURN_CODE: Values can be parsed with [device_getIDGStatusCodeString\(\)](#)

11.2.4.19 ctls_registerCallBkp()

```
void ctls_registerCallBkp (
    pMSR_callback pMSRf )
```

To register the ctls callback function to get the CTLS card data pointer. It's the same as [msr_registerCallBkp\(\)](#). (Pass NULL to disable the callback.)

11.2.4.20 ctls_removeAllApplicationData()

```
int ctls_removeAllApplicationData ( )
```

Remove All Application Data

Removes all the Application Data

Returns

RETURN_CODE: Values can be parsed with [device_getIDGStatusCodeString\(\)](#)

11.2.4.21 ctls_removeAllCAPK()

```
int ctls_removeAllCAPK ( )
```

Remove All Certificate Authority Public Key

Removes all the CAPK

Returns

RETURN_CODE: Values can be parsed with [device_getIDGStatusCodeString\(\)](#)

11.2.4.22 ctls_removeApplicationData()

```
int ctls_removeApplicationData (
    IN BYTE * AID,
    IN int AIDLen )
```

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>AIDLen</i>	the length of AID data buffer

Returns

RETURN_CODE: Values can be parsed with [device_getIDGStatusCodeString\(\)](#)

11.2.4.23 ctls_removeCAPK()

```
int ctls_removeCAPK (
    IN BYTE * capk,
    IN int capkLen )
```

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>capkLen</i>	the length of capk data buffer

Returns

RETURN_CODE: Values can be parsed with [device_getIDGStatusCodeString\(\)](#)

11.2.4.24 ctls_removeConfigurationGroup()

```
int ctls_removeConfigurationGroup (
    int group )
```

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>	Values can be parsed with device_getIDGStatusCodeString()
--------------------	---

11.2.4.25 ctls_retrieveAIDList()

```
int ctls_retrieveAIDList (
```



```
OUT BYTE * AIDList,  
IN_OUT int * AIDListLen )
```

Retrieve AID list

Returns all the AID names installed on the terminal for CTLS. .

Parameters

<i>AIDList</i>	array of AID name byte arrays
<i>AIDListLen</i>	the length of AIDList array buffer

Returns

RETURN_CODE: Values can be parsed with [device_getIDGStatusCodeString\(\)](#)

11.2.4.26 ctls_retrieveApplicationData()

```
int ctls_retrieveApplicationData (  
    IN BYTE * AID,  
    IN int AIDLen,  
    OUT BYTE * tlv,  
    IN_OUT int * tlvLen )
```

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>AIDLen</i>	the length of AID data buffer.
<i>tlv</i>	The TLV elements of the requested AID
<i>tlvLen</i>	the length of tlv data buffer.

Returns

RETURN_CODE: Values can be parsed with [device_getIDGStatusCodeString\(\)](#)

11.2.4.27 ctls_retrieveCAPK()

```
int ctls_retrieveCAPK (  
    IN BYTE * capk,  
    IN int capkLen,  
    OUT BYTE * key,  
    IN_OUT int * keyLen )
```

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>capkLen</i>	the length of capk data buffer
<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>keyLen</i>	the length of key data buffer

Returns

RETURN_CODE: Values can be parsed with [device_getIDGStatusCodeString\(\)](#)

11.2.4.28 `ctls_retrieveCAPKList()`

```
int ctls_retrieveCAPKList (
    OUT BYTE * keys,
    IN_OUT int * keysLen )
```

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>keysLen</i>	the length of keys data buffer

Returns

RETURN_CODE: Values can be parsed with [device_getIDGStatusCodeString\(\)](#)

11.2.4.29 `ctls_retrieveTerminalData()`

```
int ctls_retrieveTerminalData (
    OUT BYTE * tlv,
    IN_OUT int * tlvLen )
```

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>tlvLen</i>	the length of tlv data buffer

Returns

RETURN_CODE: Values can be parsed with [device_getIDGStatusCodeString\(\)](#)

11.2.4.30 `ctls_setApplicationData()`

```
int ctls_setApplicationData (  
    IN BYTE * tlv,  
    IN int tlvLen )
```

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>tlvLen</i>	the length of tlv data buffer
---------------	-------------------------------

Returns

RETURN_CODE: Values can be parsed with [device_getIDGStatusCodeString\(\)](#)

11.2.4.31 `ctls_setCAPK()`

```
int ctls_setCAPK (  
    IN BYTE * capk,  
    IN int capkLen )
```

Set Certificate Authority Public Key

Sets the CAPK as specified by the CAKey structure

Parameters

<i>capk</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>capkLen</i>	the length of capk data buffer

Returns

RETURN_CODE: Values can be parsed with [device_getIDGStatusCodeString\(\)](#)

11.2.4.32 **ctls_setConfigurationGroup()**

```
int ctls_setConfigurationGroup (
    IN BYTE * tlv,
    IN int tlvLen )
```

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 or DFEE2D). A second tag must exist
<i>tlvLen</i>	the length of tlv data buffer

Returns

RETURN_CODE: Values can be parsed with [device_getIDGStatusCodeString\(\)](#)

11.2.4.33 **ctls_setTerminalData()**

```
int ctls_setTerminalData (
    IN BYTE * tlv,
    IN int tlvLen )
```

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 `ctls_getConfigurationGroup(int group)`, and deleted with `ctls_removeConfigurationGroup(int group)`. You cannot delete group 0

Parameters

<i>tlv</i>	TerminalData configuration file
<i>tlvLen</i>	the length of tlv data buffer

Return values

<i>RETURN_CODE</i>	Values can be parsed with <code>device_getIDGStatusCodeString()</code>
--------------------	--

11.2.4.34 ctls_startTransaction()

```
int ctls_startTransaction (
    IN double amount,
    IN double amtOther,
    IN int type,
    IN const int _timeout,
    IN BYTE * tags,
    IN int tagsLen )
```

Start CTLS Transaction Request

Authorizes the CTLS transaction for an ICC card

The tags will be returned in the callback routine.

Parameters

<i>amount</i>	Transaction amount value (tag value 9F02) - SEE IMPORTANT NOTE BELOW
<i>amtOther</i>	Other amount value, if any (tag value 9F03) - SEE IMPORTANT NOTE BELOW
<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 stream. Example, tag 9F0C with amount 0x000000000100 would be 0x9F0C06000000000100 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>tagsLen</i>	The length of tags data buffer.

>>>>>IMPORTANT: parameters for amount and amtOther MUST BE PASSED AS A DOUBLE VALUE WITH DECIMAL POINT. Example, do not pass 1, but instead pass 1.0 or 1.00. Otherwise, results will be unpredictable

Returns

RETURN_CODE: Values can be parsed with `device_getIDGStatusCodeString()` Note: if auto poll is on, it will return the error IDG_P2_STATUS_CODE_COMMAND_NOT_ALLOWED

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) 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

11.2.4.35 device_activateTransaction()

```
int device_activateTransaction (
    IN const int _timeout,
    IN BYTE * tags,
    IN int tagsLen )
```

Start Transaction Request

Authorizes the transaction for an MSR/CTLS/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. Example, tag 9F02 with amount 0x000000000100 would be 0x9F0206000000000100 Be sure to include 9F02 (amount)and9C (transaction type).
<i>tagsLen</i>	The length of tags data buffer.

>>>>>IMPORTANT: parameters for amount and amtOther MUST BE PASSED AS A DOUBLE VALUE WITH DECIMAL POINT. Example, do not pass 1, but instead pass 1.0 or 1.00. Otherwise, results will be unpredictable

Returns

RETURN_CODE: Values can be parsed with [device_getIDGStatusCodeString\(\)](#) Note: if auto poll is on, it will return the error IDG_P2_STATUS_CODE_COMMAND_NOT_ALLOWED

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 device_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

11.2.4.36 `device_buzzerOnOff()`

```
int device_buzzerOnOff ( )
```

Use this function to make the buzzer beep once

Returns

RETURN_CODE: Values can be parsed with [device_getIDGStatusCodeString\(\)](#)

11.2.4.37 `device_calibrateParameters()`

```
int device_calibrateParameters (
    BYTE delta )
```

Calibrate reference parameters

Parameters

<i>delta</i>	Delta value (0x02 standard default value)
--------------	---

Returns

RETURN_CODE: Values can be parsed with [device_getResponseCodeString\(\)](#)

11.2.4.38 `device_cancelTransaction()`

```
int device_cancelTransaction ( )
```

Cancel Transaction

Cancels the currently executing transaction.

Returns

RETURN_CODE: Values can be parsed with [device_getIDGStatusCodeString\(\)](#)

11.2.4.39 `device_cancelTransactionSilent()`

```
int device_cancelTransactionSilent (
    int enable )
```

Cancel Transaction Silent

Cancel transaction with or without showing the LCD message

Parameters

<i>enable</i>	0: With LCD message 1: Without LCD message
---------------	--

Returns

success or error code. Values can be parsed with `device_getIDGStatusCodeString`

11.2.4.40 `device_close()`

```
int device_close ( )
```

Close the device

Returns

RETURN_CODE: 0: success, 0x0A: failed

11.2.4.41 `device_configureButtons()`

```
int device_configureButtons (
    IN BYTE done,
    IN BYTE swipe,
    IN BYTE delay )
```

Configures the buttons on the ViVOpay Vendi reader

Parameters

<i>done</i>	0x01: the Done switch is enabled 0x00: the Done switch is disabled
<i>swipe</i>	0x01: the Swipe Card switch is enabled 0x00: the Swipe Card switch is disabled
<i>delay</i>	an unsigned delay value (<= 30) in seconds

Returns

RETURN_CODE: Values can be parsed with [device_getIDGStatusCodeString\(\)](#)

11.2.4.42 `device_controlBeep()`

```
int device_controlBeep (
    int index,
    int frequency,
    int duration )
```

Control Beep - AUGUSTA

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`

11.2.4.43 device_controlIndicator()

```
int device_controlIndicator (
    IN int indicator,
    IN int enable )
```

Control Indicators

Control the reader. If connected, returns success. Otherwise, returns timeout.

Parameters

<i>indicator</i>	description as follows: <ul style="list-style-type: none">• 00h: ICC LED• 01h: Blue MSR• 02h: Red MSR• 03h: Green MSR
<i>enable</i>	TRUE = ON, FALSE = OFF

Returns

success or error code. Values can be parsed with `device_getResponseCodeString`

See also

`ErrorCode`

11.2.4.44 device_controlLED()

```
int device_controlLED (
    byte indexLED,
    byte control,
    int intervalOn,
    int intervalOff )
```

Control MSR LED - AUGUSTA

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)

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString

intervalOn = 500, int intervalOff = 500

11.2.4.45 device_controlLED_ICC()

```
int device_controlLED_ICC (
    int controlMode,
    int interval )
```

Control ICC LED - AUGUSTA

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

11.2.4.46 device_controlLED_MSR()

```
int device_controlLED_MSR (
    byte control,
    int intervalOn,
    int intervalOff )
```

Control the MSR LED - AUGUSTA

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

int intervalOn = 500, int intervalOff = 500)

11.2.4.47 device_controlUserInterface()

```
int device_controlUserInterface (  
    IN BYTE * values )
```

Control User Interface - NEO only

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) • 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
---------------	--

Returns

RETURN_CODE: Values can be parsed with [device_getIDGStatusCodeString\(\)](#)

11.2.4.48 device_createDirectory()

```
int device_createDirectory (
    IN char * directoryName,
    IN int directoryNameLen )
```

Create Directory This command adds a subdirectory to the indicated path.

Parameters

<i>directoryName</i>	Directory Name. The data for this command is 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>directoryNameLen</i>	Directory Name Length.

Returns

RETURN_CODE: Values can be parsed with [device_getIDGStatusCodeString\(\)](#)

11.2.4.49 device_deleteDirectory()

```
int device_deleteDirectory (
    IN char * dirName,
    IN int dirNameLen )
```

Delete Directory This command deletes an empty directory. For NEO 2 devices, it will delete the directory even the directory is not empty.

Parameters

<i>dirName</i>	Complete path of the directory you want to delete. You do not need to specify the root directory. Indicate subdirectories with a forward slash (/). For NEO 2 devices, to delete the root directory, simply pass "" with 0 for dirNameLen.
<i>dirNameLen</i>	Directory Name Length.

Returns

RETURN_CODE: Values can be parsed with [device_getIDGStatusCodeString\(\)](#)

11.2.4.50 device_deleteFile()

```
int device_deleteFile (
    IN char * fileName,
    IN int fileNameLen )
```

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>filenameLen</i>	File Name Length.

Returns

RETURN_CODE: Values can be parsed with [device_getIDGStatusCodeString\(\)](#)

11.2.4.51 device_disableBlueLED()

```
int device_disableBlueLED ( )
```

Stops the blue LEDs on the ViVOpay Vendi reader from flashing in left to right sequence and turns the LEDs off, and contactless function is disable at the same time

Returns

RETURN_CODE: Values can be parsed with [device_getIDGStatusCodeString\(\)](#)

11.2.4.52 device_enableBlueLED()

```
int device_enableBlueLED (
    IN BYTE * data,
    IN int dataLen )
```

Use this function to control the blue LED behavior on the Vendi reader

Parameters

<i>data</i>	Sequence data Byte 0 (Cycle): 0 = Cycle once, 1 = Repeat Byte 1 (LEDs): LED State Bitmap Byte 2-3 (Duration): Given in multiples of 10 millisecond Byte 4 (LED): LED State Bitmap Byte 5-6 (Duration): Given in multiples of 10 millisecond Byte 7-24 (Additional LED/Durations): Define up to 8 LED and duration pairs
-------------	---

LED State Bitmap: Bit 8: Left blue LED, 0 = off, 1 = on Bit 7: Center Blue LED, 0 = off, 1 = on Bit 6: Right Blue LED 0 = off, 1 = on Bit 5: Yellow LED, 0 = off, 1 = on Bit 4: Reserved for future use Bit 3: Reserved for future use Bit 2: Reserved for future use Bit 1: Reserved for future use

Parameters

<i>dataLen</i>	Length of the sequence data: 0 or 4 to 25 bytes
----------------	---

Returns

RETURN_CODE: Values can be parsed with [device_getIDGStatusCodeString\(\)](#)

11.2.4.53 device_enableExternalLCDMessages()

```
int device_enableExternalLCDMessages (
    IN int enableExtLCDMsg )
```

Enable or disable the external LCD message It will turn off the external LCD messages including EMV transactions. (For the users who only need MSR and/or CTLS transactions.) The function only works for VP5300

Parameters

<i>enableExtLCDMsg</i>	1 = ON, 0 = OFF
------------------------	-----------------

Returns

success or error code. Values can be parsed with `device_getIDGStatusCodeString`

See also

`ErrorCode`

11.2.4.54 device_enableL100PassThrough()

```
int device_enableL100PassThrough (
    int enableL100PassThrough )
```

Enable L100 Pass Through

Enables Pass Through Mode for direct communication to L100 hook up to NEO II device

Parameters

<i>enableL100PassThrough</i>	1 = pass through ON, 0 = pass through OFF
------------------------------	---

Returns

RETURN_CODE: Values can be parsed with [device_getResponseCodeString\(\)](#)

11.2.4.55 device_enableL80PassThrough()

```
int device_enableL80PassThrough (
    int enableL80PassThrough )
```

Enable L80 Pass Through

Enables Pass Through Mode for direct communication to L80 hook up to NEO II device

Parameters

<i>enableL80PassThrough</i>	1 = pass through ON, 0 = pass through OFF
-----------------------------	---

Returns

RETURN_CODE: Values can be parsed with [device_getResponseCodeString\(\)](#)

11.2.4.56 device_enablePassThrough()

```
int device_enablePassThrough (
```



```
int enablePassThrough )
```

Enable Pass Through - NEO

Enables Pass Through Mode for direct communication with L1 interface (power on icc, send apdu, etc).

Parameters

<i>enablePassThrough</i>	1 = pass through ON, 0 = pass through OFF
--------------------------	---

Returns

RETURN_CODE: Values can be parsed with [device_getIDGStatusCodeString\(\)](#)

11.2.4.57 device_enableRFAntenna()

```
int device_enableRFAntenna (
    IN int enableAntenna )
```

Enable or disable the RF Antenna

Parameters

<i>enableAntenna</i>	1 = ON, 0 = OFF
----------------------	-----------------

Returns

success or error code. Values can be parsed with [device_getIDGStatusCodeString](#)

See also

[ErrorCode](#)

11.2.4.58 device_enhancedPassthrough()

```
int device_enhancedPassthrough (
    IN BYTE * data,
    IN int dataLen )
```

Enables pass through mode for ICC. Required when direct ICC commands are required (power on/off ICC, exchange APDU)

Parameters

<i>data</i>	The data includes Poll Timeout, Flags, Contact Interface to Use, Beep Indicator, LED Status, and Display Strings.
<i>dataLen</i>	length of data

Returns

success or error code. Values can be parsed with [device_getIDGStatusCodeString](#)

See also

[ErrorCode](#)

11.2.4.59 device_enterStopMode()

```
int device_enterStopMode ( )
```

Enter Stop Mode

Set device enter to stop 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.

Returns

RETURN_CODE: Values can be parsed with [device_getResponseCodeString](#)

11.2.4.60 device_getAudioVolume()

```
int device_getAudioVolume (
    OUT BYTE * volume )
```

Get Audio Volume This command retrieves the reader's audio volume.

Parameters

<i>volume</i>	Value 0-20, where 0 is silent and 20 is full volume
---------------	---

Returns

RETURN_CODE: Values can be parsed with [device_getIDGStatusCodeString\(\)](#)

11.2.4.61 device_getButtonConfiguration()

```
int device_getButtonConfiguration (
    OUT BYTE * done,
    OUT BYTE * swipe,
    OUT BYTE * delay )
```

Reads the button configuration from the ViVOpay Vendi reader

Parameters

<i>done</i>	0x01: the Done switch is enabled 0x00: the Done switch is disabled
<i>swipe</i>	0x01: the Swipe Card switch is enabled 0x00: the Swipe Card switch is disabled
<i>delay</i>	an unsigned delay value in seconds

Returns

RETURN_CODE: Values can be parsed with [device_getIDGStatusCodeString\(\)](#)

11.2.4.62 device_getCameraParameters()

```
int device_getCameraParameters (
    OUT BYTE * isAutoFocus,
    OUT BYTE * focalLength )
```

Get Camera Parameters This command is used to get the camera parameters (e.g., auto/fixed focal length as focus).

Parameters

<i>isAutoFocus</i>	0: fixed focus, 1: auto focus
<i>focalLength</i>	focal length Value 0x00-0xFF, where 0x00 is the farthest, 0xFF is nearest. Not used for auto focus.

Returns

RETURN_CODE: Values can be parsed with [device_getIDGStatusCodeString\(\)](#)

11.2.4.63 device_getCurrentDeviceType()

```
int device_getCurrentDeviceType ( )
```

Get current active device type

Returns

: return the device type defined as DEVICE_TYPE in the IDTDef.h

11.2.4.64 device_getDateTime()

```
int device_getDateTime (
    OUT BYTE * dateTime )
```

DEPRECATED : please use [device_getDateTime_Len\(OUT BYTE* dateTime, IN_OUT int *dateTimeLen\)](#)

Polls device for Date and Time

Parameters

<i>dateTime</i>	Response returned of Date and Time; needs to have at least 6 bytes of memory
-----------------	--

Returns

RETURN_CODE: Values can be parsed with [device_getResponseCodeString\(\)](#)

11.2.4.65 device_getDateTime_Len()

```
int device_getDateTime_Len (
    OUT BYTE * dateTime,
    IN_OUT int * dateTimeLen )
```

Polls device for Date and Time

Parameters

<i>dateTime</i>	Response returned of Date and Time
<i>dateTime</i>	Length of Date and Time

Returns

RETURN_CODE: Values can be parsed with [device_getResponseCodeString\(\)](#)

11.2.4.66 device_getDeviceMemoryUsageInfo()

```
int device_getDeviceMemoryUsageInfo (
    OUT int * freeHeapSize,
    OUT int * notFreedBlockCnt,
    OUT int * minEverFreeHeapSize )
```

Update Firmware with zip file Updates the firmware of NEO 2 devices.

Parameters

<i>firmwareZipFilename</i>	Firmware zip file name. <ul style="list-style-type: none"> For example "package_VP6300 FW v1.01.003.0432.T.zip"
<i>firmwareZipFilenameLen</i>	Firmware zip file name length

Returns

RETURN_CODE: Values can be parsed with [device_getIDGStatusCodeString\(\)](#)

Firmware update status is returned in the callback with the following values: sender = device type state = DEVICE_FIRMWARE_UPDATE current block total blocks ResultCode:

- RETURN_CODE_DO_SUCCESS = Firmware Update Completed Successfully
- Any other return code represents an error condition Note: to call this function under Windows, the executable file "unzip.exe" is required Get Device Memory Usage Information

Parameters

<i>freeHeapSize</i>	Free Heap Size: Available heap size
<i>notFreedBlockCnt</i>	Memory Not Freed Block Count: Memory in use block count
<i>minEverFreeHeapSize</i>	Minimum Ever Free Heap Size: The lowest ever available heap size

Returns

RETURN_CODE: Values can be parsed with [device_getIDGStatusCodeString\(\)](#)

11.2.4.67 device_getDeviceTreeVersion()

```
int device_getDeviceTreeVersion (
    OUT char * deviceTreeVersion,
    IN_OUT int * deviceTreeVersionLen )
```

Polls device for Device Tree Version

Parameters

<i>deviceTreeVersion</i>	Response returned of Device Tree Version
<i>deviceTreeVersionLen</i>	Length of Device Tree Version

Returns

RETURN_CODE: Values can be parsed with [device_getResponseCodeString\(\)](#)

11.2.4.68 device_getDriveFreeSpace()

```
int device_getDriveFreeSpace (
    OUT int * free,
    OUT int * used )
```

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

Returns

RETURN_CODE: Values can be parsed with [device_getIDGStatusCodeString\(\)](#)

11.2.4.69 device_getDRS()

```
int device_getDRS (
    BYTE * codeDRS,
    int * codeDRSLen )
```

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

Parameters

<i>codeDRSLen</i>	the length of codeDRS
-------------------	-----------------------

Returns

RETURN_CODE: Values can be parsed with [device_getResponseCodeString\(\)](#) Note: Only support TTK devices

11.2.4.70 `device_getFirmwareVersion()`

```
int device_getFirmwareVersion (
    OUT char * firmwareVersion )
```

DEPRECATED : please use [device_getFirmwareVersion_Len\(\)](#)(OUT char* firmwareVersion, IN_OUT int *firmwareVersionLen)

Polls device for Firmware Version

Parameters

<i>firmwareVersion</i>	Response returned of Firmware Version; needs to have at least 128 bytes of memory
------------------------	---

Returns

RETURN_CODE: Values can be parsed with [device_getResponseCodeString\(\)](#)

11.2.4.71 `device_getFirmwareVersion_Len()`

```
int device_getFirmwareVersion_Len (
    OUT char * firmwareVersion,
    IN_OUT int * firmwareVersionLen )
```

Polls device for Firmware Version

Parameters

<i>firmwareVersion</i>	Response returned of Firmware Version
------------------------	---------------------------------------

Parameters

<i>firmwareVersionLen</i>	Length of Firmware Version
---------------------------	----------------------------

Returns

RETURN_CODE: Values can be parsed with [device_getResponseCodeString\(\)](#)

11.2.4.72 device_getIDGStatusCodeString()

```
void device_getIDGStatusCodeString (
    IN int returnCode,
    OUT char * despcrition )
```

Review the return code description.

Parameters

<i>returnCode</i>	the response result.
<i>description</i>	

Return values

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

- 0: "no error, beginning task";
- 1: "no response from reader";
- 2: "invalid response data";
- 01: " Incorrect Header Tag";
- 02: " Unknown Command";
- 03: " Unknown Sub-Command";
- 04: " CRC Error in Frame";
- 05: " Incorrect Parameter";
- 06: " Parameter Not Supported";
- 07: " Mal-formatted Data";
- 08: " Timeout";
- 0A: " Failed / NACK";
- 0B: " Command not Allowed";
- 0C: " Sub-Command not Allowed";
- 0D: " Buffer Overflow (Data Length too large for reader buffer)";
- 0E: " User Interface Event";

- 10: " Need clear firmware(apply in boot loader only)";
- 11: " Communication type not supported, VT-1, burst, etc. Need encrypted firmware (apply in boot loader only)";
- 12: " Secure interface is not functional or is in an intermediate state.";
- 13: " Data field is not mod 8";
- 14: " Pad 0x80 not found where expected";
- 15: " Specified key type is invalid";
- 16: " Could not retrieve key from the SAM (InitSecureComm)";
- 17: " Hash code problem";
- 18: " Could not store the key into the SAM (InstallKey)";
- 19: " Frame is too large";
- 1A: " Unit powered up in authentication state but POS must resend the InitSecureComm command";
- 1B: " The EEPROM may not be initialized because SecCommInterface does not make sense";
- 1C: " Problem encoding APDU Module-Specific Status Codes ";
- 20: " Unsupported Index (ILM) SAM Transceiver error - problem communicating with the SAM (Key Mgr)";
- 21: " Unexpected Sequence Counter in multiple frames for single bitmap (ILM)Length error in data returned from the SAM (Key Mgr) - 22: " Improper bit map (ILM)"; - 23: " Request Online Authorization"; - 24: " ViV← OCard3 raw data read successful"; - 25: " Message index not available (ILM) ViVOcomm activate transaction card type (ViVOcomm)"; - 26: " Version Information Mismatch (ILM)"; - 27: " Not sending commands in correct index message index (ILM)"; - 28: " Time out or next expected message not received (ILM)"; - 29: " ILM languages not available for viewing (ILM)"; - 2A: " Other language not supported (ILM)"; - 41: " from 41 to 4F, Module-specific errors for Key Manager";
- 50: " Auto-Switch OK";
- 51: " Auto-Switch failed";
- 70: " Antenna Error 80h Use another card";
- 81: " Insert or swipe card";
- 90: " Data encryption Key does not exist";
- 91: " Data encryption Key KSN exhausted";

11.2.4.73 device_getKeyStatus()

```
int device_getKeyStatus (
    int * newFormat,
    BYTE * status,
    int * statusLen )
```

Get Key Status

Gets the status of loaded keys

Parameters

<i>status</i>	newFormat for Augusta and miniSmartII only 1: new format of key status 0: reserved format for support previous device
---------------	---

Parameters

<i>status</i>	For L80, L100, Augusta and miniSmartII: When the newFormat is 0, data format as follows. For Augusta and miniSmartII: 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 For L80 and L100: byte 0: PIN DUKPT Key byte 1: PIN Master Key byte 2: Standard PIN Session Key byte 3: Desjardins PIN Session Key byte 4: Account/MSR DUKPT Key, 1 Exists, 0 None, 0xFF STOP, Does not support, always 0 byte 5: Account/ICC DUKPT Key, 1 Exists, 0 None, 0xFF STOP, Does not support, always 0 byte 6: Admin DUKPT Key, 1 Exists, 0 None, 0xFF STOP byte 7: Data DUKPT Key, 1 Exists, 0 None, 0xFF STOP byte 8: MAC DUKPT Key, 1 Exists, 0 None, 0xFF STOP
---------------	--

when the newFormat is 1, 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) For NEO2 and SREDKey2: Each unit of three bytes represents one key's parameters (index and slot). Key Name Index (1 byte): 0x14 - LCL-KEK 0x01 - Pin encryption Key (NEO2 only) 0x02 - Data encryption Key 0x05 - MAC DUKPT Key 0x0A - PCI Pairing Key (NEO2 only) Key Slot (2 bytes): Indicate different slots of a certain Key Name Example: slot =5 (0x00 0x05), slot=300 (0x01 0x2C) For BTPay380, slot is always 0 For example, 0x14 0x00 0x00 0x02 0x00 0x00 0x0A 0x00 0x00 will represent [KeyNameIndex=0x14,KeySlot=0x0000], [KeyNameIndex=0x02,KeySlot=0x0000] and [KeyNameIndex=0x0A,KeySlot=0x0000]

Parameters

<i>statusLen</i>	the length of status
------------------	----------------------

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString

11.2.4.74 device_getL100PassThroughMode()

```
int device_getL100PassThroughMode ( )
```

Get L100 Pass Through Mode

Get current Pass Through Mode for direct communication to L100 hook up to NEO II device

Returns

RETURN_CODE: return 1 if L100 Pass Through Mode is TRUE, 0 if L100 Pass Through Mode is FALSE

11.2.4.75 device_getL80PassThroughMode()

```
int device_getL80PassThroughMode ( )
```

Get L80 Pass Through Mode

Get current Pass Through Mode for direct communication to L80 hook up to NEO II device

Returns

RETURN_CODE: return 1 if L80 Pass Through Mode is TRUE, 0 if L80 Pass Through Mode is FALSE

11.2.4.76 device_getMerchantRecord()

```
int device_getMerchantRecord (
    IN int index,
    OUT BYTE * record )
```

DEPRECATED : please use [device_getMerchantRecord_Len](#)(IN int index, OUT BYTE * record, IN_OUT int *recordLen)

Get Merchant Record

Gets the merchant record for the device.

Parameters

<i>index</i>	Merchant Record index, valid values 1-6
<i>record;</i>	needs to have at least 99 bytes of memory response data from reader. Merchant Record Index: 1 byte enabled: 1 byte Merchant ID: 32 bytes Length of Merchant URL: 1 byte Merchant URL: 64 bytes

Returns

success or error code. Values can be parsed with [device_getIDGStatusCodeString\(\)](#)

See also

[ErrorCode](#)

11.2.4.77 device_getMerchantRecord_Len()

```
int device_getMerchantRecord_Len (
    IN int index,
    OUT BYTE * record,
    IN_OUT int * recordLen )
```

Get Merchant Record

Gets the merchant record for the device.

Parameters

<i>index</i>	Merchant Record index, valid values 1-6
<i>record</i>	response data from reader. Merchant Record Index: 1 byte enabled: 1 byte Merchant ID: 32 bytes Length of Merchant URL: 1 byte Merchant URL: 64 bytes
<i>recordLen</i>	Length of record

Returns

success or error code. Values can be parsed with [device_getIDGStatusCodeString\(\)](#)

See also

ErrorCode

11.2.4.78 device_getNEOAltDevice()

```
int device_getNEOAltDevice ( )
```

Get the alternative device type for the NEO 2 or 3 readers

Returns

RETURN_CODE: The alternative device type of the NEO 2/3 readers.

11.2.4.79 device_getResponseCodeString()

```
void device_getResponseCodeString (
    IN int returnCode,
    OUT char * description )
```

Review the return code description.

Parameters

<i>returnCode</i>	the response result.
<i>description</i>	

Return values

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

- 0: "no error, beginning task";
- 1: "no response from reader";
- 2: "invalid response data";
- 3: "time out for task or CMD";
- 4: "wrong parameter";
- 5: "SDK is doing MSR or ICC task";
- 6: "SDK is doing PINPad task";
- 7: "SDK is doing CTLS task";
- 8: "SDK is doing EMV task";
- 9: "SDK is doing Other task";
- 10: "err response or data";
- 11: "no reader attached";

- 12: "mono audio is enabled";
- 13: "did connection";
- 14: "audio volume is too low";
- 15: "task or CMD be canceled";
- 16: "UF wrong string format";
- 17: "UF file not found";
- 18: "UF wrong file format";
- 19: "Attempt to contact online host failed";
- 20: "Attempt to perform RKI failed";
- 22: "Buffer size is not enough";
- 0X300: "Key Type(TDES) of Session Key is not same as the related Master Key.";
- 0X400: "Related Key was not loaded.";
- 0X500: "Key Same.";
- 0X501: "Key is all zero";
- 0X502: "TR-31 format error";
- 0X702: "PAN is Error Key.";
- 0X705: "No Internal MSR PAN (or Internal MSR PAN is erased timeout)";
- 0X0C01: "Incorrect Frame Tag";
- 0X0C02: "Incorrect Frame Type";
- 0X0C03: "Unknown Frame Type";
- 0X0C04: "Unknown Command";
- 0X0C05: "Unknown Sub-Command";
- 0X0C06: "CRC Error";
- 0X0C07: "Failed";
- 0X0C08: "Timeout";
- 0X0C0A: "Incorrect Parameter";
- 0X0C0B: "Command Not Supported";
- 0X0C0C: "Sub-Command Not Supported";
- 0X0C0D: "Parameter Not Supported / Status Abort Command";
- 0X0C0F: "Sub-Command Not Allowed";
- 0X0D01: "Incorrect Header Tag";
- 0X0D02: "Unknown Command";
- 0X0D03: "Unknown Sub-Command";
- 0X0D04: "CRC Error in Frame";
- 0X0D05: "Incorrect Parameter";
- 0X0D06: "Parameter Not Supported";

- 0X0D07: "Mal-formatted Data";
- 0X0D08: "Timeout";
- 0X0D0A: "Failed / NACK";
- 0X0D0B: "Command not Allowed";
- 0X0D0C: "Sub-Command not Allowed";
- 0X0D0D: "Buffer Overflow (Data Length too large for reader buffer)";
- 0X0D0E: "User Interface Event";
- 0X0D11: "Communication type not supported, VT-1, burst, etc.";
- 0X0D12: "Secure interface is not functional or is in an intermediate state.";
- 0X0D13: "Data field is not mod 8";
- 0X0D14: "Pad - 0X80 not found where expected";
- 0X0D15: "Specified key type is invalid";
- 0X0D1: "Could not retrieve key from the SAM(InitSecureComm)";
- 0X0D17: "Hash code problem";
- 0X0D18: "Could not store the key into the SAM(InstallKey)";
- 0X0D19: "Frame is too large";
- 0X0D1A: "Unit powered up in authentication state but POS must resend the InitSecureComm command";
- 0X0D1B: "The EEPROM may not be initialized because SecCommInterface does not make sense";
- 0X0D1C: "Problem encoding APDU";
- 0X0D20: "Unsupported Index(ILM) SAM Transceiver error - problem communicating with the SAM(Key Mgr)";
- 0X0D2: "Unexpected Sequence Counter in multiple frames for single bitmap(ILM) Length error in data returned from the SAM(Key Mgr)";
- 0X0D22: "Improper bit map(ILM)";
- 0X0D23: "Request Online Authorization";
- 0X0D24: "ViVOCard3 raw data read successful";
- 0X0D25: "Message index not available(ILM) ViVOcomm activate transaction card type(ViVOcomm)";
- 0X0D26: "Version Information Mismatch(ILM)";
- 0X0D27: "Not sending commands in correct index message index(ILM)";
- 0X0D28: "Time out or next expected message not received(ILM)";
- 0X0D29: "ILM languages not available for viewing(ILM)";
- 0X0D2A: "Other language not supported(ILM)";
- 0X0D41: "Unknown Error from SAM";
- 0X0D42: "Invalid data detected by SAM";
- 0X0D43: "Incomplete data detected by SAM";
- 0X0D44: "Reserved";
- 0X0D45: "Invalid key hash algorithm";

- 0X0D46: "Invalid key encryption algorithm";
- 0X0D47: "Invalid modulus length";
- 0X0D48: "Invalid exponent";
- 0X0D49: "Key already exists";
- 0X0D4A: "No space for new RID";
- 0X0D4B: "Key not found";
- 0X0D4C: "Crypto not responding";
- 0X0D4D: "Crypto communication error";
- 0X0D4E: "Module-specific error for Key Manager";
- 0X0D4F: "All key slots are full (maximum number of keys has been installed)";
- 0X0D50: "Auto-Switch OK";
- 0X0D51: "Auto-Switch failed";
- 0X0D90: "Account DUKPT Key not exist";
- 0X0D91: "Account DUKPT Key KSN exhausted";
- 0X0D00: "This Key had been loaded.";
- 0X0E00: "Base Time was loaded.";
- 0X0F00: "Encryption Or Decryption Failed.";
- 0X1000: "Battery Low Warning (It is High Priority Response while Battery is Low.)";
- 0X1800: "Send 'Cancel Command' after send 'Get Encrypted PIN' & 'Get Numeric' & 'Get Amount'; - 0↵
- X1900: "Press 'Cancel' key after send 'Get Encrypted PIN' & 'Get Numeric' & 'Get Amount';
- 0X30FF: "Security Chip is not connect";
- 0X3000: "Security Chip is deactivation & Device is In Removal Legally State.";
- 0X3101: "Security Chip is activation & Device is In Removal Legally State.";
- 0X5500: "No Admin DUKPT Key.";
- 0X5501: "Admin DUKPT Key STOP.";
- 0X5502: "Admin DUKPT Key KSN is Error.";
- 0X5503: "Get Authentication Code1 Failed.";
- 0X5504: "Validate Authentication Code Error.";
- 0X5505: "Encrypt or Decrypt data failed.";
- 0X5506: "Not Support the New Key Type.";
- 0X5507: "New Key Index is Error.";
- 0X5508: "Step Error.";
- 0X5509: "KSN Error";
- 0X550A: "MAC Error.";
- 0X550B: "Key Usage Error.";
- 0X550C: "Mode Of Use Error.";

- 0X550F: "Other Error.";
- 0X6000: "Save or Config Failed / Or Read Config Error.";
- 0X6200: "No Serial Number.";
- 0X6900: "Invalid Command - Protocol is right, but task ID is invalid.";
- 0X6A01: "Unsupported Command - Protocol and task ID are right, but command is invalid - In this State";
- 0X6A00: "Unsupported Command - Protocol and task ID are right, but command is invalid.";
- 0X6B00: "Unknown parameter in command - Protocol task ID and command are right, but parameter is invalid.";
- 0X6C00: "Unknown parameter in command - Protocol task ID and command are right, but length is out of the requirement.";
- 0X7200: "Device is suspend (MKSK suspend or press password suspend).";
- 0X7300: "PIN DUKPT is STOP (21 bit 1).";
- 0X7400: "Device is Busy.";
- 0XE100: "Can not enter sleep mode";
- 0XE200: "File has existed";
- 0XE300: "File has not existed";
- 0XE313: "IO line low -- Card error after session start";
- 0XE400: "Open File Error";
- 0XE500: "SmartCard Error";
- 0XE600: "Get MSR Card data is error";
- 0XE700: "Command time out";
- 0XE800: "File read or write is error";
- 0XE900: "Active 1850 error!";
- 0XEA00: "Load bootloader error";
- 0XEF00: "Protocol Error- STX or ETX or check error.";
- 0XEB00: "Picture is not exist";
- 0X2C02: "No Microprocessor ICC seated";
- 0X2C06: "no card seated to request ATR";
- 0X2D01: "Card Not Supported, ";
- 0X2D03: "Card Not Supported, wants CRC";
- 0X690D: "Command not supported on reader without ICC support";
- 0X8100: "ICC error time out on power-up";
- 0X8200: "invalid TS character received - Wrong operation step";
- 0X8300: "Decode MSR Error";
- 0X8400: "TriMagII no Response";
- 0X8500: "No Swipe MSR Card";
- 0X8510: "No Financial Card";

- 0X8600: "Unsupported F, D, or combination of F and D";
- 0X8700: "protocol not supported EMV TD1 out of range";
- 0X8800: "power not at proper level";
- 0X8900: "ATR length too long";
- 0X8B01: "EMV invalid TA1 byte value";
- 0X8B02: "EMV TB1 required";
- 0X8B03: "EMV Unsupported TB1 only 00 allowed";
- 0X8B04: "EMV Card Error, invalid BWI or CWI";
- 0X8B06: "EMV TB2 not allowed in ATR";
- 0X8B07: "EMV TC2 out of range";
- 0X8B08: "EMV TC2 out of range";
- 0X8B09: "per EMV96 TA3 must be > - 0XF";
- 0X8B10: "ICC error on power-up";
- 0X8B11: "EMV T=1 then TB3 required";
- 0X8B12: "Card Error, invalid BWI or CWI";
- 0X8B13: "Card Error, invalid BWI or CWI";
- 0X8B17: "EMV TC1/TB3 conflict-";
- 0X8B20: "EMV TD2 out of range must be T=1";
- 0X8C00: "TCK error";
- 0XA304: "connector has no voltage setting";
- 0XA305: "ICC error on power-up invalid (SBLK(IFSD) exchange";
- 0XE301: "ICC error after session start";
- 0XFF00: "Request to go online";
- 0XFF01: "EMV: Accept the offline transaction";
- 0XFF02: "EMV: Decline the offline transaction";
- 0XFF03: "EMV: Accept the online transaction";
- 0XFF04: "EMV: Decline the online transaction";
- 0XFF05: "EMV: Application may fallback to magstripe technology";
- 0XFF06: "EMV: ICC detected tah the conditions of use are not satisfied";
- 0XFF07: "EMV: ICC didn't accept transaction";
- 0XFF08: "EMV: Transaction was cancelled";
- 0XFF09: "EMV: Application was not selected by kernel or ICC format error or ICC missing data error";
- 0XFF0A: "EMV: Transaction is terminated";
- 0XFF0B: "EMV: Other EMV Error";
- 0XFFFF: "NO RESPONSE";
- 0XF002: "ICC communication timeout";

- 0XF003: "ICC communication Error";
- 0XF00F: "ICC Card Seated and Highest Priority, disable MSR work request";
- 0XF200: "AID List / Application Data is not exist";
- 0XF201: "Terminal Data is not exist";
- 0XF202: "TLV format is error";
- 0XF203: "AID List is full";
- 0XF204: "Any CA Key is not exist";
- 0XF205: "CA Key RID is not exist";
- 0XF206: "CA Key Index it not exist";
- 0XF207: "CA Key is full";
- 0XF208: "CA Key Hash Value is Error";
- 0XF209: "Transaction format error";
- 0XF20A: "The command will not be processing";
- 0XF20B: "CRL is not exist";
- 0XF20C: "CRL number exceed max number";
- 0XF20D: "Amount, Other Amount, Trasaction Type are missing";
- 0XF20E: "The Identification of algorithm is mistake";
- 0XF20F: "No Financial Card";
- 0XF210: "In Encrypt Result state, TLV total Length is greater than Max Length";
- 0X1001: "INVALID ARG";
- 0X1002: "FILE_OPEN_FAILED";
- 0X1003: "FILE OPERATION_FAILED";
- 0X2001: "MEMORY_NOT_ENOUGH";
- 0X3002: "SMARTCARD_FAIL";
- 0X3003: "SMARTCARD_INIT_FAILED";
- 0X3004: "FALLBACK_SITUATION";
- 0X3005: "SMARTCARD_ABSENT";
- 0X3006: "SMARTCARD_TIMEOUT";
- 0X3012: "EMV_RESULT_CODE_MSR_CARD_ERROR_FALLBACK";
- 0X5001: "EMV_PARSING_TAGS_FAILED";
- 0X5002: "EMV_DUPLICATE_CARD_DATA_ELEMENT";
- 0X5003: "EMV_DATA_FORMAT_INCORRECT";
- 0X5004: "EMV_NO_TERM_APP";
- 0X5005: "EMV_NO_MATCHING_APP";
- 0X5006: "EMV_MISSING_MANDATORY_OBJECT";
- 0X5007: "EMV_APP_SELECTION_RETRY";

- 0X5008: "EMV_GET_AMOUNT_ERROR";
- 0X5009: "EMV_CARD_REJECTED";
- 0X5010: "EMV_AIP_NOT_RECEIVED";
- 0X5011: "EMV_AFL_NOT_RECEIVED";
- 0X5012: "EMV_AFL_LEN_OUT_OF_RANGE";
- 0X5013: "EMV_SFI_OUT_OF_RANGE";
- 0X5014: "EMV_AFL_INCORRECT";
- 0X5015: "EMV_EXP_DATE_INCORRECT";
- 0X5016: "EMV_EFF_DATE_INCORRECT";
- 0X5017: "EMV_ISS_COD_TBL_OUT_OF_RANGE";
- 0X5018: "EMV_CRYPTOGRAM_TYPE_INCORRECT";
- 0X5019: "EMV_PSE_NOT_SUPPORTED_BY_CARD";
- 0X5020: "EMV_USER_SELECTED_LANGUAGE";
- 0X5021: "EMV_SERVICE_NOT_ALLOWED";
- 0X5022: "EMV_NO_TAG_FOUND";
- 0X5023: "EMV_CARD_BLOCKED";
- 0X5024: "EMV_LEN_INCORRECT";
- 0X5025: "CARD_COM_ERROR";
- 0X5026: "EMV_TSC_NOT_INCREASED";
- 0X5027: "EMV_HASH_INCORRECT";
- 0X5028: "EMV_NO_ARC";
- 0X5029: "EMV_INVALID_ARC";
- 0X5030: "EMV_NO_ONLINE_COMM";
- 0X5031: "TRAN_TYPE_INCORRECT";
- 0X5032: "EMV_APP_NO_SUPPORT";
- 0X5033: "EMV_APP_NOT_SELECT";
- 0X5034: "EMV_LANG_NOT_SELECT";
- 0X5035: "EMV_NO_TERM_DATA";
- 0X5039: "EMV_PIN_ENTRY_TIMEOUT";
- 0X6001: "CVM_TYPE_UNKNOWN";
- 0X6002: "CVM_AIP_NOT_SUPPORTED";
- 0X6003: "CVM_TAG_8E_MISSING";
- 0X6004: "CVM_TAG_8E_FORMAT_ERROR";
- 0X6005: "CVM_CODE_IS_NOT_SUPPORTED";
- 0X6006: "CVM_COND_CODE_IS_NOT_SUPPORTED";
- 0X6007: "NO_MORE_CVM";

- 0X6008: "PIN_BYPASSED_BEFORE";
- 0X7001: "PK_BUFFER_SIZE_TOO_BIG";
- 0X7002: "PK_FILE_WRITE_ERROR";
- 0X7003: "PK_HASH_ERROR";
- 0X8001: "NO_CARD_HOLDER_CONFIRMATION";
- 0X8002: "GET_ONLINE_PIN";
- 0XD000: "Data not exist";
- 0XD001: "Data access error";
- 0XD100: "RID not exist";
- 0XD101: "RID existed";
- 0XD102: "Index not exist";
- 0XD200: "Maximum exceeded";
- 0XD201: "Hash error";
- 0XD205: "System Busy";
- 0X0E01: "Unable to go online";
- 0X0E02: "Technical Issue";
- 0X0E03: "Declined";
- 0X0E04: "Issuer Referral transaction";
- 0X0F01: "Decline the online transaction";
- 0X0F02: "Request to go online";
- 0X0F03: "Transaction is terminated";
- 0X0F05: "Application was not selected by kernel or ICC format error or ICC missing data error";
- 0X0F07: "ICC didn't accept transaction";
- 0X0F0A: "Application may fallback to magstripe technology";
- 0X0F0C: "Transaction was cancelled";
- 0X0F0D: "Timeout";
- 0X0F0F: "Other EMV Error";
- 0X0F10: "Accept the offline transaction";
- 0X0F11: "Decline the offline transaction";
- 0X0F21: "ICC detected that the conditions of use are not satisfied";
- 0X0F22: "No app were found on card matching terminal configuration";
- 0X0F23: "Terminal file does not exist";
- 0X0F24: "CAPK file does not exist";
- 0X0F25: "CRL Entry does not exist";
- 0X0FFE: "code when blocking is disabled";
- 0X0FFF: "code when command is not applicable on the selected device";

- 0XF005: "ICC Encrypted C-APDU Data Structure Length Error Or Format Error.";
- 0XBBE0: "CM100 Success";
- 0XBBE1: "CM100 Parameter Error";
- 0XBBE2: "CM100 Low Output Buffer";
- 0XBBE3: "CM100 Card Not Found";
- 0XBBE4: "CM100 Collision Card Exists";
- 0XBBE5: "CM100 Too Many Cards Exist";
- 0XBBE6: "CM100 Saved Data Does Not Exist";
- 0XBBE8: "CM100 No Data Available";
- 0XBBE9: "CM100 Invalid CID Returned";
- 0XBBEA: "CM100 Invalid Card Exists";
- 0XBBE C: "CM100 Command Unsupported";
- 0XBBED: "CM100 Error In Command Process";
- 0XBBE E: "CM100 Invalid Command";
- 0X9031: "Unknown command";
- 0X9032: "Wrong parameter (such as the length of the command is incorrect)";
- 0X9038: "Wait (the command couldnt be finished in BWT)";
- 0X9039: "Busy (a previously command has not been finished)";
- 0X903A: "Number of retries over limit";
- 0X9040: "Invalid Manufacturing system data";
- 0X9041: "Not authenticated";
- 0X9042: "Invalid Master DUKPT Key";
- 0X9043: "Invalid MAC Key";
- 0X9044: "Reserved for future use";
- 0X9045: "Reserved for future use";
- 0X9046: "Invalid DATA DUKPT Key";
- 0X9047: "Invalid PIN Pairing DUKPT Key";
- 0X9048: "Invalid DATA Pairing DUKPT Key";
- 0X9049: "No nonce generated";
- 0X9949: "No GUID available. Perform getVersion first.";
- 0X9950: "MAC Calculation unsuccessful. Check BDk value.";
- 0X904A: "Not ready";
- 0X904B: "Not MAC data";
- 0X9050: "Invalid Certificate";
- 0X9051: "Duplicate key detected";
- 0X9052: "AT checks failed";

- 0X9053: "TR34 checks failed";
- 0X9054: "TR31 checks failed";
- 0X9055: "MAC checks failed";
- 0X9056: "Firmware download failed";
- 0X9060: "Log is full";
- 0X9061: "Removal sensor unengaged";
- 0X9062: "Any hardware problems";
- 0X9070: "ICC communication timeout";
- 0X9071: "ICC data error (such check sum error)";
- 0X9072: "Smart Card not powered up";

11.2.4.80 device_getRTCDateTime()

```
int device_getRTCDateTime (
    IN BYTE * dateTime,
    IN_OUT int * dateTimeLen )
```

get RTC date and time of the device

Parameters

<i>dateTime</i>	<dateTime data>=""> is: 6 byte data for YYMMDDHHMMSS in hex. For example 0x171003102547 stands for 2017 Oct 3rd 10:25:47
<i>dateTimeLen</i>	return 6 bytes if successful

Returns

success or error code. Values can be parsed with device_getResponseCodeString

See also

ErrorCode

11.2.4.81 device_getSDKWaitTime()

```
int device_getSDKWaitTime ( )
```

Get SDK Wait Time

Get the SDK wait time for transactions

Returns

SDK wait time in seconds

11.2.4.82 device_getSpectrumProKSN()

```
int device_getSpectrumProKSN (
    IN int type,
    OUT BYTE * KSN )
```

DEPRECATED : please use [device_getSpectrumProKSN_Len\(IN int type, OUT BYTE * KSN, IN_OUT int *KSNLen\)](#)

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) • 2: Data Encryption Key (DEK) • 5: MAC Key (MAK) • 10: RKL Key Encryption Key (REK) • 20: HSM DUKPT Key
<i>KSN</i>	Key Serial Number; needs to have at least 10 bytes of memory

Returns

RETURN_CODE: Values can be parsed with [device_getResponseCodeString\(\)](#)

11.2.4.83 device_getSpectrumProKSN_Len()

```
int device_getSpectrumProKSN_Len (
    IN int type,
    OUT BYTE * KSN,
    IN_OUT int * KSNLen )
```

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) • 2: Data Encryption Key (DEK) • 5: MAC Key (MAK) • 10: RKL Key Encryption Key (REK) • 20: HSM DUKPT Key
<i>KSN</i>	Key Serial Number
<i>KSNLen</i>	Length of KSN

Returns

RETURN_CODE: Values can be parsed with [device_getResponseCodeString\(\)](#)

11.2.4.84 device_getTamperStatus()

```
int device_getTamperStatus (
    OUT int * isTampered )
```

Get Tamper Status This command check the tamper status for NEO 2 readers.

Parameters

<i>isTampered</i>	1: the reader is tampered. 0: the reader is not tampered.
-------------------	---

Returns

RETURN_CODE: Values can be parsed with [device_getIDGStatusCodeString\(\)](#)

11.2.4.85 device_getThreadStackSize()

```
int device_getThreadStackSize ( )
```

Get Thread Stack Size

Get the stack size setting for newly created threads

Returns

Thread Stack Size

11.2.4.86 device_init()

```
int device_init ( )
```

Initial the device by USB

It will detect the device and trying connect.

The connect status can be checked by [device_isConnected\(\)](#).

Note: after the function returns success, the function [device_setCurrentDevice\(\)](#) has to be called to set the device type.

Returns

RETURN_CODE: Values can be parsed with [device_getIDGStatusCodeString\(\)](#)

11.2.4.87 device_isAttached()

```
int device_isAttached (
    int deviceType )
```

Check if the device is attached to the USB port The function [device_init\(\)](#) must be called before this function.

Parameters

<i>deviceType,the</i>	device type of the USB device
-----------------------	-------------------------------

Returns

1 if the device is attached, or 0 if the device is not attached

11.2.4.88 device_isConnected()

```
int device_isConnected ( )
```

Check the device connected status

Returns

DEVICE_DISCONNECT=0, or DEVICE_CONNECTED = 1

11.2.4.89 device_lcdDisplayClear()

```
int device_lcdDisplayClear ( )
```

Use this function to clear the LCD display

Returns

RETURN_CODE: Values can be parsed with [device_getIDGStatusCodeString\(\)](#)

11.2.4.90 device_lcdDisplayLine1Message()

```
int device_lcdDisplayLine1Message (
    IN BYTE * message,
    IN int messageLen )
```

Use this function to display text on the LCD display. On the Vendi reader the LCD is a 2-line character display.

Parameters

<i>message</i>	Valid messages for the first line of text are between 1 and 16 printable characters long. If the text message is greater than 16 bytes but not more than 32 bytes, byte 17 and onward are displayed as a second row of text. All messages are left justified on the LCD display.
<i>messageLen</i>	Length of the message: 1 to 32 bytes

Returns

RETURN_CODE: Values can be parsed with [device_getIDGStatusCodeString\(\)](#)

11.2.4.91 device_lcdDisplayLine2Message()

```
int device_lcdDisplayLine2Message (
    IN BYTE * message,
    IN int messageLen )
```

Use this function to display the message on line 2 of the LCD display. On the Vendi reader the LCD is a 2-line character display.

Parameters

<i>message</i>	Valid messages are between 1 and 16 printable characters long. All messages are left justified on the LCD display.
<i>messageLen</i>	Length of the message: 1 to 16 bytes

Returns

RETURN_CODE: Values can be parsed with [device_getIDGStatusCodeString\(\)](#)

11.2.4.92 device_listDirectory()

```
int device_listDirectory (
    IN char * directoryName,
    IN int directoryNameLen,
    IN int recursive,
    IN int onSD,
    OUT char * directory,
    IN_OUT int * directoryLen )
```

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>directoryNameLen</i>	Directory Name Length. If null, root directory is listed
<i>recursive</i>	Included sub-directories
<i>onSD</i>	0: use internal memory, 1: use SD card The returned directory information The returned directory information length

Returns

RETURN_CODE: Values can be parsed with [device_getIDGStatusCodeString\(\)](#)

11.2.4.93 device_loadCertCA()

```
int device_loadCertCA (
    IN BYTE CertType,
    IN BYTE * CACertData,
    IN int CACertDataLen )
```

Use this function to load CA Cert (Intermediate Certificate)

Parameters

<i>CertType</i>	Application CA: 0x00 TLS CA: 0x01
<i>CACertData</i>	Multi bytes CA cert data
<i>CACertDataLen</i>	Length of the CACertData

Returns

RETURN_CODE: Values can be parsed with [device_getIDGStatusCodeString\(\)](#)

11.2.4.94 device_outputLog()

```
int device_outputLog (
    IN char * filename,
    IN int filenameLen )
```

Output Log

Save the log to a file

Parameters

<i>filename</i>	the file name including the path
<i>filenameLen</i>	the length of filename

Returns

RETURN_CODE: Values can be parsed with [device_getIDGStatusCodeString\(\)](#)

11.2.4.95 device_pingDevice()

```
int device_pingDevice ( )
```

Ping Device - NEO only

Pings the reader. If connected, returns success. Otherwise, returns timeout.

Returns

RETURN_CODE: Values can be parsed with [device_getIDGStatusCodeString\(\)](#)

11.2.4.96 device_playAudio()

```
int device_playAudio (
    IN char * fileName,
    IN int fileNameLen,
    IN int onSD )
```

Play Audio This command plays an audio file loaded from the inserted SD card. The VP6800 supports 16bit PCM format .WAV files.

Parameters

<i>filename</i>	Complete path and file name of the wav file you want to play. You do not need to specify the root directory. Indicate subdirectories with a forward slash (/).
<i>filenameLen</i>	File Name Length. Maximum file name length is 30.
<i>onSD</i>	0: use internal memory (the Maximum audio file in Flash is 5M and only 2 audio file is supported), 1: use SD card

Returns

RETURN_CODE: Values can be parsed with [device_getIDGStatusCodeString\(\)](#)

11.2.4.97 device_pollCardReader()

```
int device_pollCardReader (
    OUT BYTE * status )
```

DEPRECATED : please use [device_pollCardReader_Len\(OUT BYTE * status, IN_OUT int *statusLen\)](#)

Poll Card Reader

Provides information about the state of the Card Reader

Parameters

<i>status</i>	<p>Six bytes indicating card reader information Byte 0:</p> <ul style="list-style-type: none"> • 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) • Bit 4: RFU • Bit 5: RFU • Bit 6: DATA_DUKPT Key Valid • Bit 7: Key is initialized (MFK and RSA Key pairs)
---------------	--

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)
- 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)

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

Returns

RETURN_CODE: Values can be parsed with [device_getResponseCodeString\(\)](#)

11.2.4.98 device_pollCardReader_Len()

```
int device_pollCardReader_Len (
    OUT BYTE * status,
    IN_OUT int * statusLen )
```

Poll Card Reader

Provides information about the state of the Card Reader

Parameters

<i>status</i>	<p>Six bytes indicating card reader information Byte 0:</p> <ul style="list-style-type: none"> • 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) • Bit 4: RFU • Bit 5: RFU • Bit 6: DATA_DUKPT Key Valid • Bit 7: Key is initialized (MFK and RSA Key pairs)
---------------	--

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)
- 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)

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>statusLen</i>	Length of status
------------------	------------------

Returns

RETURN_CODE: Values can be parsed with [device_getResponseCodeString\(\)](#)

11.2.4.99 device_pollForToken()

```
int device_pollForToken (
    IN int timeout,
    OUT BYTE * respData,
    IN_OUT int * respDataLen )
```

Poll for Token

Polls for a PICC

Parameters

<i>timeout</i>	timeout in milliseconds, must be multiple of 10 milliseconds. 30, 120, 630, or 1150 for example.
<i>respData</i>	Response data will be stored in respData. 1 byte of card type, and the Serial Number (or the UID) of the PICC if available.
<i>respDataLen</i>	Length of systemCode.

Returns

RETURN_CODE: Values can be parsed with [device_getIDGStatusCodeString\(\)](#)

11.2.4.100 device_queryFile()

```
int device_queryFile (
    IN char * directoryName,
    IN int directoryNameLen,
    IN char * fileName,
    IN int fileNameLen,
    OUT int * isExist,
    OUT BYTE * timeStamp,
    IN_OUT int * timeStampLen,
    OUT char * fileSize,
    IN_OUT int * fileSizeLen )
```

Query File This command checks if the specified file exists in NAND Flash..

Parameters

<i>directoryName</i>	Directory name string. No longer than 32 bytes. ASCII string, terminated by 0x00.
<i>directoryNameLen</i>	Directory Name Length.
<i>fileName</i>	File name string. No longer than 32 bytes. ASCII string, terminated by 0x00.
<i>fileNameLen</i>	File Name Length.
<i>isExist</i>	File exists: 1, File not exists 0.
<i>timeStamp</i>	Latest time stamp of the file. 6 bytes BCD code if the file exists.
<i>timeStampLen</i>	Length of timeStamp. 6 if the file exists, 0 if the file does not exist.
<i>fileSize</i>	Zero-terminated ASCII string of the file size.
<i>fileSizeLen</i>	Length of fileSize.

Returns

RETURN_CODE: Values can be parsed with [device_getIDGStatusCodeString\(\)](#)

11.2.4.101 device_readFileFromSD()

```
int device_readFileFromSD (
    IN char * directoryName,
    IN int directoryNameLen,
    IN char * fileName,
    IN int fileNameLen,
```

```

    IN int  startingOffset,
    IN int  numBytes,
    OUT BYTE * fileData,
    IN_OUT int * fileDataLen )

```

Read Out File from SD Card This command retrieves a file from the SD card of the reader.

Parameters

<i>directoryName</i>	Directory name string. No longer than 32 bytes. ASCII string, terminated by 0x00.
<i>directoryNameLen</i>	Directory Name Length.
<i>fileName</i>	File name string. No longer than 32 bytes. ASCII string, terminated by 0x00.
<i>fileNameLen</i>	File Name Length.
<i>startingOffset</i>	Starting offset in the file to retrieve
<i>numBytes</i>	Number of bytes of file data to retrieve
<i>fileData</i>	the file data read from the SD card
<i>fileDataLen</i>	Length of fileData.

Returns

RETURN_CODE: Values can be parsed with [device_getIDGStatusCodeString\(\)](#)

11.2.4.102 device_rebootDevice()

```
int device_rebootDevice ( )
```

Reboot Device - NGA 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\(\)](#)

11.2.4.103 device_registerCameraCallBk()

```
void device_registerCameraCallBk (
    pCMR_callback pCMRf )
```

To register the camera callback function to get the image data. (Pass NULL to disable the callback.)

11.2.4.104 device_registerCardStatusFrontSwitchCallBk()

```
void device_registerCardStatusFrontSwitchCallBk (
    pCSFS_callback pCSFSf )
```

To register the card status and front switch callback function to get status. (Pass NULL to disable the callback.)

11.2.4.105 device_registerFWCallBk()

```
void device_registerFWCallBk (
    pFW_callback pFWf )
```

To register the firmware update callback function to get the firmware update status. (Pass NULL to disable the callback.)

11.2.4.106 device_registerRKICallBk()

```
void device_registerRKICallBk (
    pRKI_callback pRKIf )
```

To register the RKI callback function to get the RKI status. (Pass NULL to disable the callback.)

11.2.4.107 device_rrcConnect()

```
int device_rrcConnect ( )
```

Use this function to allow a host to establish an RRC connection to a reader

Returns

RETURN_CODE: Values can be parsed with [device_getIDGStatusCodeString\(\)](#)

11.2.4.108 device_rrcDisconnect()

```
int device_rrcDisconnect ( )
```

Use this function to allow a host to terminate its existing RRC connection to a reader

Returns

RETURN_CODE: Values can be parsed with [device_getIDGStatusCodeString\(\)](#)

11.2.4.109 device_rrcDownloadApp()

```
int device_rrcDownloadApp (
    IN char * zipFileName,
    IN int zipFileNameLen,
    IN char * appName,
    IN int appNameLen )
```

Use this function to transfer a compressed application file from a host to the reader, extracts it, and performs signature verification on its contents.

Parameters

<i>zipFileName</i>	The zip file name
<i>zipFileNameLen</i>	Length of the zipFileName
<i>AppName</i>	Application Name in ASCII
<i>AppNameLen</i>	Length of Application Name between 2 to 128

Returns

RETURN_CODE: Values can be parsed with [device_getIDGStatusCodeString\(\)](#)

11.2.4.110 device_rrcInstallApp()

```
int device_rrcInstallApp (
    IN char * appName,
    IN int appNameLen )
```

Use this function to install an application on a reader device

Parameters

<i>AppName</i>	Application Name in ASCII
<i>AppNameLen</i>	Length of Application Name between 2 to 128

Returns

RETURN_CODE: Values can be parsed with [device_getIDGStatusCodeString\(\)](#)

11.2.4.111 device_rrcRunApp()

```
int device_rrcRunApp (
    IN char * appName,
    IN int appNameLen )
```

Use this function to run an application on a reader device

Parameters

<i>AppName</i>	Application Name in ASCII
<i>AppNameLen</i>	Length of Application Name between 2 to 128

Returns

RETURN_CODE: Values can be parsed with [device_getIDGStatusCodeString\(\)](#)

11.2.4.112 device_rrcUninstallApp()

```
int device_rrcUninstallApp (
    IN char * appName,
    IN int appNameLen )
```

Use this function to uninstall an application on a reader device

Parameters

<i>AppName</i>	Application Name in ASCII
<i>AppNameLen</i>	Length of Application Name between 2 to 128

Returns

RETURN_CODE: Values can be parsed with [device_getIDGStatusCodeString\(\)](#)

11.2.4.113 device_selfCheck()

```
int device_selfCheck ( )
```

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.

Returns

RETURN_CODE: Values can be parsed with [device_getResponseCodeString](#)

11.2.4.114 device_SendDataCommand()

```
int device_SendDataCommand (
    IN BYTE * cmd,
    IN int cmdLen,
    IN BYTE * data,
    IN int dataLen,
    OUT BYTE * response,
    IN_OUT int * respLen )
```

Send a Command to NGA device

Sends a command to the device .

Parameters

<i>cmd</i>	buffer of command to execute.
<i>cmdLen,the</i>	length of the buffer cmd.
<i>data</i>	buffer of IDG command data.
<i>dataLen,the</i>	length of the buffer data.
<i>response</i>	Response data
<i>respLen,the</i>	length of Response data

Returns

RETURN_CODE: Values can be parsed with [device_getResponseCodeString\(\)](#)

11.2.4.115 device_SendDataCommandITP()

```
int device_SendDataCommandITP (
    IN BYTE * cmd,
    IN int cmdLen,
    IN BYTE * data,
    IN int dataLen,
    OUT BYTE * response,
    IN_OUT int * respLen )
```

Send a Command to ITP device

Sends a command to the device .

Parameters

<i>cmd</i>	buffer of command to execute.
<i>cmdLen,the</i>	length of the buffer cmd.
<i>data</i>	buffer of ITP command data.
<i>dataLen,the</i>	length of the buffer data.
<i>response</i>	Response data
<i>respLen,the</i>	length of Response data

Returns

RETURN_CODE: Values can be parsed with [device_getResponseCodeString\(\)](#)

11.2.4.116 device_SendDataCommandNEO()

```
int device_SendDataCommandNEO (
    IN int cmd,
    IN int subCmd,
    IN BYTE * data,
    IN int dataLen,
    OUT BYTE * response,
    IN_OUT int * respLen )
```

Send a Command to NEO device

Sends a command to the NEO device .

Parameters

<i>cmd</i>	command to execute.
<i>subCmd,sub</i>	command to execute.
<i>data</i>	buffer of NEO command data.
<i>dataLen,the</i>	length of the buffer data.
<i>response</i>	Response data
<i>respLen,the</i>	length of Response data

Returns

RETURN_CODE: Values can be parsed with [device_getIDGStatusCodeString\(\)](#)

11.2.4.117 device_setAudioVolume()

```
int device_setAudioVolume (
    IN BYTE volume )
```

Set Audio Volume This command sets the reader's audio volume.

Parameters

<i>Value</i>	0-20, where 0 is silent and 20 is full volume
--------------	---

Returns

RETURN_CODE: Values can be parsed with [device_getIDGStatusCodeString\(\)](#)

11.2.4.118 device_setBurstMode()

```
int device_setBurstMode (  
    IN BYTE mode )
```

Send Burst Mode - NEO

Sets the burst mode for the device.

Parameters

<i>mode</i>	0 = OFF, 1 = Always On, 2 = Auto Exit
-------------	---------------------------------------

Returns

success or error code. Values can be parsed with [device_getIDGStatusCodeString](#)

See also

[ErrorCode](#)

11.2.4.119 device_setCameraParameters()

```
int device_setCameraParameters (  
    IN BYTE isAutoFocus,  
    IN BYTE focalLength )
```

Set Camera Parameters This command is used to set the camera parameters (e.g., auto/fixed focal length as focus).

Parameters

<i>isAutoFocus</i>	0: fixed focus, 1: auto focus
<i>focalLength</i>	focal length Value 0x00-0xFF, where 0x00 is the farthest, 0xFF is nearest. Not used for auto focus.

Returns

RETURN_CODE: Values can be parsed with [device_getIDGStatusCodeString\(\)](#)

11.2.4.120 device_setCancelTransactionMode()

```
int device_setCancelTransactionMode (
```

```
int mode )
```

Set Cancel Transaction Mode

Set the cancel transaction mode to be with or without LCD message

Parameters

<i>mode</i>	0: With LCD message 1: Without LCD message
-------------	--

Returns

success or error code. 1: Success, 0: Failed

11.2.4.121 device_setConfigPath()

```
int device_setConfigPath (
    const char * path )
```

Set the path to the config xml file(s) if any

Parameters

<i>path</i>	The path to the config xml files (such as "NEO2_Devices.xml" which contains the information of NEO2 devices). Only need to specify the path to the folder which contains the config files. File names are not needed. The maximum length of path is 200 characters including the '\0' at the end.
-------------	---

Returns

RETURN_CODE: Values can be parsed with [device_getIDGStatusCodeString\(\)](#)

11.2.4.122 device_setCoreDumpLogFile()

```
int device_setCoreDumpLogFile (
    IN char * filename,
    IN int filenameLen )
```

Set the Core Dump Log File Name

Set the file name of the log file when the core dump occurs for the firmware

Parameters

<i>filename</i>	the file name including the path
<i>filenameLen</i>	the length of filename; the maximum length is 99 bytes not including the null character

Returns

RETURN_CODE: Values can be parsed with [device_getIDGStatusCodeString\(\)](#)

11.2.4.123 device_setCurrentDevice()

```
int device_setCurrentDevice (
    int deviceType )
```

Sets the current device to talk to

The connect status can be checked by [device_isConnected\(\)](#).

Note: the file "NEO2_Devices.xml" is required for NEO 2 readers

Parameters

<i>deviceType</i>	Device to connect to
	<pre>enum DEVICE_TYPE { IDT_DEVICE_UNKNOWN=0, IDT_DEVICE_AUGUSTA_HID, IDT_DEVICE_AUGUSTA_KB, IDT_DEVICE_AUGUSTA_S_HID, IDT_DEVICE_AUGUSTA_S_KB, IDT_DEVICE_AUGUSTA_S_TTK_HID, IDT_DEVICE_SPECTRUM_PRO, IDT_DEVICE_MINISMART_II, IDT_DEVICE_L80, IDT_DEVICE_L100, IDT_DEVICE_UNIPAY, IDT_DEVICE_UNIPAY_I_V, IDT_DEVICE_VP3300_AJ, IDT_DEVICE_KIOSK_III, IDT_DEVICE_KIOSK_III_S, IDT_DEVICE_PIP_READER, IDT_DEVICE_VENDI, IDT_DEVICE_VP3300_USB, IDT_DEVICE_UNIPAY_I_V_TTK, IDT_DEVICE_VP3300_BT, IDT_DEVICE_VP8800, IDT_DEVICE_SREDKEY2_HID, IDT_DEVICE_SREDKEY2_KB, IDT_DEVICE_NEO2, IDT_DEVICE_MINISMART_II_COM = IDT_DEVICE_NEO2+5, IDT_DEVICE_SPECTRUM_PRO_COM, IDT_DEVICE_KIOSK_III_COM, IDT_DEVICE_KIOSK_III_S_COM, IDT_DEVICE_PIP_READER_COM, IDT_DEVICE_VP3300_COM, IDT_DEVICE_NEO2_COM, IDT_DEVICE_MAX_DEVICES = IDT_DEVICE_NEO2_COM+5 };</pre>

Returns

RETURN_CODE: 1: success, 0: failed

11.2.4.124 device_setMerchantRecord()

```
int device_setMerchantRecord (
    int index,
    int enabled,
    char * merchantID,
    char * merchantURL )
```

Set Merchant Record - NEO 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

Returns

RETURN_CODE: Values can be parsed with [device_getIDGStatusCodeString\(\)](#)

11.2.4.125 device_setNEO2DevicesConfigs()

```
int device_setNEO2DevicesConfigs (
    IN const char * configs,
    IN int len )
```

Pass the content of the config xml file ("NEO2_Devices.xml") as a string to the SDK instead of reading the config xml file by the SDK. It needs to be called before `device_init()`, otherwise the SDK will try to read the config xml file.

Parameters

<i>configs</i>	The content read from the config xml file ("NEO2_Devices.xml" which contains the information of NEO2 devices).
<i>len</i>	The length of the string configs. The maximum length is 5000 bytes.

11.2.4.126 `device_setNEOAltDevice()`

```
void device_setNEOAltDevice (
    int alt )
```

Set the alternative device type for the NEO 2 or 3 readers

Parameters

<i>alt</i>	The alternative device type of the NEO 2/3 readers.
------------	---

11.2.4.127 `device_setNEOGen()`

```
void device_setNEOGen (
    int gen )
```

Set the generation for the NEO 2 or 3 readers

Parameters

<i>gen</i>	The generation of the NEO 2/3 readers. 2: NEO 2, 3: NEO 3
------------	---

11.2.4.128 `device_setPollMode()`

```
int device_setPollMode (
    IN BYTE mode )
```

Set Poll Mode - NEO

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
-------------	-----------------------------------

Returns

RETURN_CODE: Values can be parsed with [device_getIDGStatusCodeString\(\)](#)

11.2.4.129 device_setRKI_URL()

```
void device_setRKI_URL (
    IN char * rkiURL,
    IN int rkiURLLen )
```

Set the URL for RKI

Parameters

<i>rkiURL</i>	The URL for RKI (less than 100 characters). Pass NULL to reset to default URL
<i>rkiURLLen</i>	The length of rkiURL. Pass 0 to reset to default URL

Returns**11.2.4.130 device_setRTCDateTime()**

```
int device_setRTCDateTime (
    IN BYTE * dateTime,
    IN int dateTimeLen )
```

set RTC date and time of the device

Parameters

<i>dateTime</i>	<dateTime data>=""> is: 6 byte data for YYMMDDHHMMSS in hex. For example 0x171003102547 stands for 2017 Oct 3rd 10:25:47
<i>dateTimeLen</i>	should be always 6 bytes

Returns

success or error code. Values can be parsed with [device_getResponseCodeString](#)

See also

[ErrorCode](#)

11.2.4.131 device_setSDKWaitTime()

```
void device_setSDKWaitTime (
    int waitTime )
```

Set SDK Wait Time

Set the SDK wait time for transactions

Parameters

<i>waitTime</i>	The SDK wait time for transaction in seconds. The Maximum is 2147483 seconds.
-----------------	---

11.2.4.132 device_setSleepModeTime()

```
int device_setSleepModeTime (
    int time )
```

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.
-------------	------------------------------------

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString

11.2.4.133 device_setSystemLanguage()

```
int device_setSystemLanguage (
    char * language )
```

Set Model Number for the device**Parameters**

<i>sNumber</i>	Model Number
----------------	--------------

Returns

RETURN_CODE: Values can be parsed with device_getIDGStatusCodeString Set System Language Sets the language for the message displayed in the LCD screen

Parameters

<i>language</i>	2-byte ASCII code, can be "EN" or "JP"
-----------------	--

Returns

success or error code. Values can be parsed with device_getIDGStatusCodeString

See also

ErrorCode

11.2.4.134 device_setThreadStackSize()

```
void device_setThreadStackSize (
    int threadSize )
```

Set Thread Stack Size

Set the stack size setting for newly created threads

11.2.4.135 device_setTransactionExponent()

```
void device_setTransactionExponent (
    int exponent )
```

Sets the transaction exponent to be used with device_startTransaction. Default value is 2

Parameters

<i>exponent, The</i>	exponent to use when calling device_startTransaction
----------------------	--

11.2.4.136 device_startListenNotifications()

```
int device_startListenNotifications ( )
```

Start Listen Notifications This function enables Card Status and Front Switch notifications.

Returns

RETURN_CODE: Values can be parsed with [device_getIDGStatusCodeString\(\)](#)

11.2.4.137 device_startQRCodeScan()

```
int device_startQRCodeScan (
    IN int _timeout )
```

Start QR Code Scanning

Enables QR Code scanning with the default window, waiting for the QR code.

Parameters

<i>timeout</i>	QR Code Scan Timeout Value. Between 30 and 65536 seconds.
----------------	---

Returns

RETURN_CODE: Values can be parsed with [device_getIDGStatusCodeString\(\)](#) Note: if auto poll mode is on, it will return command not allowed error

11.2.4.138 device_startQRCodeScanWithDisplayWindowInfo()

```
int device_startQRCodeScanWithDisplayWindowInfo (
    IN int _timeout,
    IN int x,
    IN int y,
    IN int width,
    IN int height )
```

Start QR Code Scanning with Display Window Info

Enables QR Code scanning, waiting for the QR code.

Parameters

<i>timeout</i>	QR Code Scan Timeout Value. Between 30 and 65536 seconds.
<i>x</i>	the x-coordinate of the display window.
<i>y</i>	the y-coordinate of the display window.
<i>width</i>	the width of the display window.
<i>height</i>	the height of the display window.

Returns

RETURN_CODE: Values can be parsed with [device_getIDGStatusCodeString\(\)](#) Note: if auto poll mode is on, it will return command not allowed error

11.2.4.139 device_startRKI()

```
int device_startRKI (
    IN const char * caPath,
    IN int isProduction )
```

Start remote key injection.

Parameters

<i>caPath</i>	The path to ca-certificates.crt. It should be NULL, because the file is not used anymore.
<i>isProduction</i>	1: The reader is a production unit, 0: The reader is not a production unit

Returns

success or error code.

See also

[ErrorCode](#)

11.2.4.140 device_startTakingPhoto()

```
int device_startTakingPhoto (
    IN int _timeout )
```

Start Taking Photo

Enables the camera to take a photo.

Parameters

<i>timeout</i>	Photo taking Timeout Value. Between 30 and 65536 seconds.
----------------	---

Returns

RETURN_CODE: Values can be parsed with [device_getIDGStatusCodeString\(\)](#) Note: if auto poll mode is on, it will return command not allowed error

11.2.4.141 device_startTransaction()

```
int device_startTransaction (
    IN double amount,
    IN double amtOther,
    IN int type,
    IN const int _timeout,
    IN BYTE * tags,
    IN int tagsLen )
```

Start Transaction Request

Authorizes the transaction for an MSR/CTLS/ICC card

The tags will be returned in the callback routine.

Parameters

<i>amount</i>	Transaction amount value (tag value 9F02) • SEE IMPORTANT NOTE BELOW
<i>amtOther</i>	Other amount value, if any (tag value 9F03) • SEE IMPORTANT NOTE BELOW
<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. 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>tagsLen</i>	The length of tags data buffer.

>>>>>IMPORTANT: parameters for amount and amtOther MUST BE PASSED AS A DOUBLE VALUE WITH DECIMAL POINT. Example, do not pass 1, but instead pass 1.0 or 1.00. Otherwise, results will be unpredictable

Returns

RETURN_CODE: Values can be parsed with [device_getIDGStatusCodeString\(\)](#) Note: if auto poll is on, it will return the error IDG_P2_STATUS_CODE_COMMAND_NOT_ALLOWED

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 device_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

11.2.4.142 device_stopAudio()

```
int device_stopAudio ( )
```

Stop Audio This command stops the audio the reader is playing.

Returns

RETURN_CODE: Values can be parsed with [device_getIDGStatusCodeString\(\)](#)

11.2.4.143 device_stopListenNotifications()

```
int device_stopListenNotifications ( )
```

Stop Listen Notifications This function disables Card Status and Front Switch notifications.

Returns

RETURN_CODE: Values can be parsed with [device_getIDGStatusCodeString\(\)](#)

11.2.4.144 device_stopQRCodeScan()

```
int device_stopQRCodeScan ( )
```

Stop QR Code Scanning Cancels QR Code scanning request.

Returns

RETURN_CODE: Values can be parsed with [device_getIDGStatusCodeString\(\)](#)

11.2.4.145 device_stopTakingPhoto()

```
int device_stopTakingPhoto ( )
```

Stop Taking Photo Cancels Photo Taking request.

Returns

RETURN_CODE: Values can be parsed with [device_getIDGStatusCodeString\(\)](#)

11.2.4.146 device_toSDCard()

```
void device_toSDCard (
    int forSDCard )
```

To SD Card

Set the destination of the file or directory function

Parameters

<i>forSDCard</i>	0: for internal memory, 1: for SD card
------------------	--

11.2.4.147 `device_transferFile()`

```
int device_transferFile (
    IN char * fileName,
    IN int fileNameLen,
    IN BYTE * file,
    IN int fileLen )
```

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>filenameLen</i>	File Name Length.
<i>file</i>	The data file.
<i>fileLen</i>	File Length.

Returns

RETURN_CODE: Values can be parsed with [device_getIDGStatusCodeString\(\)](#)

11.2.4.148 `device_turnOffYellowLED()`

```
int device_turnOffYellowLED ( )
```

Use this function to turn off the ViVOpay Vendi reader yellow LED. This LED is located below the three blue LEDs

Returns

RETURN_CODE: Values can be parsed with [device_getIDGStatusCodeString\(\)](#)

11.2.4.149 `device_turnOnYellowLED()`

```
int device_turnOnYellowLED ( )
```

Use this function to turn on the ViVOpay Vendi reader yellow LED. This LED is located below the three blue LEDs

Returns

RETURN_CODE: Values can be parsed with [device_getIDGStatusCodeString\(\)](#)

11.2.4.150 device_updateFirmware()

```
int device_updateFirmware (
    IN BYTE * firmwareData,
    IN int  firmwareDataLen,
    IN char * firmwareName,
    IN int  encryptionType,
    IN BYTE * keyBlob,
    IN int  keyBlobLen )
```

Update Firmware - NGA 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>firmwareDataLen</i>	Length of firmwareData
<i>firmwareName</i>	Firmware name. Must be one of the following two strings (with appropriate version information) <ul style="list-style-type: none"> • "SP K21 APP Vx.xx.xxx" • "SP MAX APP Vx.xx.xxx"
<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)
<i>keyBlobLen</i>	Length of keyBlob

Returns

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

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

11.2.4.151 device_verifyBackdoorKey()

```
int device_verifyBackdoorKey ( )
```

Verify Backdoor Key to Unlock Security

Returns

RETURN_CODE: Values can be parsed with `device_getResponseCodeString`

Note: The function is only for TTK devices.

11.2.4.152 emv_activateTransaction()

```
int emv_activateTransaction (
    IN int timeout,
    IN BYTE * tags,
    IN int tagsLen,
    IN int forceOnline )
```

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>	Tags to be included in the request. Passed as a TLV stream. Example, tag 9F02 with amount 0x000000000100 would be 0x9F0206000000000100
<i>tagsLen</i>	Length of tags
<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 = DFEE1A079F029F36959F37

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString >>>>IMPORTANT: parameters for amount and amtOther MUST BE PASSED AS A DOUBLE VALUE WITH DECIMAL POINT. Example, do not pass 1, but instead pass 1.0 or 1.00. Otherwise, results will be unpredictable.

11.2.4.153 emv_allowFallback()

```
void emv_allowFallback (
    IN int allow )
```

Allow fallback for EMV transactions. Default is TRUE

Parameters

<i>allow</i>	TRUE = allow fallback, FALSE = don't allow fallback
--------------	---

11.2.4.154 emv_authenticateTransaction()

```
int emv_authenticateTransaction (
    IN BYTE * updatedTLV,
    IN int updatedTLVLen )
```

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 = DFEE1A079F029F36959F37
<i>updatedTLVLen</i>	

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString

11.2.4.155 emv_authenticateTransactionWithTimeout()

```
int emv_authenticateTransactionWithTimeout (
    IN int timeout,
    IN BYTE * updatedTLV,
    IN int updatedTLVLen )
```

Authenticate EMV Transaction Request with Timeout

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>timeout</i>	Timeout value in seconds.
<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 = DFEE1A079F029F36959F37
<i>updatedTLVLen</i>	

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString

11.2.4.156 emv_callbackResponseLCD()

```
int emv_callbackResponseLCD (
    IN int type,
    byte selection )
```

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

Returns

RETURN_CODE: Values can be parsed with [device_getResponseCodeString\(\)](#)

11.2.4.157 emv_callbackResponseMSR()

```
int emv_callbackResponseMSR (
    IN BYTE * MSR,
    IN_OUT int MSRLen )
```

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>MSRLen</i>	the length of Swiped track data

Returns

RETURN_CODE: Values can be parsed with [device_getResponseCodeString\(\)](#)

11.2.4.158 emv_cancelTransaction()

```
int emv_cancelTransaction ( )
```

Cancel EMV Transaction

Cancels the currently executing EMV transaction.

Returns

RETURN_CODE: Values can be parsed with [device_getResponseCodeString\(\)](#)

11.2.4.159 emv_completeTransaction()

```
int emv_completeTransaction (
    IN int commError,
    IN BYTE * authCode,
    IN int authCodeLen,
    IN BYTE * iad,
    IN int iadLen,
    IN BYTE * tlvScripts,
    IN int tlvScriptsLen,
    IN BYTE * tlv,
    IN int tlvLen )
```

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(1) if host was unreachable, or FALSE(0) 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>authCodeLen</i>	the length of authCode
<i>iad</i>	Issuer Authentication Data, if any. Example 0x11223344556677883030 (tag value 91).
<i>iadLen</i>	the length of iadLen
<i>tlvScripts</i>	71/72 scripts, if any
<i>tlvScriptsLen</i>	the length of tlvScriptsLen
<i>tlv</i>	Additional TLV data to return with transaction results (if any)
<i>tlvLen</i>	the length of tlv

Returns

RETURN_CODE: Values can be parsed with [device_getResponseCodeString](#)

11.2.4.160 emv_getAutoAuthenticateTransaction()

```
int emv_getAutoAuthenticateTransaction ( )
```

Gets auto authenticate value for EMV transactions.

Returns

RETURN_CODE: TRUE = auto authenticate, FALSE = manually authenticate

11.2.4.161 emv_getAutoCompleteTransaction()

```
int emv_getAutoCompleteTransaction ( )
```

Gets auto complete value for EMV transactions.

Returns

RETURN_CODE: TRUE = auto complete, FALSE = manually complete

11.2.4.162 emv_getEMVConfigurationCheckValue()

```
int emv_getEMVConfigurationCheckValue (
    OUT BYTE * checkValue,
    IN_OUT int * checkValueLen )
```

Get EMV Kernel configuration check value info

Parameters

<i>checkValue</i>	Response returned of Kernel configuration check value info
<i>checkValueLen</i>	the length of checkValue

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString

11.2.4.163 emv_getEMVKernelCheckValue()

```
int emv_getEMVKernelCheckValue (
    OUT BYTE * checkValue,
    IN_OUT int * checkValueLen )
```

Get EMV Kernel check value info

Parameters

<i>checkValue</i>	Response returned of Kernel check value info
<i>checkValueLen</i>	the length of checkValue

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString

11.2.4.164 emv_getEMVKernelVersion()

```
int emv_getEMVKernelVersion (
    OUT char * version )
```

DEPRECATED : please use [emv_getEMVKernelVersion_Len\(OUT char* version, IN_OUT int *versionLen\)](#)

Polls device for EMV Kernel Version

Parameters

<i>version</i>	Response returned of Kernel Version; needs to have at least 128 bytes of memory.
----------------	--

Returns

RETURN_CODE: Values can be parsed with [device_getResponseCodeString\(\)](#)

11.2.4.165 emv_getEMVKernelVersion_Len()

```
int emv_getEMVKernelVersion_Len (
    OUT char * version,
    IN_OUT int * versionLen )
```

Polls device for EMV Kernel Version

Parameters

<i>version</i>	Response returned of Kernel Version
<i>versionLen</i>	Length of version

Returns

RETURN_CODE: Values can be parsed with [device_getResponseCodeString\(\)](#)

11.2.4.166 emv_registerCallBk()

```
void emv_registerCallBk (
    pEMV_callback pEMVf )
```

To register the emv callback function to get the EMV processing response. (Pass NULL to disable the callback.)

11.2.4.167 emv_removeAllApplicationData()

```
int emv_removeAllApplicationData ( )
```

Remove All Application Data

Removes all the Application Data

Returns

RETURN_CODE: Values can be parsed with [device_getResponseCodeString\(\)](#)

11.2.4.168 emv_removeAllCAPK()

```
int emv_removeAllCAPK ( )
```

Remove All Certificate Authority Public Key

Removes all the CAPK

Returns

RETURN_CODE: Values can be parsed with [device_getResponseCodeString\(\)](#)

11.2.4.169 emv_removeAllCRL()

```
int emv_removeAllCRL ( )
```

Remove All Certificate Revocation List Entries

Removes all CRLentry entries

Returns

RETURN_CODE: Values can be parsed with [device_getResponseCodeString\(\)](#)

11.2.4.170 emv_removeApplicationData()

```
int emv_removeApplicationData (
    IN BYTE * AID,
    IN int AIDLen )
```

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>AIDLen</i>	the length of AID data buffer

Returns

RETURN_CODE: Values can be parsed with [device_getResponseCodeString\(\)](#)

11.2.4.171 emv_removeCAPK()

```
int emv_removeCAPK (
    IN BYTE * capk,
    IN int capkLen )
```

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>capkLen</i>	the length of capk data buffer

Returns

RETURN_CODE: Values can be parsed with [device_getResponseCodeString\(\)](#)

11.2.4.172 emv_removeCRL()

```
int emv_removeCRL (
    IN BYTE * list,
    IN int lsLen )
```

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>lsLen</i>	the length of list data buffer

Returns

RETURN_CODE: Values can be parsed with [device_getResponseCodeString\(\)](#)

11.2.4.173 emv_removeTerminalData()

```
int emv_removeTerminalData ( )
```

Remove Terminal Data

Removes the Terminal Data

Returns

RETURN_CODE: Values can be parsed with [device_getResponseCodeString\(\)](#)

11.2.4.174 emv_retrieveAIDList()

```
int emv_retrieveAIDList (
    OUT BYTE * AIDList,
    IN_OUT int * AIDListLen )
```

Retrieve AID list

Returns all the AID names installed on the terminal.

Parameters

<i>AIDList</i>	array of AID name byte arrays
<i>AIDListLen</i>	the length of AIDList array buffer

Returns

RETURN_CODE: Values can be parsed with [device_getResponseCodeString\(\)](#)

11.2.4.175 emv_retrieveApplicationData()

```
int emv_retrieveApplicationData (
    IN BYTE * AID,
    IN int AIDLen,
    OUT BYTE * tlv,
    IN_OUT int * tlvLen )
```

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>AIDLen</i>	the length of AID data buffer.
<i>tlv</i>	The TLV elements of the requested AID
<i>tlvLen</i>	the length of tlv data buffer.

Returns

RETURN_CODE: Values can be parsed with [device_getResponseCodeString\(\)](#)

11.2.4.176 emv_retrieveCAPK()

```
int emv_retrieveCAPK (
    IN BYTE * capk,
    IN int capkLen,
    OUT BYTE * key,
    IN_OUT int * keyLen )
```

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>capkLen</i>	the length of capk data buffer

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>keyLen</i>	the length of key data buffer

Returns

RETURN_CODE: Values can be parsed with [device_getResponseCodeString\(\)](#)

11.2.4.177 emv_retrieveCAPKList()

```
int emv_retrieveCAPKList (
    OUT BYTE * keys,
    IN_OUT int * keysLen )
```

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>keysLen</i>	the length of keys data buffer

Returns

RETURN_CODE: Values can be parsed with [device_getResponseCodeString\(\)](#)

11.2.4.178 emv_retrieveCRL()

```
int emv_retrieveCRL (
    OUT BYTE * list,
    IN_OUT int * listLen )
```

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>lssLen</i>	the length of list data buffer

Returns

RETURN_CODE: Values can be parsed with [device_getResponseCodeString\(\)](#)

11.2.4.179 emv_retrieveTerminalData()

```
int emv_retrieveTerminalData (
    OUT BYTE * tlv,
    IN_OUT int * tlvLen )
```

Retrieve Terminal Data

Retrieves the Terminal Data.

Parameters

<i>tlv</i>	Response returned as a TLV
<i>tlvLen</i>	the length of tlv data buffer

Returns

RETURN_CODE: Values can be parsed with [device_getResponseCodeString\(\)](#)

11.2.4.180 emv_retrieveTerminalID()

```
int emv_retrieveTerminalID (
    OUT char * terminalID )
```

DEPRECATED : please use [emv_retrieveTerminalID_Len\(OUT char* terminalID, IN_OUT int *terminalIDLen\)](#)

Gets the terminal ID as printable characters .

Parameters

<i>terminalID</i>	Terminal ID string; needs to have at least 30 bytes of memory
-------------------	---

Returns

RETURN_CODE: Values can be parsed with [device_getResponseCodeString\(\)](#)

11.2.4.181 emv_retrieveTerminalID_Len()

```
int emv_retrieveTerminalID_Len (
```

```
OUT char * terminalID,  
IN_OUT int * terminalIDLen )
```

Gets the terminal ID as printable characters .

Parameters

<i>terminalID</i>	Terminal ID string
<i>terminalIDLen</i>	Length of terminalID

Returns

RETURN_CODE: Values can be parsed with [device_getResponseCodeString\(\)](#)

11.2.4.182 emv_retrieveTransactionResult()

```
int emv_retrieveTransactionResult (  
    IN BYTE * tags,  
    IN int tagsLen,  
    IDTTransactionData * cardData )
```

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>tagsLen</i>	Length of tag list
<i>cardData</i>	All requested tags returned as unencrypted, encrypted and masked TLV data in IDTTransactionData object

Returns

RETURN_CODE: Values can be parsed with [device_getResponseCodeString\(\)](#)

11.2.4.183 emv_setApplicationData()

```
int emv_setApplicationData (  
    IN BYTE * name,  
    IN int nameLen,  
    IN BYTE * tlv,  
    IN int tlvLen )
```

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>nameLen</i>	the length of name data buffer of Application name,
<i>tlv</i>	Application data in TLV format

Parameters

<i>tlvLen</i>	the length of tlv data buffer
---------------	-------------------------------

Returns

RETURN_CODE: Values can be parsed with [device_getResponseCodeString\(\)](#)

11.2.4.184 emv_setApplicationDataTLV()

```
int emv_setApplicationDataTLV (
    IN BYTE * tlv,
    IN int tlvLen )
```

Set Application Data by TLV

Sets the Application Data as specified by the TLV data

Parameters

<i>tlv</i>	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) Example valid TLV, for Group #2, AID a0000000035010: "dfee2d01029f0607a0000000051010ffe10101ffe50110ffe30114ffe20106"
<i>tlvLen</i>	the length of tlv data buffer

Returns

RETURN_CODE: Values can be parsed with [device_getResponseCodeString\(\)](#)

11.2.4.185 emv_setAutoAuthenticateTransaction()

```
void emv_setAutoAuthenticateTransaction (
    IN int authenticate )
```

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

Parameters

<i>authenticate</i>	TRUE = auto authenticate, FALSE = manually authenticate
---------------------	---

11.2.4.186 emv_setAutoCompleteTransaction()

```
void emv_setAutoCompleteTransaction (
    IN int complete )
```

Enables complete for EMV transactions. If a `emv_authenticateTransaction` results in code 0x0004 (go online), then `emv_completeTransaction` can automatically execute if parameter is set to TRUE

Parameters

<i>complete</i>	TRUE = auto complete, FALSE = manually complete
-----------------	---

11.2.4.187 emv_setCAPK()

```
int emv_setCAPK (
    IN BYTE * capk,
    IN int capkLen )
```

Set Certificate Authority Public Key

Sets the CAPK as specified by the CAKey structure

Parameters

<i>capk</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>capkLen</i>	the length of capk data buffer

Returns

RETURN_CODE: Values can be parsed with [device_getResponseCodeString\(\)](#)

11.2.4.188 emv_setCRL()

```
int emv_setCRL (
    IN BYTE * list,
    IN int lsLen )
```

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>lsLen</i>	the length of list data buffer

Returns

RETURN_CODE: Values can be parsed with [device_getResponseCodeString\(\)](#)

11.2.4.189 emv_setTerminalData()

```
int emv_setTerminalData (
    IN BYTE * tlv,
    IN int tlvLen )
```

Set Terminal Data

Sets the Terminal Data as specified by the TerminalData structure passed as a parameter

Parameters

<i>tlv</i>	TerminalData configuration file
<i>tlvLen</i>	the length of tlv data buffer

Return values

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

11.2.4.190 emv_setTerminalID()

```
int emv_setTerminalID (
    IN char * terminalID )
```

Sets the terminal ID as printable characters .

Parameters

<i>terminalID</i>	Terminal ID to set
-------------------	--------------------

Returns

RETURN_CODE: Values can be parsed with [device_getResponseCodeString\(\)](#)

11.2.4.191 emv_setTerminalMajorConfiguration()

```
int emv_setTerminalMajorConfiguration (
    IN int configuration )
```

Sets the terminal major configuration in ICS .

Parameters

<i>configuration</i>	A configuration value, range 1-23 <ul style="list-style-type: none"> • 1 = 1C • 2 = 2C • 3 = 3C • 4 = 4C • 5 = 5C ... • 23 = 23C
----------------------	--

Returns

RETURN_CODE: Values can be parsed with [device_getResponseCodeString\(\)](#)

11.2.4.192 `emv_setTransactionParameters()`

```
void emv_setTransactionParameters (
    IN double amount,
    IN double amtOther,
    IN int type,
    IN int timeout,
    IN BYTE * tags,
    IN int tagsLen )
```

Set EMV Transaction Parameters

Set the parameters to be used on EMV transactions for an ICC card when Auto Poll is on

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 (Maximum Length = 500 bytes). Passed as a string. Example, tag 9F0C with amount 0x000000000100 would be "9F0C06000000000100" 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 = DFEE1A079F029F36959F37
<i>tagsLen</i>	the length of tags

11.2.4.193 `emv_startTransaction()`

```
int emv_startTransaction (
```

```

    IN double amount,
    IN double amtOther,
    IN int exponent,
    IN int type,
    IN int timeout,
    IN BYTE * tags,
    IN int tagsLen,
    IN int forceOnline )

```

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) - SEE IMPORTANT NOTE BELOW
<i>amtOther</i>	Other amount value, if any (tag value 9F03) - SEE IMPORTANT NOTE BELOW
<i>exponent</i>	Number of characters after decimal 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 stream. Example, tag 9F0C with amount 0x000000000100 would be 0x9F0C06000000000100 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>tagsLen</i>	Length of tags
<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 = DFEE1A079F029F36959F37

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString >>>>>IMPORTANT: parameters for amount and amtOther MUST BE PASSED AS A DOUBLE VALUE WITH DECIMAL POINT. Example, do not pass 1, but instead pass 1.0 or 1.00. Otherwise, results will be unpredictable

11.2.4.194 executeTransaction()

```

int executeTransaction (
    WorldPayData * data,
    pWP_callback wpCallback,
    int requestOnly )

```

executeTransaction Requests 3 sets of public keys: encrypting Keys, signing/validating keys, signing/validating 3rd party apps

Parameters

<i>data</i>	WorldPay data object
<i>wpCallback</i>	WorldPay callback

Parameters

<i>requestOnly</i>	<ul style="list-style-type: none"> • 0 = send transaction and return response, • 1 = send transaction and return both request and response, • 2 = do not send transaction, instead return request
--------------------	--

Returns

RETURN_CODE: Values can be parsed with [device_getIDGStatusCodeString\(\)](#)

11.2.4.195 executeTransaction_WorldNet()

```
int executeTransaction_WorldNet (
    WorldNetData * data,
    pWN_callback wnCallback,
    int requestOnly )
```

executeTransaction_WorldNet Requests 3 sets of public keys: encrypting Keys, signing/validating keys, signing/validating 3rd party apps

Parameters

<i>data</i>	WorldNet data object
<i>wnCallback</i>	WorldNet callback
<i>requestOnly</i>	<ul style="list-style-type: none"> • 0 = send transaction and return response, • 1 = send transaction and return both request and response, • 2 = do not send transaction, instead return request

Returns

RETURN_CODE: Values can be parsed with [device_getIDGStatusCodeString\(\)](#)

11.2.4.196 felica_authentication()

```
int felica_authentication (
    IN BYTE * key,
    IN int keyLen )
```

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

Parameters

<i>key</i>	16-byte key used for MAC generation of Read or Write with MAC
<i>keyLen</i>	length of key, must be 16 bytes

Returns

RETURN_CODE: Values can be parsed with [device_getIDGStatusCodeString\(\)](#)

11.2.4.197 felica_cancelCodeEntry()

```
int felica_cancelCodeEntry ( )
```

FeliCa Cancel Code Entry

Cancels FeliCa Get Code request

Returns

RETURN_CODE: Values can be parsed with [device_getIDGStatusCodeString\(\)](#)

11.2.4.198 felica_getCode()

```
int felica_getCode ( )
```

FeliCa Get Code

Start the FeliCa get code process Since the firmware timeout is 180 seconds, the SDK timeout of the command is set to 181 seconds.

Returns

RETURN_CODE: Values can be parsed with [device_getIDGStatusCodeString\(\)](#)

11.2.4.199 felica_poll()

```
int felica_poll (
    IN BYTE * systemCode,
    IN int systemCodeLen,
    OUT BYTE * respData,
    OUT int * respDataLen )
```

FeliCa Poll for Card

Polls for a Felica Card

Parameters

<i>systemCode</i>	System Code.
<i>systemCodeLen</i>	Length of systemCode. Must be 2 bytes
<i>respData</i>	response data will be stored in respData. Poll response as explained in FeliCA Lite-S User's Manual
<i>respDataLen</i>	Length of systemCode.

Returns

RETURN_CODE: Values can be parsed with [device_getIDGStatusCodeString\(\)](#)

11.2.4.200 felica_read()

```
int felica_read (
    IN BYTE * serviceCodeList,
    IN int serviceCodeListLen,
    IN int blockCnt,
    IN BYTE * blockList,
    IN int blockListLen,
    OUT BYTE * blockData,
    OUT int * blockDataLen )
```

FeliCa Read

Reads up to 4 blocks.

Parameters

<i>serviceCodeList</i>	Service Code List. Each service code in Service Code List = 2 bytes of data
<i>serviceCodeListLen</i>	Length of serviceCodeList
<i>blockCnt</i>	Number of blocks in blockList. Maximum 4 block requests
<i>blockList</i>	Block to read. Each block in blockList = 2 or 3 bytes of data.
<i>blockListLen</i>	Length of blockList.
<i>blockData</i>	Blocks read will be stored in blockData. Each block 16 bytes.
<i>blockDataLen</i>	Length of blockData.

Returns

RETURN_CODE: Values can be parsed with [device_getIDGStatusCodeString\(\)](#)

11.2.4.201 felica_readWithMac()

```
int felica_readWithMac (
    IN int blockCnt,
    IN BYTE * blockList,
    IN int blockListLen,
    OUT BYTE * blockData,
    OUT int * blockDataLen )
```

FeliCa Read with MAC Generation

Reads up to 3 blocks with MAC Generation. FeliCa Authentication must be performed first

Parameters

<i>blockCnt</i>	Number of blocks in blockList. Maximum 3 block requests
<i>blockList</i>	Block to read. Each block in blockList = 2 or 3 bytes of data.
<i>blockListLen</i>	Length of blockList.
<i>blockData</i>	Blocks read will be stored in blockData. Each block is 16 bytes.
<i>blockDataLen</i>	Length of blockData.

Returns

RETURN_CODE: Values can be parsed with [device_getIDGStatusCodeString\(\)](#)

11.2.4.202 felica_requestService()

```
int felica_requestService (
    IN BYTE * nodeCode,
    IN int nodeCodeLen,
    OUT BYTE * respData,
    OUT int * respDataLen )
```

FeliCa Request Service

Request Service for a Felica Card

Parameters

<i>nodeCode</i>	Node Code List. Each node 2 bytes
<i>nodeCodeLen</i>	Length of nodeCode.
<i>respData</i>	response data will be stored in respData. Response as explained in FeliCA Lite-S User's Manual
<i>respDataLen</i>	Length of respData.

Returns

RETURN_CODE: Values can be parsed with [device_getIDGStatusCodeString\(\)](#)

11.2.4.203 felica_SendCommand()

```
int felica_SendCommand (
    IN BYTE * command,
    IN int commandLen,
    OUT BYTE * respData,
    OUT int * respDataLen )
```

FeliCa Send Command

Send a Felica Command

Parameters

<i>command</i>	Command data from settlement center to be sent to felica card
<i>commandLen</i>	Length of command data
<i>respData</i>	Response data from felica card.
<i>respDataLen</i>	Length of respData.

Returns

RETURN_CODE: Values can be parsed with [device_getIDGStatusCodeString\(\)](#)

11.2.4.204 felica_write()

```
int felica_write (
    IN BYTE * serviceCodeList,
    IN int serviceCodeListLen,
    IN int blockCnt,
    IN BYTE * blockList,
    IN int blockListLen,
    IN BYTE * blockData,
    IN int blockDataLen,
    OUT BYTE * statusFlag,
    OUT int * statusFlagLen )
```

FeliCa Write

Writes a block

Parameters

<i>serviceCodeList</i>	Service Code List. Each service code in Service Code List = 2 bytes of data
<i>serviceCodeListLen</i>	Length of serviceCodeList
<i>blockCnt</i>	Number of blocks in blockList. Currently only support 1 block.
<i>blockList</i>	Block list. Each block in blockList = 2 or 3 bytes of data.
<i>blockListLen</i>	Length of blockData.
<i>blockData</i>	Block to write.
<i>blockDataLen</i>	Length of blockData. Must be 16 bytes.
<i>respData</i>	If successful, the Status Flag (2 bytes) is stored in respData.resData. Status flag response as explained in FeliCA Lite-S User's Manual, Section 4.5

Returns

RETURN_CODE: Values can be parsed with [device_getIDGStatusCodeString\(\)](#)

11.2.4.205 felica_writeWithMac()

```
int felica_writeWithMac (
    IN BYTE blockNum,
    IN BYTE * blockData,
    IN int blockDataLen )
```

FeliCa Write with MAC Generation

Writes a block with MAC Generation. FeliCa Authentication must be performed first

Parameters

<i>blockNum</i>	Number of block
<i>blockData</i>	Block to write.
<i>blockDataLen</i>	Length of blockData. Must be 16 bytes.

Returns

RETURN_CODE: Values can be parsed with [device_getIDGStatusCodeString\(\)](#)

11.2.4.206 forwardTransaction()

```
int forwardTransaction (
    IN pWP_callback wpCallback,
    IN char * forwardID,
    IN int forwardIDLen,
    IN char * password,
    IN int passwordLen,
    IN int bypassProcessing )
```

forwardTransaction Send the saved data to WorldPay and complete the transaction.

Parameters

<i>wpCallback</i>	WPCallback is the callback to send the results. Should be the same as executeTransaction callback.
<i>forwardID</i>	= ID, which could be either unique ID or Memo.
<i>forwardIDLen</i>	The length of forwardID.
<i>password</i>	= password. If null/blank, no password.
<i>passwordLen</i>	The length of passwordLen.
<i>bypassProcess</i>	= true means read the file from disk, but don't send to WorldPay, then delete the transaction as if you did send to WorldPay. The purpose of this is to allow them to delete transactions from the storage without sending to WorldPay.

Returns

RETURN_CODE: Values can be parsed with [device_getIDGStatusCodeString\(\)](#)

11.2.4.207 forwardTransaction_WorldNet()

```
int forwardTransaction_WorldNet (
    IN char * apiKey,
    IN int apiKeyLen,
    IN pWN_callback wnCallback,
    IN char * forwardID,
    IN int forwardIDLen,
    IN char * password,
    IN int passwordLen,
    IN int bypassProcessing )
```

forwardTransaction_WorldNet Send the saved data to WorldNet and complete the transaction.

Parameters

<i>wnCallback</i>	WNCallback is the callback to send the results. Should be the same as executeTransaction_WorldNet callback.
<i>forwardID</i>	= ID, which could be either unique ID or Memo.
<i>forwardIDLen</i>	The length of forwardID.
<i>password</i>	= password. If null/blank, no password.
<i>passwordLen</i>	The length of passwordLen.
<i>bypassProcess</i>	= true means read the file from disk, but don't send to WorldPay, then delete the transaction as if you did send to WorldPay. The purpose of this is to allow them to delete transactions from the storage without sending to WorldPay.

Returns

RETURN_CODE: Values can be parsed with [device_getIDGStatusCodeString\(\)](#)

11.2.4.208 icc_disable()

```
int icc_disable ( )
```

ICC Function enable/disable - AUGUSTA Disable ICC function

Returns

RETURN_CODE: Values can be parsed with [device_getResponseCodeString](#)

11.2.4.209 icc_enable()

```
int icc_enable (
    IN int withNotification )
```

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

Parameters

<i>withNotification</i>	<ul style="list-style-type: none">• 1: with notification when ICC seated status changed,• 0: without notification.
-------------------------	---

Returns

RETURN_CODE: Values can be parsed with [device_getResponseCodeString](#)

11.2.4.210 icc_exchangeAPDU()

```
int icc_exchangeAPDU (
    IN BYTE * c_APDU,
    IN int cLen,
    OUT BYTE * reData,
    IN_OUT int * reLen )
```

Exchange APDU with plain text - AUGUSTA 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>cLen</i>	APDU data packet length
<i>reData</i>	Unencrypted APDU response
<i>reLen</i>	Unencrypted APDU response data length

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString

11.2.4.211 icc_exchangeEncryptedAPDU()

```
int icc_exchangeEncryptedAPDU (
    IN BYTE * c_APDU,
    IN int cLen,
    OUT BYTE * reData,
    IN_OUT int * reLen )
```

Exchange APDU with encrypted data - AUGUSTA 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>cLen</i>	data packet length
<i>reData</i>	response encrypted APDU response. Can be three options:

[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]

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.

Parameters

<i>reLen</i>	encrypted APDU response data length
--------------	-------------------------------------

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString

11.2.4.212 `icc_getAPDU_KSN()`

```
int icc_getAPDU_KSN (
    OUT BYTE * KSN,
    IN_OUT int * inLen )
```

Get APDU KSN - AUGUSTA

Retrieves the KSN used in ICC Encrypted APDU usage

Parameters

<i>KSN</i>	Returns the encrypted APDU packet KSN
<i>inLen</i>	KSN data length

Returns

RETURN_CODE: Values can be parsed with `device_getResponseCodeString`

11.2.4.213 `icc_getFunctionStatus()`

```
int icc_getFunctionStatus (
    OUT int * enabled,
    OUT int * withNotification )
```

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

Parameters

<i>enabled</i>	<ul style="list-style-type: none">• 1: ICC Function enabled,• 0: means disabled.
<i>withNotification</i>	1 means with notification when ICC seated status changed. 0 means without notification.

Returns

RETURN_CODE: Values can be parsed with `device_getResponseCodeString`

11.2.4.214 `icc_getICCRaderStatus()`

```
int icc_getICCRaderStatus (
    OUT BYTE * status )
```

Get Reader Status - AUGUSTA

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
---------------	--

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString

11.2.4.215 icc_getKeyFormatForICCDUKPT()

```
int icc_getKeyFormatForICCDUKPT (
    OUT BYTE * format )
```

Get Key Format For DUKPT - AUGUSTA

Specifies how data will be encrypted with Data Key or PIN key (if DUKPT key loaded). This applies to both MSR and ICC

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.
---------------	--

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString

11.2.4.216 icc_getKeyTypeForICCDUKPT()

```
int icc_getKeyTypeForICCDUKPT (
    OUT BYTE * type )
```

Get Key Type for DUKPT - AUGUSTA

Specifies the key type used for ICC DUKPT encryption This applies to both MSR and ICC

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.
-------------	--

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString

11.2.4.217 icc_powerOffICC()

```
int icc_powerOffICC ( )
```

Power Off ICC

Powers down the ICC

Returns

RETURN_CODE: Values can be parsed with [device_getResponseCodeString\(\)](#)

If Success, empty If Failure, ASCII encoded data of error string

11.2.4.218 `icc_powerOnICC()`

```
int icc_powerOnICC (
    OUT BYTE * ATR,
    IN_OUT int * inLen )
```

Power On ICC

Power up the currently selected microprocessor card in the ICC reader

Parameters

<i>ATR,the</i>	ATR data response when succeeded power on ICC,
<i>inLen,the</i>	length of ATR data

Returns

RETURN_CODE: Values can be parsed with [device_getResponseCodeString\(\)](#)

11.2.4.219 `icc_setKeyFormatForICCDUKPT()`

```
int icc_setKeyFormatForICCDUKPT (
    IN BYTE format )
```

Set Key Format for DUKPT - AUGUSTA

Sets how data will be encrypted, with either TDES or AES (if DUKPT key loaded) This applies to both MSR and ICC

Parameters

<i>format</i>	encryption Encryption Type <ul style="list-style-type: none">• 00: Encrypt with TDES• 01: Encrypt with AES• 02: Encrypt with TransArmor - AUGUSTA only
---------------	--

Returns

RETURN_CODE: Values can be parsed with [device_getResponseCodeString](#)

11.2.4.220 `icc_setKeyTypeForICCDUKPT()`

```
int icc_setKeyTypeForICCDUKPT (
    IN BYTE type )
```

Set Key Type for DUKPT Key - AUGUSTA

Sets which key the data will be encrypted with, with either Data Key or PIN key (if DUKPT key loaded) This applies to both MSR and ICC

Parameters

<i>type</i>	Encryption Type
	<ul style="list-style-type: none"> • 00: Encrypt with Data Key • 01: Encrypt with PIN Key

Returns

RETURN_CODE: Values can be parsed with `device_getResponseCodeString`

11.2.4.221 `iso8583_deserializeFromXML()`

```
int iso8583_deserializeFromXML (
    IN BYTE * serializedMessage,
    IN int serializedMessageLength,
    OUT DL_ISO8583_HANDLER * ISOHandler,
    OUT DL_ISO8583_MSG * ISOMessage )
```

Deserialize the XML-formatted ISO8583 message.

Parameters

<i>serializedMessage</i>	- The XML-formatted message
<i>serializedMessageLength</i>	- The length of the XML-formatted message
<i>ISOHandler</i>	- A null ISO8583 handler
<i>ISOMessage</i>	- The ISO8583 message structure

Returns

RETURN_CODE: Values can be parsed with [device_getIDGStatusCodeString\(\)](#)

11.2.4.222 `iso8583_displayMessage()`

```
int iso8583_displayMessage (
    IN DL_ISO8583_HANDLER * ISOHandler,
    IN DL_ISO8583_MSG * ISOMessage )
```

Display the messages in a formatted manner on the screen for verifying results.

Parameters

<i>ISOHandler</i>	- The ISO8583 handler
<i>ISOMessage</i>	- The ISO8583 message structure

Returns

RETURN_CODE: Values can be parsed with [device_getIDGStatusCodeString\(\)](#)

11.2.4.223 iso8583_freeMessage()

```
int iso8583_freeMessage (
    IN DL_ISO8583_MSG * ISOMessage )
```

Deallocate the ISO8583 message structure's memory.

Parameters

<i>ISOMessage</i>	- The ISO8583 message structure
-------------------	---------------------------------

Returns

RETURN_CODE: Values can be parsed with [device_getIDGStatusCodeString\(\)](#)

11.2.4.224 iso8583_get1987Handler()

```
int iso8583_get1987Handler (
    OUT DL_ISO8583_HANDLER * ISOHandler )
```

Get the ISO8583 1987 version handler.

Parameters

<i>ISOHandler</i>	A handler with knowledge of the ISO8583 1987 version fields
-------------------	---

Returns

RETURN_CODE: Values can be parsed with [device_getIDGStatusCodeString\(\)](#)

11.2.4.225 iso8583_get1993Handler()

```
int iso8583_get1993Handler (
    OUT DL_ISO8583_HANDLER * ISOHandler )
```

Get the ISO8583 1993 version handler.

Parameters

<i>ISOHandler</i>	A handler with knowledge of the ISO8583 1993 version fields
-------------------	---

Returns

RETURN_CODE: Values can be parsed with [device_getIDGStatusCodeString\(\)](#)

11.2.4.226 iso8583_get2003Handler()

```
int iso8583_get2003Handler (
    OUT DL_ISO8583_HANDLER * ISOHandler )
```

Get the ISO8583 2003 version handler.

Parameters

<i>ISOHandler</i>	A handler with knowledge of the ISO8583 2003 version fields
-------------------	---

Returns

RETURN_CODE: Values can be parsed with [device_getIDGStatusCodeString\(\)](#)

11.2.4.227 iso8583_getField()

```
int iso8583_getField (
    IN DL_UINT16 dataField,
    IN DL_ISO8583_HANDLER * ISOHandler,
    OUT DL_ISO8583_FIELD_DEF * field )
```

Get the specified field's information using the data field.

Parameters

<i>dataField</i>	- The data field number
<i>ISOHandler</i>	- The ISO8583 handler
<i>field</i>	- The requested field

Returns

RETURN_CODE: Values can be parsed with [device_getIDGStatusCodeString\(\)](#)

11.2.4.228 iso8583_getMessageField()

```
int iso8583_getMessageField (
    IN DL_UINT16 dataField,
    IN DL_ISO8583_MSG * ISOMessage,
    OUT DL_ISO8583_MSG_FIELD * messageField )
```

Get the specified message field using the data field.

Parameters

<i>dataField</i>	- The data field number
------------------	-------------------------

Parameters

<i>ISOMessage</i>	- The ISO8583 message structure
<i>messageField</i>	- The requested message field

Returns

0 if the if the setting was applied; otherwise, return -1 on failure

11.2.4.229 iso8583_initializeMessage()

```
int iso8583_initializeMessage (
    OUT DL_ISO8583_MSG * ISOMessage )
```

Initialize the ISO8583 message structure.

Parameters

<i>ISOMessage</i>	- The initialized ISO8583 message structure
-------------------	---

Returns

RETURN_CODE: Values can be parsed with [device_getIDGStatusCodeString\(\)](#)

11.2.4.230 iso8583_packMessage()

```
int iso8583_packMessage (
    IN const DL_ISO8583_HANDLER * ISOHandler,
    IN const DL_ISO8583_MSG * ISOMessage,
    OUT DL_UINT8 * packedData,
    OUT DL_UINT16 * packedDataLength )
```

Pack the message fields into an array.

Parameters

<i>ISOHandler</i>	- The ISO8583 handler
<i>ISOMessage</i>	- The ISO8583 message structure
<i>packedData</i>	- The packaged data
<i>packedDataLength</i>	- The packaged data's length

Returns

RETURN_CODE: Values can be parsed with [device_getIDGStatusCodeString\(\)](#)

11.2.4.231 iso8583_removeMessageField()

```
int iso8583_removeMessageField (
```

```
IN DL_UINT16 dataField,  
OUT DL_ISO8583_MSG * ISOMessage )
```

Remove the specified message field.

Parameters

<i>dataField</i>	- The data field number
<i>ISOMessage</i>	- The ISO8583 message structure

Returns

RETURN_CODE: Values can be parsed with [device_getIDGStatusCodeString\(\)](#)

11.2.4.232 iso8583_serializeToXML()

```
int iso8583_serializeToXML (  
    IN DL_ISO8583_HANDLER * ISOHandler,  
    IN DL_ISO8583_MSG * ISOMessage,  
    OUT BYTE * serializedMessage,  
    OUT int * serializedMessageLength )
```

Serialize the message fields into an XML format.

Parameters

<i>ISOHandler</i>	- The ISO8583 handler
<i>ISOMessage</i>	- The ISO8583 message structure
<i>serializedMessage</i>	- The XML-formatted message
<i>serializedMessageLength</i>	- The XML message's length

Returns

RETURN_CODE: Values can be parsed with [device_getIDGStatusCodeString\(\)](#)

11.2.4.233 iso8583_setMessageField()

```
int iso8583_setMessageField (  
    IN DL_UINT16 dataField,  
    IN const DL_UINT8 * data,  
    OUT DL_ISO8583_MSG * ISOMessage )
```

Set the specified message field.

Parameters

<i>dataField</i>	- The data field number
<i>data</i>	- The data to apply
<i>ISOMessage</i>	- The ISO8583 message structure

Returns

RETURN_CODE: Values can be parsed with [device_getIDGStatusCodeString\(\)](#)

11.2.4.234 iso8583_unpackMessage()

```
int iso8583_unpackMessage (
    IN const DL_ISO8583_HANDLER * ISOHandler,
    IN const DL_UINT8 * packedData,
    IN DL_UINT16 packedDataLength,
    OUT DL_ISO8583_MSG * ISOMessage )
```

Unpack the message field array into the ISO8583 message structure.

Parameters

<i>ISOHandler</i>	- The ISO8583 handler
<i>packedData</i>	- The packaged data
<i>packedDataLength</i>	- The packaged data's length
<i>ISOMessage</i>	- The ISO8583 message structure

Returns

RETURN_CODE: Values can be parsed with [device_getIDGStatusCodeString\(\)](#)

11.2.4.235 lcd_addButton()

```
int lcd_addButton (
    IN char * screenName,
    IN int screenNameLen,
    IN char * buttonName,
    IN int buttonNameLen,
    IN BYTE type,
    IN BYTE alignment,
    IN int xCord,
    IN int yCord,
    IN char * label,
    IN int labelLen,
    OUT IDTLCDItem * returnItem )
```

Add Button

Adds a button to a selected screen. Must execute `lcd_createScreen` first to establish a screen to draw on. The SDK timeout of the command is set to 6 seconds.

Parameters

<i>screenName</i>	Screen name that will be the target of add button
<i>screenNameLen</i>	Length of screenName
<i>buttonName</i>	Button name that will be the target of add button
<i>buttonNameLen</i>	Length of buttonName

Parameters

<i>type</i>	Button Type <ul style="list-style-type: none"> • Large = 0x01 • Medium = 0x02 • Invisible = 0x03 (70px by 60 px)
<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 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 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>labelLen</i>	Length of label
<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

Returns

RETURN_CODE: Values can be parsed with [device_getIDGStatusCodeString\(\)](#)

11.2.4.236 lcd_addEthernet()

```
int lcd_addEthernet (
    IN char * screenName,
    IN int  screenNameLen,
    IN char * objectName,
    IN int  objectNameLen,
    IN BYTE alignment,
    IN int  xCord,
    IN int  yCord,
    OUT IDTLCDItem * returnItem )
```

Add Ethernet Settings

Adds an Ethernet settings to a selected screen. Must execute lcd_createScreen first to establish a screen to draw on. The SDK timeout of the command is set to 6 seconds.

Parameters

<i>screenName</i>	Screen name that will be the target of add widget
<i>screenNameLen</i>	Length of screenName
<i>objectName</i>	Object name that will be the target of add widget
<i>objectNameLen</i>	Length of objectName

Parameters

<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

Returns

RETURN_CODE: Values can be parsed with [device_getIDGStatusCodeString\(\)](#)

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

11.2.4.237 lcd_addExtVideo()

```
int lcd_addExtVideo (
    IN char * screenName,
    IN int screenNameLen,
    IN char * objectName,
    IN int objectNameLen,
    IN BYTE alignment,
    IN int xCord,
    IN int yCord,
    IN BYTE loop,
    IN BYTE numVideos,
    IN char * filenames,
    IN int filenamesLen,
    OUT IDTLCDItem * returnItem )
```

Add Extended Video

Adds a extended video to a selected screen. Must execute `lcd_createScreen` first to establish a screen to draw on. The SDK timeout of the command is set to 6 seconds.

Parameters

<i>screenName</i>	Screen name that will be the target of add video
<i>screenNameLen</i>	Length of screenName
<i>objectName</i>	Object name that will be the target of add video
<i>objectNameLen</i>	Length of objectName
<i>alignment</i>	Position for Video <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 Video, range 0-271
<i>yCord</i>	y-coordinate for Video, range 0-479
<i>loop</i>	loop the videos when it's done. 0: Do not Loop 1: Loop
<i>numVideos</i>	number of video files, range 1-4
<i>filenames</i>	Filenames of the videos separated by '\0'. Must be available in the sd card.
<i>filenamesLen</i>	Length of filenames excluding the last NULL character. Should be less than 64.
<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 video

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
video	1

11.2.4.238 lcd_addImage()

```
int lcd_addImage (
    IN char * screenName,
    IN int screenNameLen,
    IN char * objectName,
    IN int objectNameLen,
    IN BYTE alignment,
    IN int xCord,
    IN int yCord,
```

```

IN char * filename,
IN int  filenameLen,
OUT IDTLCDItem * returnItem )

```

Add Image

Adds a image to a selected screen. Must execute `lcd_createScreen` first to establish a screen to draw on. The SDK timeout of the command is set to 6 seconds.

Parameters

<i>screenName</i>	Screen name that will be the target of add image
<i>screenNameLen</i>	Length of screenName
<i>objectName</i>	Object name that will be the target of add image
<i>objectNameLen</i>	Length of objectName
<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 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 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>filenameLen</i>	Length of filename
<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

Returns

RETURN_CODE: Values can be parsed with [device_getIDGStatusCodeString\(\)](#)

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

11.2.4.239 lcd_addItemToList()

```

int lcd_addItemToList (
    IN BYTE * listGraphicsID,
    IN char * itemName,

```

```

    IN char * itemID,
    IN int selected )

```

Adds an item to an existing list.

Custom Display Mode must be enabled for custom text.

Parameters

<i>listGraphicsID</i>	Existing list's graphics ID (4 byte array) that is provided during creation
<i>itemName</i>	Item name (Maximum: 127 characters)
<i>itemID</i>	Identifier for the item (Maximum: 31 characters)
<i>selected</i>	If the item should be selected

Returns

RETURN_CODE: Values can be parsed with [device_getIDGStatusCodeString\(\)](#)

11.2.4.240 lcd_addLED()

```

int lcd_addLED (
    IN char * screenName,
    IN int screenNameLen,
    IN char * objectName,
    IN int objectNameLen,
    IN BYTE alignment,
    IN int xCord,
    IN int yCord,
    OUT IDTLCDItem * returnItem,
    IN BYTE * LED,
    IN int LEDLen )

```

Add LED

Adds a LED widget to a selected screen. Must execute `lcd_createScreen` first to establish a screen to draw on. The SDK timeout of the command is set to 6 seconds.

Parameters

<i>screenName</i>	Screen name that will be the target of add LED
<i>screenNameLen</i>	Length of screenName
<i>objectName</i>	Object name that will be the target of add LED
<i>objectNameLen</i>	Length of objectName
<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 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 LED, range 0-271
<i>yCord</i>	y-coordinate for LED, range 0-479

Parameters

<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>LEDLen</i>	Length of LED

Returns

RETURN_CODE: Values can be parsed with [device_getIDGStatusCodeString\(\)](#)

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

11.2.4.241 lcd_addText()

```
int lcd_addText (
    IN char * screenName,
    IN int screenNameLen,
    IN char * objectName,
    IN int objectNameLen,
    IN BYTE alignment,
    IN int xCord,
    IN int yCord,
    IN int width,
    IN int height,
    IN BYTE fontID,
    IN BYTE * color,
    IN int colorLen,
    IN char * label,
    IN int labelLen,
    OUT IDTLCDItem * returnItem )
```

Add text

Adds a text component to a selected screen. Must execute `lcd_createScreen` first to establish a screen to draw on. The SDK timeout of the command is set to 6 seconds.

Parameters

<i>screenName</i>	Screen name that will be the target of add text
<i>screenNameLen</i>	Length of screenName
<i>objectName</i>	Object name that will be the target of add text
<i>objectNameLen</i>	Length of objectName
<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 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 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>colorLen</i>	Length of color
<i>label</i>	Label to show on the text
<i>labelLen</i>	Length of label
<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

Returns

RETURN_CODE: Values can be parsed with [device_getIDGStatusCodeString\(\)](#)

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

Font ID	Typography Name	Font	Size
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
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

11.2.4.242 `lcd_addVideo()`

```
int lcd_addVideo (
    IN char * screenName,
    IN int screenNameLen,
    IN char * objectName,
    IN int objectNameLen,
    IN BYTE alignment,
    IN int xCord,
    IN int yCord,
    IN char * filename,
    IN int filenameLen,
    OUT IDTLCDItem * returnItem )
```

Add Video

Adds a video to a selected screen. Must execute `lcd_createScreen` first to establish a screen to draw on. The SDK timeout of the command is set to 6 seconds.

Parameters

<i>screenName</i>	Screen name that will be the target of add video
<i>screenNameLen</i>	Length of screenName
<i>objectName</i>	Object name that will be the target of add video
<i>objectNameLen</i>	Length of objectName
<i>alignment</i>	Position for Video <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 Video, range 0-271
<i>yCord</i>	y-coordinate for Video, range 0-479
<i>filename</i>	Filename of the video. Must be available in the sd card.
<i>filenameLen</i>	Length of filename
<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 video

Returns

RETURN_CODE: Values can be parsed with [device_getIDGStatusCodeString\(\)](#)

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

Item	Maximum can be created for each screen
video	1

11.2.4.243 lcd_cancelSlideShow()

```
int lcd_cancelSlideShow (
    OUT BYTE * statusCode,
    IN_OUT int * statusCodeLen )
```

Cancel slide show Cancel the slide show currently running

Parameters

<i>statusCode</i>	If the return code is not Success (0), the kernel may return a four-byte Extended Status Code
<i>statusCodeLen</i>	the length of the Extended Status Code (should be 4 bytes)

Returns

RETURN_CODE: Values can be parsed with [device_getIDGStatusCodeString\(\)](#)

11.2.4.244 lcd_captureSignature()

```
int lcd_captureSignature (
    IN int timeout )
```

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
----------------	---

Returns

RETURN_CODE: Values can be parsed with [device_getIDGStatusCodeString\(\)](#)

11.2.4.245 lcd_clearDisplay()

```
int lcd_clearDisplay (
    IN BYTE control )
```

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>control</i>	for L80 and L100 only. 0:First Line 1:Second Line 2:Third Line 3:Fourth Line 0xFF: All Screen
----------------	---

Returns

RETURN_CODE: Values can be parsed with [device_getIDGStatusCodeString\(\)](#)

11.2.4.246 lcd_clearEventQueue()

```
int lcd_clearEventQueue ( )
```

Removes all entries from the event queue.

Returns

RETURN_CODE: Values can be parsed with [device_getIDGStatusCodeString\(\)](#)

11.2.4.247 lcd_clearScreenInfo()

```
int lcd_clearScreenInfo ( )
```

Clear Screen Info

Clear all current screen information in RAM and flash. And then show 'power-on screen'

Returns

RETURN_CODE: Values can be parsed with [device_getIDGStatusCodeString\(\)](#)

11.2.4.248 lcd_cloneScreen()

```
int lcd_cloneScreen (
    IN char * screenName,
    IN int screenNameLen,
    IN char * cloneName,
    IN int cloneNameLen,
    OUT int * cloneID )
```

Clone Screen

Clones an existing screen.

Parameters

<i>screenName</i>	Screen name to clone.
<i>screenNameLen</i>	Length of screenName.
<i>cloneName</i>	Name of the cloned screen.
<i>cloneNameLen</i>	Length of cloneName.
<i>cloneID</i>	Screen ID of the cloned screen

Returns

RETURN_CODE: Values can be parsed with [device_getIDGStatusCodeString\(\)](#)

11.2.4.249 lcd_createInputField()

```
int lcd_createInputField (
    IN BYTE * specs,
    IN int specsLen,
    OUT BYTE * graphicId )
```

DEPRECATED : please use [lcd_createInputField_Len\(IN BYTE *specs, IN int specsLen, OUT BYTE *graphicId, IN_OUT int *graphicIdLen\)](#)

Create an input field on the screen.

Parameters

<i>specs</i>	The specs of the input field: Length (bytes) Description
--------------	--

-----	-----
2 - 4	X coordinate in pixels, zero terminated ASCII
-----	-----
2 - 4	Y coordinate in pixels, zero terminated ASCII
-----	-----
2 - 4	Width in pixels, zero terminated ASCII. Set to 0 (30h) for calculated width.
-----	-----
2 - 4	Height in pixels, zero terminated ASCII. Set to 0 (30h) for calculated height.
-----	-----
2	Font designation. Default font = 1, zero terminated ASCII
-----	-----
2 - 3	Zero terminated ASCII Font ID
-----	-----
3	Zero terminated ASCII hexadecimal display option flag
Bit 0	0 = No Border
1	= Show Border
Bit 1	0 = Characters are first displayed on the leftmost area of the screen.
1	= The first character entered is displayed on the rightmost area of
	the screen, and, as further digits are entered, characters scroll
	from the right to the left.
Bit 2 - 15	Reserved
-----	-----
1 or 9	Foreground color, zero terminated ASCII hexadecimal
-----	-----
1 or 9	Background color, zero terminated ASCII hexadecimal
-----	-----
1 or 9	Border color, zero terminated ASCII hexadecimal
-----	-----
1 - 65	Prefill String, zero terminated ASCII

| ----- | ----- |
 | 1 - 65 | Format String, zero terminated ASCII |

Parameters

<i>specsLen</i>	The length of specs
<i>graphicsID</i>	The graphicID of the event (required to be 4 bytes)

Returns

RETURN_CODE: Values can be parsed with [device_getIDGStatusCodeString\(\)](#)

11.2.4.250 lcd_createInputField_Len()

```
int lcd_createInputField_Len (
    IN BYTE * specs,
    IN int specsLen,
    OUT BYTE * graphicId,
    IN_OUT int * graphicIdLen )
```

Create an input field on the screen.

Parameters

<i>specs</i>	The specs of the input field: Length (bytes) Description
--------------	--

| ----- | ----- |
2 - 4	X coordinate in pixels, zero terminated ASCII
2 - 4	Y coordinate in pixels, zero terminated ASCII
-----	-----
2 - 4	Width in pixels, zero terminated ASCII. Set to 0 (30h) for calculated width.
-----	-----
2 - 4	Height in pixels, zero terminated ASCII. Set to 0 (30h) for calculated height.
-----	-----
2	Font designation. Default font = 1, zero terminated ASCII
-----	-----
2 - 3	Zero terminated ASCII Font ID
-----	-----
3	Zero terminated ASCII hexadecimal display option flag
	Bit 0 0 = No Border
	1 = Show Border
	Bit 1 0 = Characters are first displayed on the leftmost area of the screen.
	1 = The first character entered is displayed on the rightmost area of
	the screen, and, as further digits are entered, characters scroll


```

| | from the right to the left. |
| | Bit 2 - 15 Reserved |
| ----- | ----- |
| 1 or 9 | Foreground color, zero terminated ASCII hexadecimal |
| ----- | ----- |
| 1 or 9 | Background color, zero terminated ASCII hexadecimal |
| ----- | ----- |
| 1 or 9 | Border color, zero terminated ASCII hexadecimal |
| ----- | ----- |
| 1 - 65 | Prefill String, zero terminated ASCII |
| ----- | ----- |
| 1 - 65 | Format String, zero terminated ASCII |

```

Parameters

<i>specsLen</i>	The length of specs
<i>graphicsID</i>	The graphicID of the event (required to be 4 bytes)
<i>graphicsIDLen</i>	Length of graphicsID

Returns

RETURN_CODE: Values can be parsed with [device_getIDGStatusCodeString\(\)](#)

11.2.4.251 lcd_createList()

```

int lcd_createList (
    IN int posX,
    IN int posY,
    IN int numColumns,
    IN int numRows,
    IN int fontDesignation,
    IN int fontID,
    IN int verticalScrollArrowsVisible,
    IN int borderedListItems,
    IN int borderedScrollArrows,
    IN int touchSensitive,
    IN int automaticScrolling,
    OUT BYTE * graphicsID )

```

DEPRECATED : please use `lcd_createList_Len(IN int posX, IN int posY, IN int numColumns, IN int numRows, IN int fontDesignation, IN int fontID, IN int verticalScrollArrowsVisible, IN int borderedListItems, IN int borderedScrollArrows, IN int touchSensitive, IN int automaticScrolling, OUT BYTE *graphicsID, IN_OUT int *graphicsIDLen)`

Creates a display list.

Parameters

<i>posX</i>	X coordinate in pixels
<i>posY</i>	Y coordinate in pixels
<i>numColumns</i>	Number of columns to display

Parameters

<i>numOfRows</i>	Number of rows to display
<i>fontDesignation</i>	Font Designation 1 - Default font
<i>fontID</i>	Font styling Font ID Height in pixels Font Properties

| ----- | ----- | ----- |

| 1 | 13 | Regular |

| 2 | 17 | Regular |

| 3 | 17 | Bold |

| 4 | 22 | Regular |

| 5 | 20 | Regular |

| 6 | 20 | Bold |

| 7 | 29 | Regular |

| 8 | 38 | Regular |

| 9 | 38 | Bold |

| 10 | 58 | Regular |

| 11 | 58 | Bold, mono-space |

| 12 | 14 | Regular, mono-space, 8 pixels wide |

| 13 | 15 | Regular, mono-space, 9 pixels wide |

| 14 | 17 | Regular, mono-space, 9 pixels wide |

| 15 | 20 | Regular, mono-space, 11 pixels wide |

| 16 | 21 | Regular, mono-space, 12 pixels wide |

| 17 | 25 | Regular, mono-space, 14 pixels wide |

| 18 | 30 | Regular, mono-space, 17 pixels wide |

Parameters

<i>verticalScrollArrowsVisible</i>	Display vertical scroll arrows by default
<i>borederedListItems</i>	Draw border around list items
<i>borederedScrollArrows</i>	Draw border around scroll arrows (if visible)
<i>touchSensitive</i>	List items are touch enabled
<i>automaticScrolling</i>	Enable automatic scrolling of list when new items exceed display area
<i>graphicsID</i>	A four byte array containing the ID of the created element (optional) if graphicsID is NULL, the SDK will not return graphicsID if graphicsID is not NULL, the SDK will return graphicsID, but it will need 4 bytes of memory

Returns

RETURN_CODE: Values can be parsed with [device_getIDGStatusCodeString\(\)](#)

11.2.4.252 lcd_createList_Len()

```
int lcd_createList_Len (
    IN int posX,
```

```

    IN int posY,
    IN int numOfColumns,
    IN int numOfRows,
    IN int fontDesignation,
    IN int fontID,
    IN int verticalScrollArrowsVisible,
    IN int borderedListItems,
    IN int borderdScrollArrows,
    IN int touchSensitive,
    IN int automaticScrolling,
    OUT BYTE * graphicsID,
    IN_OUT int * graphicsIDLen )

```

Creates a display list.

Parameters

<i>posX</i>	X coordinate in pixels
<i>posY</i>	Y coordinate in pixels
<i>numOfColumns</i>	Number of columns to display
<i>numOfRows</i>	Number of rows to display
<i>fontDesignation</i>	Font Designation 1 - Default font
<i>fontID</i>	Font styling Font ID Height in pixels Font Properties

```

| ----- | ----- | ----- |
| 1 | 13 | Regular |
| 2 | 17 | Regular |
| 3 | 17 | Bold |
| 4 | 22 | Regular |
| 5 | 20 | Regular |
| 6 | 20 | Bold |
| 7 | 29 | Regular |
| 8 | 38 | Regular |
| 9 | 38 | Bold |
| 10 | 58 | Regular |
| 11 | 58 | Bold, mono-space |
| 12 | 14 | Regular, mono-space, 8 pixels wide |
| 13 | 15 | Regular, mono-space, 9 pixels wide |
| 14 | 17 | Regular, mono-space, 9 pixels wide |
| 15 | 20 | Regular, mono-space, 11 pixels wide |
| 16 | 21 | Regular, mono-space, 12 pixels wide |
| 17 | 25 | Regular, mono-space, 14 pixels wide |
| 18 | 30 | Regular, mono-space, 17 pixels wide |

```

Parameters

<i>verticalScrollArrowsVisible</i>	Display vertical scroll arrows by default
<i>borederedListItems</i>	Draw border around list items
<i>borederedScrollArrows</i>	Draw border around scroll arrows (if visible)

Parameters

<i>touchSensitive</i>	List items are touch enabled
<i>automaticScrolling</i>	Enable automatic scrolling of list when new items exceed display area
<i>graphicsID</i>	A four byte array containing the ID of the created element (optional)
<i>graphicsIDLen</i>	Length of graphicsID (optional)

Returns

RETURN_CODE: Values can be parsed with [device_getIDGStatusCodeString\(\)](#)

11.2.4.253 lcd_createScreen()

```
int lcd_createScreen (
    IN char * screenName,
    IN int screenNameLen,
    OUT int * ScreenID )
```

Create Screen

Creates a new screen.

Parameters

<i>screenName</i>	Screen name to use.
<i>screenNameLen</i>	Length of screenName.
<i>screenID</i>	Screen ID that was created.

Returns

RETURN_CODE: Values can be parsed with [device_getIDGStatusCodeString\(\)](#)

11.2.4.254 lcd_customDisplayMode()

```
int lcd_customDisplayMode (
    IN int enable )
```

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
---------------	----------------------------------

Returns

RETURN_CODE: Values can be parsed with [device_getIDGStatusCodeString\(\)](#)

11.2.4.255 `lcd_destroyScreen()`

```
int lcd_destroyScreen (
    IN char * screenName,
    IN int screenNameLen )
```

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 <code>lcd_createScreen</code> .
<i>screenNameLen</i>	Length of <i>screenName</i> .

Returns

RETURN_CODE: Values can be parsed with [device_getIDGStatusCodeString\(\)](#)

11.2.4.256 `lcd_displayButton()`

```
int lcd_displayButton (
    IN int posX,
    IN int posY,
    IN int buttonWidth,
    IN int buttonHeight,
    IN int fontDesignation,
    IN int fontID,
    IN int displayPosition,
    IN char * buttonLabel,
    IN int buttonTextColorR,
    IN int buttonTextColorG,
    IN int buttonTextColorB,
    IN int buttonBackgroundColorR,
    IN int buttonBackgroundColorG,
    IN int buttonBackgroundColorB,
    OUT BYTE * graphicsID )
```

DEPRECATED : please use `lcd_displayButton_Len(IN int posX, IN int posY, IN int buttonWidth, IN int buttonHeight, IN int fontDesignation, IN int fontID, IN int displayPosition, IN char *buttonLabel, IN int buttonTextColorR, IN int buttonTextColorG, IN int buttonTextColorB, IN int buttonBackgroundColorR, IN int buttonBackgroundColorG, IN int buttonBackgroundColorB, OUT BYTE *graphicsID, IN_OUT int *graphicsIDLen)`

Displays an interactive button.

Parameters

<i>posX</i>	X coordinate in pixels
<i>posY</i>	Y coordinate in pixels
<i>buttonWidth</i>	Width of the button
<i>buttonHeight</i>	Height of the button
<i>fontDesignation</i>	Font designation 1 - Default
<i>FontID</i>	Font styling Font ID Height in pixels Font Properties

|-----|-----|-----|

1	13	Regular
2	17	Regular
3	17	Bold
4	22	Regular
5	20	Regular
6	20	Bold
7	29	Regular
8	38	Regular
9	38	Bold
10	58	Regular
11	58	Bold, mono-space
12	14	Regular, mono-space, 8 pixels wide
13	15	Regular, mono-space, 9 pixels wide
14	17	Regular, mono-space, 9 pixels wide
15	20	Regular, mono-space, 11 pixels wide
16	21	Regular, mono-space, 12 pixels wide
17	25	Regular, mono-space, 14 pixels wide
18	30	Regular, mono-space, 17 pixels wide

Parameters

<i>displayPosition</i>	Button display position 0 - Center on line Y without clearing screen and without word wrap 1 - Center on line Y after clearing screen and without word wrap 2 - Display at (X, Y) without clearing screen and without word wrap 3 - Display at (X, Y) after clearing screen and without word wrap 4 - Center button on screen without clearing screen and without word wrap 5 - Center button on screen after clearing screen and without word wrap 64 - Center on line Y without clearing screen and with word wrap 65 - Center on line Y after clearing the screen and with word wrap 66 - Display at (X, Y) without clearing screen and with word wrap 67 - Display at (X, Y) after clearing screen and with word wrap 68 - Center button on screen without clearing screen and with word wrap 69 - Center button on screen after clearing screen and with word wrap
<i>buttonLabel</i>	Button label text (Maximum: 31 characters)
<i>buttonTextColorR</i>	- Red component for foreground color (0 - 255)
<i>buttonTextColorG</i>	- Green component for foreground color (0 - 255)
<i>buttonTextColorB</i>	- Blue component for foreground color (0 - 255)
<i>buttonBackgroundColorR</i>	- Red component for background color (0 - 255)
<i>buttonBackgroundColorG</i>	- Green component for background color (0 - 255)
<i>buttonBackgroundColorB</i>	- Blue component for background color (0 - 255)
<i>graphicsID</i>	A four byte array containing the ID of the created element (optional) if graphicsID is NULL, the SDK will not return graphicsID if graphicsID is not NULL, the SDK will return graphicsID, but it will need 4 bytes of memory

Returns

RETURN_CODE: Values can be parsed with [device_getIDGStatusCodeString\(\)](#)

11.2.4.257 lcd_displayButton_Len()

```
int lcd_displayButton_Len (
    IN int posX,
    IN int posY,
    IN int buttonWidth,
    IN int buttonHeight,
    IN int fontDesignation,
    IN int fontID,
    IN int displayPosition,
    IN char * buttonLabel,
    IN int buttonTextColorR,
    IN int buttonTextColorG,
    IN int buttonTextColorB,
    IN int buttonBackgroundColorR,
    IN int buttonBackgroundColorG,
    IN int buttonBackgroundColorB,
    OUT BYTE * graphicsID,
    IN_OUT int * graphicsIDLen )
```

Displays an interactive button.

Parameters

<i>posX</i>	X coordinate in pixels
<i>posY</i>	Y coordinate in pixels
<i>buttonWidth</i>	Width of the button
<i>buttonHeight</i>	Height of the button
<i>fontDesignation</i>	Font designation 1 - Default
<i>FontID</i>	Font styling Font ID Height in pixels Font Properties

	-----		-----		-----	
	1		13		Regular	
	2		17		Regular	
	3		17		Bold	
	4		22		Regular	
	5		20		Regular	
	6		20		Bold	
	7		29		Regular	
	8		38		Regular	
	9		38		Bold	
	10		58		Regular	
	11		58		Bold, mono-space	
	12		14		Regular, mono-space, 8 pixels wide	
	13		15		Regular, mono-space, 9 pixels wide	
	14		17		Regular, mono-space, 9 pixels wide	
	15		20		Regular, mono-space, 11 pixels wide	
	16		21		Regular, mono-space, 12 pixels wide	
	17		25		Regular, mono-space, 14 pixels wide	

| 18 | 30 | Regular, mono-space, 17 pixels wide |

Parameters

<i>displayPosition</i>	Button display position 0 - Center on line Y without clearing screen and without word wrap 1 - Center on line Y after clearing screen and without word wrap 2 - Display at (X, Y) without clearing screen and without word wrap 3 - Display at (X, Y) after clearing screen and without word wrap 4 - Center button on screen without clearing screen and without word wrap 5 - Center button on screen after clearing screen and without word wrap 64 - Center on line Y without clearing screen and with word wrap 65 - Center on line Y after clearing the screen and with word wrap 66 - Display at (X, Y) without clearing screen and with word wrap 67 - Display at (X, Y) after clearing screen and with word wrap 68 - Center button on screen without clearing screen and with word wrap 69 - Center button on screen after clearing screen and with word wrap
<i>buttonLabel</i>	Button label text (Maximum: 31 characters)
<i>buttonTextColorR</i>	- Red component for foreground color (0 - 255)
<i>buttonTextColorG</i>	- Green component for foreground color (0 - 255)
<i>buttonTextColorB</i>	- Blue component for foreground color (0 - 255)
<i>buttonBackgroundColorR</i>	- Red component for background color (0 - 255)
<i>buttonBackgroundColorG</i>	- Green component for background color (0 - 255)
<i>buttonBackgroundColorB</i>	- Blue component for background color (0 - 255)
<i>graphicsID</i>	A four byte array containing the ID of the created element (optional)
<i>graphicsIDLen</i>	Length of graphicsID (optional)

Returns

RETURN_CODE: Values can be parsed with [device_getIDGStatusCodeString\(\)](#)

11.2.4.258 lcd_displayMessage()

```
int lcd_displayMessage (
    int lineNumber,
    char * message,
    int messageLen )
```

Display Message on Line

Displays a message on a display line.
 @param lineNumber Line number to display message on (1-4)
 @param message Message to display
 @param messageLen length of message

Returns

RETURN_CODE: Values can be parsed with [device_getResponseCodeString](#)

11.2.4.259 lcd_displayParagraph()

```
int lcd_displayParagraph (
    IN int posX,
```



```

    IN int posY,
    IN int displayWidth,
    IN int displayHeight,
    IN int fontDesignation,
    IN int fontID,
    IN int displayProperties,
    IN char * displayText )

```

Displays text with scroll feature.

Custom Display Mode must be enabled.

Parameters

<i>posX</i>	X coordinate in pixels
<i>posY</i>	Y coordinate in pixels
<i>displayWidth</i>	Width of the display area in pixels (Minimum: 40px) 0 or NULL - Use the full width to display text
<i>displayHeight</i>	Height of the display area in pixels (Minimum: 100px) 0 or NULL - Use the full height to display text
<i>fontDesignation</i>	Font designation 1 - Default
<i>fontID</i>	Font styling Font ID Height in pixels Font Properties

```

| ----- | ----- | ----- |
| 1 | 13 | Regular |
| 2 | 17 | Regular |
| 3 | 17 | Bold |
| 4 | 22 | Regular |
| 5 | 20 | Regular |
| 6 | 20 | Bold |
| 7 | 29 | Regular |
| 8 | 38 | Regular |
| 9 | 38 | Bold |
| 10 | 58 | Regular |
| 11 | 58 | Bold, mono-space |
| 12 | 14 | Regular, mono-space, 8 pixels wide |
| 13 | 15 | Regular, mono-space, 9 pixels wide |
| 14 | 17 | Regular, mono-space, 9 pixels wide |
| 15 | 20 | Regular, mono-space, 11 pixels wide |
| 16 | 21 | Regular, mono-space, 12 pixels wide |
| 17 | 25 | Regular, mono-space, 14 pixels wide |
| 18 | 30 | Regular, mono-space, 17 pixels wide |

```

Parameters

<i>displayProperties</i>	Display properties for the text 0 - Center on line Y without clearing screen 1 - Center on line Y after clearing screen 2 - Display at (X, Y) without clearing screen 3 - Display at (X, Y) after clearing screen 4 - Center on screen without clearing screen 5 - Center on screen after clearing screen
--------------------------	---

Parameters

<i>displayText</i>	Display text (Maximum: 3999 characters)
--------------------	---

11.2.4.260 `lcd_displayPrompt()`

```
int lcd_displayPrompt (
    int promptNumber,
    int lineNumber )
```

Display Prompt on Line

Displays a message prompt from L80 or L100 memory.

Parameters

<i>promptNumber</i>	Prompt number (0-9)
<i>lineNumber</i>	Line number to display message prompt (1-4)

Returns

RETURN_CODE: Values can be parsed with `device_getResponseCodeString`

11.2.4.261 `lcd_displayText()`

```
int lcd_displayText (
    IN int posX,
    IN int posY,
    IN int displayWidth,
    IN int displayHeight,
    IN int fontDesignation,
    IN int fontID,
    IN int screenPosition,
    IN char * displayText,
    OUT BYTE * graphicsID )
```

DEPRECATED : please use `lcd_displayText_Len(IN int posX, IN int posY, IN int displayWidth, IN int displayHeight, IN int fontDesignation, IN int fontID, IN int screenPosition, IN char *displayText, OUT BYTE *graphicsID, IN_OUT int *graphicsIDLen)`

Displays text.

Custom Display Mode must be enabled for custom text. PIN pad entry is not allowed in Custom Display Mode but the Cancel, OK, and Clear keys remain active.

Parameters

<i>posX</i>	X coordinate in pixels
<i>posY</i>	Y coordinate in pixels
<i>displayWidth</i>	Width of the display area in pixels (optional)
<i>displayHeight</i>	Height of the display area in pixels (optional)
<i>fontDesignation</i>	Font designation 1 - Default
<i>FontID</i>	Font styling Font ID Height in pixels Font Properties

	-----		-----		-----	
	1		13		Regular	
	2		17		Regular	
	3		17		Bold	
	4		22		Regular	
	5		20		Regular	
	6		20		Bold	
	7		29		Regular	
	8		38		Regular	
	9		38		Bold	
	10		58		Regular	
	11		58		Bold, mono-space	
	12		14		Regular, mono-space, 8 pixels wide	
	13		15		Regular, mono-space, 9 pixels wide	
	14		17		Regular, mono-space, 9 pixels wide	
	15		20		Regular, mono-space, 11 pixels wide	
	16		21		Regular, mono-space, 12 pixels wide	
	17		25		Regular, mono-space, 14 pixels wide	
	18		30		Regular, mono-space, 17 pixels wide	

Parameters

<i>screenPosition</i>	Display position 0 - Center on line Y without clearing screen 1 - Center on line Y after clearing screen 2 - Display at (X, Y) without clearing screen 3 - Display at (X, Y) after clearing screen 4 - Display at center of screen without clearing screen 5 - Display at center of screen after clearing screen 6 - Display text right-justified without clearing screen 7 - Display text right-justified after clearing screen
<i>displayText</i>	Display text (Maximum: 1900 characters)
<i>graphicsID</i>	A four byte array containing the ID of the created element (optional) if graphicsID is NULL, the SDK will not return graphicsID if graphicsID is not NULL, the SDK will return graphicsID, but it will need 4 bytes of memory

Returns

RETURN_CODE: Values can be parsed with [device_getIDGStatusCodeString\(\)](#)

11.2.4.262 lcd_displayText_Len()

```
int lcd_displayText_Len (
    IN int posX,
    IN int posY,
    IN int displayWidth,
    IN int displayHeight,
    IN int fontDesignation,
    IN int fontID,
    IN int screenPosition,
```

```

    IN char * displayText,
    OUT BYTE * graphicsID,
    IN_OUT int * graphicsIDLen )

```

Displays text.

Custom Display Mode must be enabled for custom text. PIN pad entry is not allowed in Custom Display Mode but the Cancel, OK, and Clear keys remain active.

Parameters

<i>posX</i>	X coordinate in pixels
<i>posY</i>	Y coordinate in pixels
<i>displayWidth</i>	Width of the display area in pixels (optional)
<i>displayHeight</i>	Height of the display area in pixels (optional)
<i>fontDesignation</i>	Font designation 1 - Default
<i>FontID</i>	Font styling Font ID Height in pixels Font Properties

```

| ----- | ----- | ----- |
| 1 | 13 | Regular |
| 2 | 17 | Regular |
| 3 | 17 | Bold |
| 4 | 22 | Regular |
| 5 | 20 | Regular |
| 6 | 20 | Bold |
| 7 | 29 | Regular |
| 8 | 38 | Regular |
| 9 | 38 | Bold |
| 10 | 58 | Regular |
| 11 | 58 | Bold, mono-space |
| 12 | 14 | Regular, mono-space, 8 pixels wide |
| 13 | 15 | Regular, mono-space, 9 pixels wide |
| 14 | 17 | Regular, mono-space, 9 pixels wide |
| 15 | 20 | Regular, mono-space, 11 pixels wide |
| 16 | 21 | Regular, mono-space, 12 pixels wide |
| 17 | 25 | Regular, mono-space, 14 pixels wide |
| 18 | 30 | Regular, mono-space, 17 pixels wide |

```

Parameters

<i>screenPosition</i>	Display position 0 - Center on line Y without clearing screen 1 - Center on line Y after clearing screen 2 - Display at (X, Y) without clearing screen 3 - Display at (X, Y) after clearing screen 4 - Display at center of screen without clearing screen 5 - Display at center of screen after clearing screen 6 - Display text right-justified without clearing screen 7 - Display text right-justified after clearing screen
<i>displayText</i>	Display text (Maximum: 1900 characters)
<i>graphicsID</i>	A four byte array containing the ID of the created element (optional)
<i>graphicsIDLen</i>	Length of graphicsID (optional)

Returns

RETURN_CODE: Values can be parsed with [device_getIDGStatusCodeString\(\)](#)

11.2.4.263 lcd_enableBacklight()

```
int lcd_enableBacklight (
    int enable )
```

Enable/Disable LCD Backlight Trns on/off the LCD back lighting.

Parameters

<i>enable</i>	TRUE = turn ON backlight, FALSE = turn OFF backlight
---------------	--

Returns

RETURN_CODE: Values can be parsed with [device_getResponseCodeString](#)

11.2.4.264 lcd_getActiveScreen()

```
int lcd_getActiveScreen (
    OUT char * screenName,
    IN_OUT int * screenNameLen )
```

Get Active Screen

Returns the active screen ID.

Parameters

<i>screenName</i>	Screen name this is active.
<i>screenNameLen</i>	Length of screenName.

Returns

RETURN_CODE: Values can be parsed with [device_getIDGStatusCodeString\(\)](#)

11.2.4.265 lcd_getAllObjects()

```
int lcd_getAllObjects (
    IN char * screenName,
    IN int screenNameLen,
    IN_OUT int * objectNumbers,
    OUT IDTObjectInfo * objectInfo )
```

Get All Objects on Screen

Get all created objects' name on certain screen

Parameters

<i>screenName</i>	Screen name to get all objects
<i>screenNameLen</i>	Length of screenName
<i>objectNumbers</i>	Number of created objects
<i>returnObjects</i>	Array of all created objects each element includes -objectID of a created object -objectName of a created object -objectNameLen of objectName

Returns

RETURN_CODE: Values can be parsed with [device_getIDGStatusCodeString\(\)](#)

11.2.4.266 lcd_getAllScreens()

```
int lcd_getAllScreens (
    IN_OUT int * screenNumbers,
    OUT IDTScreenInfo * screenInfo )
```

Get All Screens

Get all created screens' name

Parameters

<i>screenNumbers</i>	Number of created screens
<i>screenInfo</i>	Array of all created screens each element includes -screenID of a created screen -screenName of a created screen -screenNameLen of screenName

Returns

RETURN_CODE: Values can be parsed with [device_getIDGStatusCodeString\(\)](#)

11.2.4.267 lcd_getBacklightStatus()

```
int lcd_getBacklightStatus (
    int * enabled )
```

Get Backlight Status

Returns the status of the LCD back lighting.

Parameters

<i>enabled</i>	1 = Backlight is ON, 0 = Backlight is OFF
----------------	---

Returns

RETURN_CODE: Values can be parsed with [device_getResponseCodeString](#)

11.2.4.268 lcd_getButtonEvent()

```
int lcd_getButtonEvent (
    OUT int * screenID,
    OUT int * objectID,
    OUT char * screenName,
    IN_OUT int * screenNameLen,
    OUT char * objectName,
    IN_OUT int * objectNameLen,
    OUT int * isLongPress )
```

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>screenNameLen</i>	Length of screenName
<i>objectName</i>	Button Name of the last clicked button
<i>objectNameLen</i>	Length of objectName
<i>isLongPress</i>	1 = Long Press, 0 = Short Press

Returns

RETURN_CODE: Values can be parsed with [device_getIDGStatusCodeString\(\)](#)

11.2.4.269 lcd_getInputEvent()

```
int lcd_getInputEvent (
    IN int timeout,
    OUT int * dataReceived,
    OUT BYTE * eventType,
    OUT BYTE * graphicsID,
    OUT BYTE * eventData )
```

DEPRECATED : please use lcd_getInputEvent_Len(IN int timeout, OUT int *dataReceived, OUT BYTE *eventType, IN_OUT int *eventTypeLen, OUT BYTE *graphicsID, IN_OUT int *graphicsIDLen, OUT BYTE *eventData, IN_OUT int *eventDataLen)

Requests input from the reader.

Parameters

<i>timeout</i>	Timeout amount in seconds 0 - No timeout
<i>dataReceived</i>	Indicates if an event occurred and data was received 0 - No data received 1 - Data received
<i>eventType</i>	The event type (required to be at least 4 bytes), see table below
<i>graphicsID</i>	The graphicID of the event (required to be at least 4 bytes)
<i>eventData</i>	The event data, see table below (required to be at least 73 bytes) Event Type Value (4 bytes) Event Specific Data

|-----|-----|-----|

| Button Event | 00030000h | Length = Variable |
 || | Byte 1: State (1 = Pressed, other values RFU) |
		Byte 2 - n: Null terminated caption

| Checkbox Event | 00030001h | Length = 1 byte |
		Byte 1: State (1 = Checked, 0 = Unchecked)

| Line Item Event | 00030002h | Length = 5 bytes |
 || | Byte 1: State (1 = Item Selected, other values RFU) |
		Byte 2 - n: Caption of the selected item

Keypad Event	00030003h	Length = 3 bytes
		Byte 1: State (1 = key pressed, 2 = key released, other values RFU)
		Byte 2 - 3: Key pressed and Key release
		0030h - KEYPAD_KEY_0
		0031h - KEYPAD_KEY_1
		0032h - KEYPAD_KEY_2
		0033h - KEYPAD_KEY_3
		0034h - KEYPAD_KEY_4
		0035h - KEYPAD_KEY_5
		0036h - KEYPAD_KEY_6
		0037h - KEYPAD_KEY_7
		0038h - KEYPAD_KEY_8
		0039h - KEYPAD_KEY_9
		Byte 2 - 3: Only Key pressed
		000Dh - KEYPAD_KEY_ENTER
		0008h - KEYPAD_KEY_CLEAR
		001Bh - KEYPAD_KEY_CANCEL
		0070h - FUNC_KEY_F1 (Vend III)
		0071h - FUNC_KEY_F2 (Vend III)
		0072h - FUNC_KEY_F3 (Vend III)
		0073h - FUNC_KEY_F4 (Vend III)

| Touchscreen Event | 00030004h | Length = 1 - 33 bytes |
 || | Byte 1: State (not used) |
		Byte 2 - 33: Image name (zero terminated)

| Slideshow Event | 00030005h | Length = 1 byte |
		Byte 1: State (not used)

| Transaction Event | 00030006h | Length = 9 bytes |

|| | Byte 1: State (not used) |

|| | Byte 2 - 5: Card type (0 = unknown) |

|| | Byte 6 - 9: Status - A four byte, big-endian field |

|| | Byte 9 is used to store the 1-byte status code |

|| | 00 - SUCCESS |

|| | 08 - TIMEOUT |

|| | 0A - FAILED |

|| | This is not related to the extended status codes |

| ----- | ----- | ----- |

| Radio Button Event | 00030007h | Length = 73 bytes |

|| | Byte 1: State (1 = Change ins selected button, other values RFU) |

|| | Byte 2 - 33: Null terminated group name |

|| | Byte 34 - 65: Radio button caption |

Returns

RETURN_CODE: Values can be parsed with [device_getIDGStatusCodeString\(\)](#)

11.2.4.270 lcd_getInputEvent_Len()

```
int lcd_getInputEvent_Len (
    IN int timeout,
    OUT int * dataReceived,
    OUT BYTE * eventType,
    IN_OUT int * eventTypeLen,
    OUT BYTE * graphicsID,
    IN_OUT int * graphicsIDLen,
    OUT BYTE * eventData,
    IN_OUT int * eventDataLen )
```

Requests input from the reader.

Parameters

<i>timeout</i>	Timeout amount in seconds 0 - No timeout
<i>dataReceived</i>	Indicates if an event occurred and data was received 0 - No data received 1 - Data received
<i>eventType</i>	The event type (required to be at least 4 bytes), see table below
<i>eventTypeLen</i>	Length of eventType
<i>graphicsID</i>	The graphicID of the event (required to be at least 4 bytes)
<i>graphicsIDLen</i>	length of graphicID
<i>eventData</i>	The event data, see table below (required to be at least 73 bytes) Event Type Value (4 bytes) Event Specific Data

| ----- | ----- | ----- |

| Button Event | 00030000h | Length = Variable |

|| | Byte 1: State (1 = Pressed, other values RFU) |

```

| | | Byte 2 - n: Null terminated caption |
| ----- | ----- | ----- |
| Checkbox Event | 00030001h | Length = 1 byte |
| | | Byte 1: State (1 = Checked, 0 = Unchecked) |
| ----- | ----- | ----- |
| Line Item Event | 00030002h | Length = 5 bytes |
| | | Byte 1: State (1 = Item Selected, other values RFU) |
| | | Byte 2 - n: Caption of the selected item |
| ----- | ----- | ----- |
| Keypad Event | 00030003h | Length - 3 bytes |
| | | Byte 1: State (1 = key pressed, 2 = key released, other values RFU) |
| | | Byte 2 - 3: Key pressed and Key release |
| | | 0030h - KEYPAD_KEY_0 |
| | | 0031h - KEYPAD_KEY_1 |
| | | 0032h - KEYPAD_KEY_2 |
| | | 0033h - KEYPAD_KEY_3 |
| | | 0034h - KEYPAD_KEY_4 |
| | | 0035h - KEYPAD_KEY_5 |
| | | 0036h - KEYPAD_KEY_6 |
| | | 0037h - KEYPAD_KEY_7 |
| | | 0038h - KEYPAD_KEY_8 |
| | | 0039h - KEYPAD_KEY_9 |
| | | Byte 2 - 3: Only Key pressed |
| | | 000Dh - KEYPAD_KEY_ENTER |
| | | 0008h - KEYPAD_KEY_CLEAR |
| | | 001Bh - KEYPAD_KEY_CANCEL |
| | | 0070h - FUNC_KEY_F1 (Vend III) |
| | | 0071h - FUNC_KEY_F2 (Vend III) |
| | | 0072h - FUNC_KEY_F3 (Vend III) |
| | | 0073h - FUNC_KEY_F4 (Vend III) |
| ----- | ----- | ----- |
| Touchscreen Event | 00030004h | Length = 1 - 33 bytes |
| | | Byte 1: State (not used) |
| | | Byte 2 - 33: Image name (zero terminated) |
| ----- | ----- | ----- |
| Slideshow Event | 00030005h | Length = 1 byte |
| | | Byte 1: State (not used) |
| ----- | ----- | ----- |
| Transaction Event | 00030006h | Length = 9 bytes |
| | | Byte 1: State (not used) |

```

		Byte 2 - 5: Card type (0 = unknown)
		Byte 6 - 9: Status - A four byte, big-endian field
		Byte 9 is used to store the 1-byte status code
		00 - SUCCESS
		08 - TIMEOUT
		0A - FAILED
		This is not related to the extended status codes
Radio Button Event	00030007h	Length = 73 bytes
		Byte 1: State (1 = Change ins selected button, other values RFU)
		Byte 2 - 33: Null terminated group name
		Byte 34 - 65: Radio button caption

Parameters

<i>eventDataLen</i>	Length of eventData
---------------------	---------------------

Returns

RETURN_CODE: Values can be parsed with [device_getIDGStatusCodeString\(\)](#)

11.2.4.271 lcd_getInputFieldValue()

```

int lcd_getInputFieldValue (
    IN BYTE * graphicId,
    OUT BYTE * retData,
    IN_OUT int * retDataLen )
  
```

Get the keypad data that was entered into the specified Input Field.

Parameters

<i>graphicsID</i>	The graphicID of the input field (required to be 4 bytes)
<i>retData</i>	return keypad data
<i>retDataLen</i>	The length of retData

Returns

RETURN_CODE: Values can be parsed with [device_getIDGStatusCodeString\(\)](#)

11.2.4.272 lcd_getSelectedListItem()

```

int lcd_getSelectedListItem (
    IN BYTE * listGraphicsID,
    OUT char * itemID )
  
```

DEPRECATED : please use [lcd_getSelectedListItem_Len\(IN BYTE *listGraphicsID, OUT char *itemID, IN_OUT int *itemIDLen\)](#)

Retrieves the selected item's ID.

Parameters

<i>listGraphicsID</i>	Existing list's graphics ID (4 byte array) that is provided during creation
<i>itemID</i>	The selected item's ID (Maximum: 32 characters) Need 33 bytes of memory including '\0'

11.2.4.273 lcd_getSelectedListItem_Len()

```
int lcd_getSelectedListItem_Len (
    IN BYTE * listGraphicsID,
    OUT char * itemID,
    IN_OUT int * itemIDLen )
```

Retrieves the selected item's ID.

Parameters

<i>listGraphicsID</i>	Existing list's graphics ID (4 byte array) that is provided during creation
<i>itemID</i>	The selected item's ID (Maximum: 32 characters) Need 33 bytes of memory including '\0'
<i>itemIDLen</i>	Length of itemID

11.2.4.274 lcd_linkUIWithTransactionMessageId()

```
int lcd_linkUIWithTransactionMessageId (
    IN BYTE MessageId,
    IN char * screenName,
    IN int screenNameLen )
```

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 32 bytes including the NULL character)
<i>screenNameLen</i>	Length of screenName

Returns

RETURN_CODE: Values can be parsed with [device_getIDGStatusCodeString\(\)](#)

11.2.4.275 lcd_loadScreenInfo()

```
int lcd_loadScreenInfo ( )
```

Load Screen Info

Load all current screen information from RAM to flash

Returns

RETURN_CODE: Values can be parsed with [device_getIDGStatusCodeString\(\)](#)

11.2.4.276 lcd_queryObjectbyID()

```
int lcd_queryObjectbyID (
    IN int objectID,
    OUT int * objectNumbers,
    OUT IDTScreenInfo * screenInfo )
```

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>objectNumbers</i>	Number of the checked object
<i>screenInfo</i>	screen names containing the item

Returns

RETURN_CODE: Values can be parsed with [device_getIDGStatusCodeString\(\)](#)

11.2.4.277 lcd_queryObjectbyName()

```
int lcd_queryObjectbyName (
    IN char * objectName,
    IN int objectNameLen,
    IN_OUT int * objectNumbers,
    OUT IDTScreenInfo * screenInfo )
```

Queery 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>objectNameLen</i>	Length of objectName
<i>objectNumbers</i>	Number of the checked object
<i>screenInfo</i>	Array of all the screen names that contain the object

Returns

RETURN_CODE: Values can be parsed with [device_getIDGStatusCodeString\(\)](#)

11.2.4.278 lcd_queryScreenbyID()

```
int lcd_queryScreenbyID (
    IN int screenID,
    OUT int * result,
    OUT int * screenName,
    IN_OUT int * screenNameLen )
```

Queery 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>screenNameLen</i>	Length of screenName
<i>ip</i>	Optional: IP address to execute command on (for IP connected devices)

Returns

RETURN_CODE: Values can be parsed with [device_getIDGStatusCodeString\(\)](#)

11.2.4.279 lcd_queryScreenbyName()

```
int lcd_queryScreenbyName (
    IN char * screenName,
    IN int screenNameLen,
    OUT int * result )
```

Queery Screen by Name

Check if the given screen exists or not

Parameters

<i>screenName</i>	Name of the checked screen
<i>screenNameLen</i>	Length of screenName
<i>result</i>	the checking result

Returns

RETURN_CODE: Values can be parsed with [device_getIDGStatusCodeString\(\)](#)

11.2.4.280 lcd_registerCallBk()

```
void lcd_registerCallBk (
    pLCD_callBack pLCDf )
```

To register the lcd callback function to get the LCDItem. (Pass NULL to disable the callback.)

11.2.4.281 lcd_removeItem()

```
int lcd_removeItem (
    IN char * screenName,
    IN int  screenNameLen,
    IN char * objectName,
    IN int  objectNameLen )
```

Removed Item

Removes a component.

Parameters

<i>screenName</i>	Screen name where to remove the target from.
<i>screenNameLen</i>	Length of screenName.
<i>objectName</i>	Identifier of the component
<i>objectNameLen</i>	Length of objectName.

Returns

RETURN_CODE: Values can be parsed with [device_getIDGStatusCodeString\(\)](#)

11.2.4.282 lcd_resetInitialState()

```
int lcd_resetInitialState ( )
```

Reset to Initial State. This command places the reader UI into the idle state and displays the appropriate idle display.

Returns

RETURN_CODE: Values can be parsed with [device_getIDGStatusCodeString\(\)](#)

11.2.4.283 lcd_savePrompt()

```
int lcd_savePrompt (
    int  promptNumber,
    char * prompt,
    int  promptLen )
```

Save Prompt

Saves a message prompt to L80 or L100 memory.

Parameters

<i>promptNumber</i>	Prompt number (0-9)
---------------------	---------------------

Parameters

<i>prompt</i>	Prompt string (up to 20 characters)
<i>promptLen</i>	length of prompt

Returns

RETURN_CODE: Values can be parsed with `device_getResponseCodeString`

11.2.4.284 lcd_setBackgroundImage()

```
int lcd_setBackgroundImage (
    IN char * file,
    IN int fileLen,
    IN int enable )
```

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)
<i>fileLen</i>	Length of files
<i>enable</i>	TRUE = Use Background Image, FALSE = Use Background Color

Returns

RETURN_CODE: Values can be parsed with [device_getIDGStatusCodeString\(\)](#)

11.2.4.285 lcd_setBacklight()

```
int lcd_setBacklight (
    IN BYTE isBacklightOn,
    IN BYTE backlightVal )
```

Set Backlight

Set backlight percentage. If the percent > 100, it will be rejected. If 0 < percent < 10, backlight percent will be set to 10.

Parameters

<i>isBacklightOn</i>	0: backlight off 1: backlight on
<i>backlightVal</i>	Backlight percent value to be set

Returns

RETURN_CODE: Values can be parsed with [device_getIDGStatusCodeString\(\)](#)

11.2.4.286 lcd_setDisplayImage()

```
int lcd_setDisplayImage (
    IN char * file,
    IN int fileLen,
    IN int posX,
    IN int posY,
    IN int posMode,
    IN int touchEnable,
    IN int clearScreen )
```

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>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)
<i>fileLen</i>	Length of files
<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) • 2 - Center on screen
<i>touchEnable</i>	TRUE = Image is touch sensitive
<i>clearScreen</i>	TRUE = Clear screen

Returns

RETURN_CODE: Values can be parsed with [device_getIDGStatusCodeString\(\)](#)

11.2.4.287 lcd_setForeBackColor()

```
int lcd_setForeBackColor (
    IN BYTE * foreRGB,
    IN int foreRGBLen,
    IN BYTE * backRGB,
    IN int backRGBLen )
```

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>Length</i>	of foreRGB. Must be 3.
<i>backRGB</i>	Background RGB. 000000 = black, FF0000 = red, 00FF00 = green, 0000FF = blue, FFFFFFFF = white
<i>Length</i>	of backRGB. Must be 3.

Returns

RETURN_CODE: Values can be parsed with [device_getIDGStatusCodeString\(\)](#)

11.2.4.288 lcd_showScreen()

```
int lcd_showScreen (
    IN char * screenName,
    IN int  screenNameLen )
```

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>screenNameLen</i>	Length of <code>screenName</code> .

Returns

RETURN_CODE: Values can be parsed with [device_getIDGStatusCodeString\(\)](#)

11.2.4.289 lcd_startSlideShow()

```
int lcd_startSlideShow (
    IN char * files,
    IN int  filesLen,
    IN int  posX,
    IN int  posY,
    IN int  posMode,
    IN int  touchEnable,
    IN int  recursion,
    IN int  touchTerminate,
    IN int  delay,
    IN int  loops,
    IN int  clearScreen )
```

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>filesLen</i>	Length of files
<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) • 2 - Center on screen

Parameters

<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)
<i>delay</i>	Number of seconds between image displays
<i>loops</i>	Number of display loops. A zero indicates continuous display
<i>clearScreen</i>	TRUE = Clear screen

Returns

RETURN_CODE: Values can be parsed with [device_getIDGStatusCodeString\(\)](#)

11.2.4.290 lcd_storeScreenInfo()

```
int lcd_storeScreenInfo ( )
```

Store Screen Info

Store all current screen information from RAM to flash

Returns

RETURN_CODE: Values can be parsed with [device_getIDGStatusCodeString\(\)](#)

11.2.4.291 lcd_updateColor()

```
int lcd_updateColor (
    IN char * screenName,
    IN int  screenNameLen,
    IN char * objectName,
    IN int  objectNameLen,
    IN BYTE * color,
    IN int  colorLen )
```

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>screenNameLen</i>	Length of screenName.
<i>objectName</i>	Identifier of the component
<i>objectNameLen</i>	Length of objectName.

Parameters

<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>colorLen</i>	Length of color.

Returns

RETURN_CODE: Values can be parsed with [device_getIDGStatusCodeString\(\)](#)

11.2.4.292 lcd_updateLabel()

```
int lcd_updateLabel (
    IN char * screenName,
    IN int screenNameLen,
    IN char * objectName,
    IN int objectNameLen,
    IN char * label,
    IN int labelLen )
```

Update Label

Updates the component label.

Parameters

<i>screenName</i>	Screen name that will be the target of update label
<i>screenNameLen</i>	Length of screenName.
<i>objectName</i>	Identifier of the component
<i>objectNameLen</i>	Length of objectName.
<i>label</i>	Label to show on the component
<i>labelLen</i>	Length of label.

Returns

RETURN_CODE: Values can be parsed with [device_getIDGStatusCodeString\(\)](#)

11.2.4.293 lcd_updatePosition()

```
int lcd_updatePosition (
    IN char * screenName,
    IN int  screenNameLen,
    IN char * objectName,
    IN int  objectNameLen,
    IN BYTE alignment,
    IN int  new_xCord,
    IN int  new_yCord )
```

Update Position

Updates the component position.

Parameters

<i>screenName</i>	Screen Name that will be the target of update position
<i>screenNameLen</i>	Length of screenName.
<i>objectName</i>	Identifier of the component
<i>objectNameLen</i>	Length of objectName.
<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 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>new_xCord</i>	x-coordinate for Text, range 0-271
<i>new_yCord</i>	y-coordinate for Text, range 0-479

Returns

RETURN_CODE: Values can be parsed with [device_getIDGStatusCodeString\(\)](#)

11.2.4.294 loyalty_cancelTransaction()

```
int loyalty_cancelTransaction ( )
```

Cancel Loyalty Transaction Only for VP6800

Cancels the currently executing transaction.

Returns

RETURN_CODE: Values can be parsed with [device_getIDGStatusCodeString\(\)](#)

11.2.4.295 loyalty_cancelTransactionSilent()

```
int loyalty_cancelTransactionSilent (
    int enable )
```

Cancel Loyalty Transaction Silent Only for VP6800

Cancel transaction with or without showing the LCD message

Parameters

<i>enable</i>	0: With LCD message 1: Without LCD message
---------------	--

Returns

success or error code. Values can be parsed with `device_getIDGStatusCodeString`

11.2.4.296 `loyalty_registerCallBk()`

```
void loyalty_registerCallBk (
    pEMV_callback pEMVf )
```

To register the loyalty callback function to get the EMV processing response. (Pass NULL to disable the callback.)
Only for VP6800

11.2.4.297 `loyalty_startTransaction()`

```
int loyalty_startTransaction (
    IN double amount,
    IN double amtOther,
    IN int type,
    IN const int _timeout,
    IN BYTE * tags,
    IN int tagsLen,
    IN const int cardType,
    IN const int iccReadType )
```

Start Loyalty Transaction Request Only for VP6800

Authorizes the transaction for an MSR/ICC card

The tags will be returned in the callback routine.

Parameters

<i>amount</i>	Transaction amount value (tag value 9F02) • SEE IMPORTANT NOTE BELOW
<i>amtOther</i>	Other amount value, if any (tag value 9F03) • SEE IMPORTANT NOTE BELOW
<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. 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>tagsLen</i>	The length of tags data buffer.

>>>>>IMPORTANT: parameters for amount and amtOther MUST BE PASSED AS A DOUBLE VALUE WITH DECIMAL POINT. Example, do not pass 1, but instead pass 1.0 or 1.00. Otherwise, results will be unpredictable

Parameters

<i>cardType</i>	Set available card to accept. 0 = MSR only. 1 = MSR and ICC.
<i>iccReadType</i>	In case of ICC reading, this is how to behave. 0 = Same as device_startTransaction 1 = When reading ICC, read DF4F(JIS2EquivalentData) in ReadRecord. If the user swipes an IC card, the reader will ask for using ICC 3 = When reading ICC, read DF4F(JIS2EquivalentData) in ReadRecord. If the user swipes an IC card, the reader will not ask for using ICC and output MSR data directly

Returns

RETURN_CODE: Values can be parsed with [device_getIDGStatusCodeString\(\)](#) Note: if auto poll is on, it will return the error IDG_P2_STATUS_CODE_COMMAND_NOT_ALLOWED

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 device_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) 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

11.2.4.298 `msr_cancelMSRSwipe()`

```
int msr_cancelMSRSwipe ( )
```

Disable MSR Swipe Cancels MSR swipe request.

Returns

RETURN_CODE: Values can be parsed with [device_getIDGStatusCodeString\(\)](#)

11.2.4.299 `msr_captureMode()`

```
int msr_captureMode (
    int isBufferMode,
    int withNotification )
```

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"> • 1: Enter into Buffer mode. • 0: Enter into Auto mode. KB mode only. Swipes automatically captured and sent to keyboard buffer
<i>withNotification</i>	<ul style="list-style-type: none"> • 1: with notification when swiped MSR data. • 0: without notification when swiped MSR data.

Returns

RETURN_CODE: Values can be parsed with `device_getResponseCodeString`

11.2.4.300 `msr_clearMSRData()`

```
int msr_clearMSRData ( )
```

Clear MSR Data Clears the MSR Data buffer

Returns

RETURN_CODE: Values can be parsed with [device_getIDGStatusCodeString\(\)](#)

11.2.4.301 msr_disable()

```
int msr_disable ( )
```

Disable MSR Disable MSR functions.

Returns

RETURN_CODE: Values can be parsed with [device_getIDGStatusCodeString\(\)](#)

11.2.4.302 msr_flushTrackData()

```
int msr_flushTrackData ( )
```

Flush Track Data Clears any track data being retained in memory by future PIN Block request.

Returns

RETURN_CODE: Values can be parsed with [device_getIDGStatusCodeString\(\)](#)

11.2.4.303 msr_getClearPANID()

```
int msr_getClearPANID (
    BYTE * value )
```

Get Clear PAN ID.

Returns the number of digits that begin the PAN that will be in the clear

Parameters

<i>value</i>	4901 <Setting value>="". setting Value: Number of digits in clear. Values are char '0' - '6'
--------------	--

Returns

RETURN_CODE: Values can be parsed with [device_getResponseCodeString](#)

11.2.4.304 msr_getExpirationMask()

```
int msr_getExpirationMask (
    BYTE * value )
```

Get MSR expiration date mask.

Parameters

<i>value</i>	5001 <Setting value>="">. setting Value: '0' = masked, '1' = not-masked
--------------	---

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString

11.2.4.305 msr_getFunctionStatus()

```
int msr_getFunctionStatus (
    int * enable,
    int * isBufferMode,
    int * withNotification )
```

Get MSR Function Status.

Gets the MSR function status

Parameters

<i>enable</i>	1 = MSR enabled, 0 = MSR disabled
<i>isBufferMode</i>	1 = buffer mode, 0 = auto mode
<i>withNotification</i>	1 = with notification, 0 = without notification

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString

11.2.4.306 msr_getKeyFormatForICCDUKPT()

```
int msr_getKeyFormatForICCDUKPT (
    OUT BYTE * format )
```

Get Key Format For DUKPT

Specifies how data will be encrypted with Data Key or PIN key (if DUKPT key loaded). This applies to both MSR and ICC

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.
---------------	---

Returns

RETURN_CODE: Values can be parsed with `device_getResponseCodeString`

11.2.4.307 msr_getKeyTypeForICCDUKPT()

```
int msr_getKeyTypeForICCDUKPT (
    OUT BYTE * type )
```

Get Key Type for DUKPT

Specifies the key type used for ICC DUKPT encryption. This applies to both MSR and ICC.

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.

Returns

RETURN_CODE: Values can be parsed with `device_getResponseCodeString`

11.2.4.308 msr_getMSRData()

```
int msr_getMSRData (
    OUT BYTE * reData,
    IN_OUT int * reLen )
```

Get MSR Data Reads the MSR Data buffer**Parameters**

<i>reData</i>	Card data
<i>reLen</i>	Card data length

Returns

RETURN_CODE: Values can be parsed with `device_getIDGStatusCodeString()`

11.2.4.309 msr_getSwipeForcedEncryptionOption()

```
int msr_getSwipeForcedEncryptionOption (
    BYTE * option )
```

Get MSR Swipe Forced Encryption Option.

Parameters

<i>option</i>	8401 <Setting value>="". Setting Value 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
---------------	---

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString

11.2.4.310 msr_getSwipeMaskOption()

```
int msr_getSwipeMaskOption (
    BYTE * option )
```

Get MSR Swipe Mask Option.

Gets the swipe mask/clear data sending option

Parameters

<i>option</i>	8601 <Setting value>="". Setting Value 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
---------------	---

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString

11.2.4.311 msr_registerCallBk()

```
void msr_registerCallBk (
    pMSR_callBack pMSRf )
```

To register the msr callback function to get the MSR card data. (Pass NULL to disable the callback.)

11.2.4.312 msr_registerCallBkp()

```
void msr_registerCallBkp (
    pMSR_callBackp pMSRf )
```

To register the msr callback function to get the MSR card data pointer. (Pass NULL to disable the callback.)

11.2.4.313 msr_setClearPANID()

```
int msr_setClearPANID (
    BYTE val )
```

Set Clear PAN ID.

Parameters

<i>val</i>	Set Clear PAN ID to value: Number of digits to show in clear. Range 0-6.
------------	--

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString

11.2.4.314 msr_setExpirationMask()

```
int msr_setExpirationMask (
    int mask )
```

Set Expiration Masking

Sets the flag to mask the expiration date

Parameters

<i>mask</i>	TRUE = mask expiration
-------------	------------------------

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString

11.2.4.315 msr_setKeyFormatForICCDUKPT()

```
int msr_setKeyFormatForICCDUKPT (
    IN BYTE format )
```

Set Key Format for DUKPT

Sets how data will be encrypted, with either TDES or AES (if DUKPT key loaded) This applies to both MSR and ICC

Parameters

<i>format</i>	encryption Encryption Type <ul style="list-style-type: none">• 00: Encrypt with TDES• 01: Encrypt with AES
---------------	---

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString

11.2.4.316 msr_setKeyTypeForICCDUKPT()

```
int msr_setKeyTypeForICCDUKPT (
    IN BYTE type )
```

Set Key Type for DUKPT Key

Sets which key the data will be encrypted with, with either Data Key or PIN key (if DUKPT key loaded) This applies to both MSR and ICC

Parameters

<i>type</i>	Encryption Type
	<ul style="list-style-type: none">• 00: Encrypt with Data Key• 01: Encrypt with PIN Key

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString

11.2.4.317 msr_setSwipeForcedEncryptionOption()

```
int msr_setSwipeForcedEncryptionOption (
    int track1,
    int track2,
    int track3,
    int track3card0 )
```

Set MSR Swipe Forced Encryption Option.

Parameters

<i>track1</i>	Set track1 encryption to true or false.
<i>track2</i>	Set track2 encryption to true or false.
<i>track3</i>	Set track3 encryption to true or false.
<i>track3card0</i>	Set track3 card0 encryption to true or false.

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString

11.2.4.318 msr_setSwipeMaskOption()

```
int msr_setSwipeMaskOption (
    int track1,
    int track2,
    int track3 )
```

Set MSR Swipe Mask Option.

Sets the swipe mask/clear data sending option

Parameters

<i>track1</i>	Set track1 mask to true or false.
<i>track2</i>	Set track2 mask to true or false.
<i>track3</i>	Set track3 mask to true or false.

Returns

RETURN_CODE: Values can be parsed with `device_getResponseCodeString`

11.2.4.319 msr_startMSRSwipe()

```
int msr_startMSRSwipe (
    IN int _timeout )
```

Start MSR Swipe Enables MSR, waiting for swipe to occur. Allows track selection. Returns IDTMSRData instance to `MSR_callBack()`

Parameters

<i>timeout</i>	Swipe Timeout Value
----------------	---------------------

Returns

RETURN_CODE: Values can be parsed with [device_getIDGStatusCodeString\(\)](#) Note: if auto poll mode is on, it will return command not allowed error

11.2.4.320 parseMSRData()

```
void parseMSRData (
    IN BYTE * resData,
    IN int resLen,
    IN_OUT IDTMSRData * cardData )
```

Parser the MSR data from the buffer into IDTMSTData structure

Parameters

<i>resData</i>	MSR card data buffer
<i>resLen</i>	the length of resData
<i>cardData</i>	the parser result with IDTMSTData structure

11.2.4.321 parsePINBlockData()

```
void parsePINBlockData (
    IN BYTE * resData,
    IN int resLen,
    IN_OUT IDTPINData * cardData )
```

Parse the PIN block data from the buffer into IDTPINData structure

Parameters

<i>resData</i>	PIN card data buffer
<i>resLen</i>	the length of resData
<i>cardData</i>	the parser result with IDTPINData structure

11.2.4.322 parsePINData()

```
void parsePINData (
    IN BYTE * resData,
    IN int resLen,
    IN_OUT IDTPINData * cardData )
```

Parse the PIN data from the buffer into IDTPINData structure

Parameters

<i>resData</i>	PIN card data buffer
<i>resLen</i>	the length of resData
<i>cardData</i>	the parser result with IDTPINData structure

11.2.4.323 pin_cancelPINEntry()

```
int pin_cancelPINEntry ( )
```

Cancel PIN Entry

Cancels PIN entry request

Returns

RETURN_CODE: Values can be parsed with [device_getResponseCodeString\(\)](#)

11.2.4.324 pin_capturePin()

```
int pin_capturePin (
    IN int timeout,
    IN int type,
    IN char * PAN,
    IN int PANLen,
    IN int minPIN,
    IN int maxPIN,
    IN char * message,
    IN int messageLen )
```

Capture PIN

Parameters

<i>timeout</i>	Timeout, in seconds. Value of 0 will use system timeout, if any
----------------	---

Parameters

<i>type</i>	PAN and Key Type <ul style="list-style-type: none">• 00h = MKSK to encrypt PIN, Internal PAN (from MSR)• 01h = DUKPT to encrypt PIN, Internal PAN (from MSR)• 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)
<i>PANLen</i>	Length of PAN
<i>minPIN</i>	Minimum PIN Length
<i>maxPIN</i>	Maximum PIN Length
<i>message</i>	LCD Message
<i>messageLen</i>	Length of message

Returns

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

11.2.4.325 `pin_capturePinExt()`

```
int pin_capturePinExt (  
    IN int type,  
    IN char * PAN,  
    IN int PANLen,  
    IN int minPIN,  
    IN int maxPIN,  
    IN char * message,  
    IN int messageLen,  
    IN char * verify,  
    IN int verifyLen )
```

- Capture PIN Ext

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>message</i>	LCD Message Up to 2 lines of text, each line 1-16, separated by 0x00
<i>messageLen</i>	Length of 1st scenario LCD message, valid in 00h~21h (0~33).If no LCD message input, length is 00h, and display default msg "PLEASE ENTER PIN"
<i>verify</i>	LCD Message Up to 2 lines of text, each line 1-16, separated by 0x00
<i>verifyLen</i>	Length of 2nd Scenario LCD message.valid in 00h~21h (0~33).This field is present only when PAN and Key Type has Verify PIN.If no LCD message input, length is 00h, and display default msg " ENTER PIN AGAIN"

Returns

RETURN_CODE: Values can be parsed with [device_getIDGStatusCodeString\(\)](#)

11.2.4.326 pin_getEncryptedOnlinePIN()

```
int pin_getEncryptedOnlinePIN (
    IN int keyType,
    IN int timeout )
```

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

Returns

RETURN_CODE: Values can be parsed with [device_getResponseCodeString\(\)](#)

11.2.4.327 pin_getEncryptedPIN()

```
int pin_getEncryptedPIN (
    int keyType,
    char * PAN,
    int PANLen,
    char * message,
    int messageLen,
    int timeout )
```

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>PANLen</i>	length of PAN
<i>message</i>	Message to display
<i>messageLen</i>	length of message
<i>timeout</i>	PIN entry timeout

Returns

RETURN_CODE: Values can be parsed with [errorCode.getErrorString\(\)](#)

11.2.4.328 pin_getFunctionKey()

```
int pin_getFunctionKey (
    int timeout )
```

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

@param timeout Timeout, in seconds

Returns

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

11.2.4.329 pin_getPAN()

```
int pin_getPAN (
    IN int getCSC,
    IN int timeout )
```

Get PAN

Requests PAN Entry on pinpad

Parameters

<i>getCSC</i>	Include Customer Service Code (also known as CVV, CVC)
<i>timeout</i>	PAN entry timeout

Returns

RETURN_CODE: Values can be parsed with [device_getResponseCodeString\(\)](#)

11.2.4.330 pin_getPanEntry()

```
int pin_getPanEntry (
    IN int csc,
    IN int expDate,
    IN int ADR,
    IN int ZIP,
    IN int mod10CK,
    IN int timeout,
    IN int encPANOnly )
```

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>csc</i>	Request CSS
<i>expDate</i>	Request Expiration Date
<i>ADR</i>	Request Address
<i>ZIP</i>	Request Zip
<i>mod10CK</i>	Validate entered PAN passes MOD-10 checking before accepting
<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>encPANOnly</i>	Output only encrypted PAN

Returns

RETURN_CODE: Values can be parsed with `device_getResponseCodeString`

11.2.4.331 pin_getPIN()

```
int pin_getPIN (
    IN int mode,
    IN int PANSource,
    IN char * iccPAN,
    IN int iccPANLen,
    IN int startTimeout,
    IN int entryTimeout,
    IN char * language,
    IN int languageLen )
```

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)
<i>PANSource</i>	<ul style="list-style-type: none"> • 0x00- ICC: PAN Captured from ICC and must be provided in iccPAN parameter • 0x01- MSR: PAN Captured from MSR swipe and will be inserted by Spectrum Pro. No need to provide iccPAN parameter.
<i>iccPAN</i>	PAN captured from ICC. When PAN captured from MSR, this parameter will be ignored
<i>iccPANLen</i>	the length of iccPAN
<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>languageLen</i>	the length of language

Returns

RETURN_CODE: Values can be parsed with [device_getResponseCodeString\(\)](#)

11.2.4.332 pin_inputFromPrompt()

```
int pin_inputFromPrompt (
    BYTE mask,
```

```

    BYTE preClearText,
    BYTE postClearText,
    int minLen,
    int maxLen,
    char * lang,
    BYTE promptID,
    char * defaultResponse,
    int defaultResponseLen,
    int timeout )

```

Get PIN Input from Prompt Results returned to PIN Callback. If successful function key capture, data is returned as IDTPINData.keyString.

```

    @param mask 0 = no masking, 1 = Display "*" for numeric key according to Pre-Cleartext and Post-Cleartext
    @param preClearText Range 0-6 Characters to mask at start of text if masking is on;
    @param postClearText Range 0-6 Characters to mask at end of text if masking is on;
    @param minLen Minimum number of digits required to input
    @param maxLen Maximum number of digits allowed to be input
    @param lang Valid values;
    "EN" is English display message
    "JP" is Japanese display message
    "ES" is Spanish display message
    "FR" is French display message
    "ZH" is Chinese display message
    "PT" is Portuguese display message
    @param promptID Valid values:
    0x00: Enter Phone Number
    0x01: Enter IP Address
    0x02: Enter Subnet Mask
    0x03: Enter Default Gateway
    0x04: Enter Odometer Reading/Mileage
    0x05: Enter Employee ID
    0x06: Enter Password for Default Factory Setting
    0x07: Enter Email Address (Full keyboard)
    @param defaultResponse Default String on input field
    @param defaultResponseLen Length of defaultResponse
    @param timeout Timeout, in seconds

```

Returns

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

11.2.4.333 pin_promptCreditDebit()

```

int pin_promptCreditDebit (
    IN char * currencySymbol,
    IN int currencySymbolLen,
    IN char * displayAmount,
    IN int displayAmountLen,
    IN int timeout,
    OUT BYTE * retData,
    IN_OUT int * retDataLen )

```

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>currencySymbolLen</i>	length of currencySymbol

Parameters

<i>displayAmount</i>	Amount to display (can be NULL)
<i>displayAmountLen</i>	length of displayAmount
<i>timeout</i>	Menu entry timeout. Valid values 2-20 seconds
<i>retData</i>	If successful, the return code is zero and the data is 1 byte (0: Credit 1: Debit). If the return code is not zero, it may be a four-byte Extended Status Code
<i>currencySymbolLen</i>	length of currencySymbol

Returns

RETURN_CODE: Values can be parsed with [device_getResponseCodeString\(\)](#)

11.2.4.334 pin_promptForAmount()

```
int pin_promptForAmount (
    IN int timeout,
    IN int minLen,
    IN int maxLen,
    IN char * message,
    IN int messageLen,
    BYTE * signature,
    IN int signatureLen )
```

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>messageLen</i>	Length of 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>signatureLen</i>	Length of signature

Returns

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

11.2.4.335 pin_promptForAmountInput()

```
int pin_promptForAmountInput (
```

```

int messageID,
int languageID,
int minLen,
int maxLen,
int timeout )

```

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 INGR↵ESE	SVP ENTREZ
6	PLEASE REENTER	POR FAVO REENT↵RAR	POR FAVO REINGR↵ESE	SVP RE-ENTREZ
7	PO NUMBER	N?MERO PO	NUMERO PO	No COMMANDE
8	DRIVER ID	LICEN?A	LICENCIA	ID CONDUCTEUR
9	ODOMETER	ODOMETER	ODOMETRO	ODOMETRE
10	ID NUMBER	N?MERO 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 N?MERO	NUMERO EMP	No TRAVAIL
14	WORK ORDER	TRABALHO ORDEM	ORDEN TRABAJO	FICHE TRAVAIL
15	VEHICLE ID	ID VE?CULO	ID VEHICULO	ID VEHICULE
16	ENTER DRIVER	ENTER DRIVER	INGRESE CONDUCT↵TOR	ENTR CONDUCTEUR
17	ENTER DEPT	ENTER DEPT	INGRESE DEPT	ENTR DEPARTEMNT
18	ENTER PHONE	ADICIONAR PHONE	INGRESE TELEFO↵NO	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 TRAB↵AJO	ENTR ID TRAVAIL
22	ROUTE NUMBER	N?MERO PATH	RUTA NUMERO	No ROUTE
23	ENTER USER ID	ENTER USER ID	INGRESE ID USUA↵RIO	ID UTILISATEUR
24	FLEET NUMBER	N?MERO DE FROTA	FLOTA NUMERO	No PARC AUTO
25	ENTER PRODUCT	ADICIONAR PROD↵UTO	INGRESE PRODUC↵TO	ENTREZ PRODUIT
26	DRIVER NUMBER	N?MERO DRIVER	CONDUCTOR NUM↵ERO	No CONDUCTEUR
27	ENTER LICENSE	ENTER LICEN?A	INGRESE LICENCIA	ENTREZ PERMIS
28	ENTER FLEET NO	ENTER NRO FROTA	INGRESE NRO FLO↵TA	ENT No PARC AUTO
29	ENTER CAR WASH	WASH ENTER	INGRESE LAVADO	ENTREZ LAVE-AUTO
30	ENTER VEHICLE	ENTER VE?CULO	INGRESE VEHICULO	ENTREZ VEHICULE
31	ENTER TRAILER	TRAILER ENTER	INGRESE TRAILER	ENTREZ REMORQ↵UE
32	ENTER ODOMETER	ENTER ODOMETER	INGRESE ODOMET↵RO	ENTREZ ODOMETRE
33	DRIVER LICENSE	CARTEIRA DE MOT↵ORISTA	LICENCIA CONDU↵CTOR	PERMIS CONDUIRE

Msg Id	English Prompt	Portuguese Prompt	Spanish Prompt	French Prompt
34	ENTER CUSTOMER	ENTER CLIENTE	INGRESE CLIENTE	ENTREZ CLIENT
35	VEHICLE NUMBER	N?MERO DO VE?CULO	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 USUARIO	INGRESE INFO USUARIO	INFO UTILISATEUR
39	ENTER CUST CODE	ENTER CODE. CLIENTE	INGRESE COD. CLIENTE	ENTR CODE CLIENT
40	ENTER EMPLOYEE	ENTER FUNCIONARIO	INGRESE EMPLEADO	ENTREZ EMPLOYE
41	ENTER ID NUMBER	ENTER N?MERO 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	N?MERO 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 VE?CULO	REINGRESE VEHICULO	RE-ENTR VEHICULE
48	ENTER VEHICLE ID	ENTER VE?CULO 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 REFER?NCIA	INGRESE REFER?NCIA	ENTREZ REFERENCE
53	ENTER AUTH NUMBER	ENTER N?MERO AUT	INGRESE NUMERO AUT	No AUTORISATION
54	ENTER HUB NUMBER	ENTER HUB NRO	INGRESE NRO HUB	ENTREZ No NOYAU
55	ENTER HUBOMETER	MEDIDA PARA ENTERAR 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

Msg Id	English Prompt	Portuguese Prompt	Spanish Prompt	French Prompt
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
67	VEHICLE #	VE?CULO #	VEHICULO #	# VEHICULE
68	ENTER VEHICLE #	ENTER VE?CULO #	INGRESE VEHICULO #	ENT # VEHICULE
69	JOB #	TRABALHO #	TRABAJO #	# TRAVAIL
70	ENTER JOB #	ENTER JOB #	INGRESE TRABAJO #	ENTREZ # TRAVAIL
71	DEPT NUMBER	N?MERO DEPT	NUMERO DEPTO	No DEPARTEMENT
72	DEPARTMENT #	DEPARTAMENTO #	DEPARTAMENTO #	DEPARTEMENT
73	ENTER DEPT #	ENTER DEPT #	INGRESE DEPTO #	ENT# DEPARTEMENT
74	LICENSE NUMBER	N?MERO DE LICEN?A	NUMERO LICENCIA	No PERMIS
75	LICENSE #	LICEN?A #	LICENCIA #	# PERMIS
76	ENTER LICENSE #	ENTER LICEN?A #	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 N?MERO DE CART?O	INGRESE NUM TARJETA	ENTREZ NO CARTE
86	EXP DATE(Yymm)	VALIDADE VAL (Aamm)	FECHA EXP (Aamm)	DATE EXPIR(Aamm)
87	PHONE NUMBER	TELEFONE	NUMERO TELEFONO	NO TEL
88	CVV START DATE	CVV DATA DE IN?CIO	CVV FECHA INICIO	CVV DATE DE DEBUT
89	ISSUE NUMBER	N?MERO DE EMISS?O	NUMERO DE EMISION	NO DEMISSION
90	START DATE (MmYy)	DATA DE IN?CIO (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
@param timeout Timeout value, in seconds

```

Returns

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

11.2.4.336 pin_promptForKeyInput()

```
int pin_promptForKeyInput (
```

```

int messageID,
int languageID,
int maskInput,
int minLen,
int maxLen,
int timeout )

```

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 INGR↵ESE	SVP ENTREZ
6	PLEASE REENTER	POR FAVO REENT↵RAR	POR FAVO REINGR↵ESE	SVP RE-ENTREZ
7	PO NUMBER	N?MERO PO	NUMERO PO	No COMMANDE
8	DRIVER ID	LICEN?A	LICENCIA	ID CONDUCTEUR
9	ODOMETER	ODOMETER	ODOMETRO	ODOMETRE
10	ID NUMBER	N?MERO 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 N?MERO	NUMERO EMP	No TRAVAIL
14	WORK ORDER	TRABALHO ORDEM	ORDEN TRABAJO	FICHE TRAVAIL
15	VEHICLE ID	ID VE?CULO	ID VEHICULO	ID VEHICULE
16	ENTER DRIVER	ENTER DRIVER	INGRESE CONDUCT↵TOR	ENTR CONDUCTEUR
17	ENTER DEPT	ENTER DEPT	INGRESE DEPT	ENTR DEPARTEMNT
18	ENTER PHONE	ADICIONAR PHONE	INGRESE TELEFO↵NO	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 TRAB↵AJO	ENTR ID TRAVAIL
22	ROUTE NUMBER	N?MERO PATH	RUTA NUMERO	No ROUTE
23	ENTER USER ID	ENTER USER ID	INGRESE ID USUA↵RIO	ID UTILISATEUR
24	FLEET NUMBER	N?MERO DE FROTA	FLOTA NUMERO	No PARC AUTO
25	ENTER PRODUCT	ADICIONAR PROD↵UTO	INGRESE PRODUC↵TO	ENTREZ PRODUIT
26	DRIVER NUMBER	N?MERO DRIVER	CONDUCTOR NUM↵ERO	No CONDUCTEUR
27	ENTER LICENSE	ENTER LICEN?A	INGRESE LICENCIA	ENTREZ PERMIS
28	ENTER FLEET NO	ENTER NRO FROTA	INGRESE NRO FLO↵TA	ENT No PARC AUTO
29	ENTER CAR WASH	WASH ENTER	INGRESE LAVADO	ENTREZ LAVE-AUTO
30	ENTER VEHICLE	ENTER VE?CULO	INGRESE VEHICULO	ENTREZ VEHICULE
31	ENTER TRAILER	TRAILER ENTER	INGRESE TRAILER	ENTREZ REMORQ↵UE
32	ENTER ODOMETER	ENTER ODOMETER	INGRESE ODOMET↵RO	ENTREZ ODOMETRE

Msg Id	English Prompt	Portuguese Prompt	Spanish Prompt	French Prompt
33	DRIVER LICENSE	CARTEIRA DE MOT↵ORISTA	LICENCIA CONDU↵CTOR	PERMIS CONDUIRE
34	ENTER CUSTOMER	ENTER CLIENTE	INGRESE CLIENTE	ENTREZ CLIENT
35	VEHICLE NUMBER	N?MERO DO VE?C↵ULO	VEHICULO NUMERO	No VEHICULE
36	ENTER CUST DATA	ENTER CLIENTE IN↵FO	INGRESE INFO CLI↵ENTE	INFO CLIENT
37	REENTER DRIVID	REENTRAR DRIVER ID	REINGRESE ID CH↵OFER	RE-ENTR ID COND
38	ENTER USER DATA	ENTER INFO USU?↵RIO	INGRESE INFO US↵UARIO	INFO UTILISATEUR
39	ENTER CUST CODE	ENTER CODE. CLI↵ENTE	INGRESE COD. CLI↵ENTE	ENTR CODE CLIENT
40	ENTER EMPLOYEE	ENTER FUNCION?↵RIO	INGRESE EMPLEA↵DO	ENTREZ EMPLOYE
41	ENTER ID NUMBER	ENTER N?MERO ID	INGRESE NUMERO ID	ENTREZ No ID
42	ENTER DRIVER ID	ENTER ID DRIVER	INGRESE ID COND↵UCTOR	No CONDUCTEUR
43	ENTER FLEET PIN	ENTER PIN FROTA	INGRESE PIN DE F↵LOTA	NIP PARC AUTO
44	ODOMETER NUMB↵ER	N?MERO ODOMET↵ER	ODOMETRO NUME↵RO	No ODOMETRE
45	ENTER DRIVER LIC	ENTER DRIVER LIC	INGRESE LIC CON↵DUCTOR	PERMIS CONDUIRE
46	ENTER TRAILER NO	NRO TRAILER ENT↵ER	INGRESE NRO TRA↵ILER	ENT No REMORQUE
47	REENTER VEHICLE	REENTRAR VE?CU↵LO	REINGRESE VEHIC↵ULO	RE-ENTR VEHICULE
48	ENTER VEHICLE ID	ENTER VE?CULO ID	INGRESE ID VEHIC↵ULO	ENTR ID VEHICULE
49	ENTER BIRTH DATE	INSERIR DATA NAC	INGRESE FECHA N↵AC	ENT DT NAISSANCE
50	ENTER DOB MMDD↵YY	ENTER FDN MMDD↵YY	INGRESE FDN MM↵DDAA	NAISSANCE MMJJAA
51	ENTER FLEET DATA	ENTER FROTA INFO	INGRESE INFO DE FLOTA	INFO PARC AUTO
52	ENTER REFERENCE	ENTER REFER?NCIA	INGRESE REFERE↵NCIA	ENTREZ REFEREN↵CE
53	ENTER AUTH NUM↵BR	ENTER N?MERO A↵UT	INGRESE NUMERO AUT	No AUTORISATION
54	ENTER HUB NUMB↵ER	ENTER HUB NRO	INGRESE NRO HUB	ENTREZ No NOYAU
55	ENTER HUBOMETER	MEDIDA PARA ENT↵RAR HUB	INGRESE MEDIDO DE HUB	COMPTEUR NOYAU
56	ENTER TRAILER ID	TRAILER ENTER ID	INGRESE ID TRAIL↵ER	ENT ID REMORQUE
57	ODOMETER READI↵NG	QUILOMETRAGEM	LECTURA ODOME↵TRO	LECTURE ODOME↵TRE
58	REENTER ODOME↵TER	REENTRAR ODOM↵ETER	REINGRESE ODO↵METRO	RE-ENT ODOMETRE
59	REENTER DRIV. ID	REENTRAR DRIVER ID	REINGRESE ID CH↵OFER	RE-ENT ID CONDUCT
60	ENTER CUSTOMER ID	ENTER CLIENTE ID	INGRESE ID CLIEN↵TE	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 # CONDUCTE↵ UR
67	VEHICLE #	VE?CULO #	VEHICULO #	# VEHICULE
68	ENTER VEHICLE #	ENTER VE?CULO #	INGRESE VEHICULO #	ENT # VEHICULE
69	JOB #	TRABALHO #	TRABAJO #	# TRAVAIL
70	ENTER JOB #	ENTER JOB #	INGRESE TRABAJO #	ENTREZ # TRAVAIL
71	DEPT NUMBER	N?MERO DEPT	NUMERO DEPTO	No DEPARTEMENT
72	DEPARTMENT #	DEPARTAMENTO #	DEPARTAMENTO #	DEPARTEMENT
73	ENTER DEPT #	ENTER DEPT #	INGRESE DEPTO #	ENT# DEPARTEME↵ NT
74	LICENSE NUMBER	N?MERO DE LIC↵ EN?A	NUMERO LICENCIA	No PERMIS
75	LICENSE #	LICEN?A #	LICENCIA #	# PERMIS
76	ENTER LICENSE #	ENTER LICEN?A #	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 N?MERO DE CART?O	INGRESE NUM TA↵ RJETA	ENTREZ NO CARTE
86	EXP DATE(YMMM)	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 IN?CIO	CVV FECHA INICIO	CVV DATE DE DEB↵ UT
89	ISSUE NUMBER	N?MERO DE EMI↵ SS?O	NUMERO DE EMISI↵ ON	NO DEMISSION
90	START DATE (MMYY)	DATA DE IN?CIO (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 1 = entry is masked with '*', 0 = 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
@param timeout Timeout value, in seconds

```

Returns

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

11.2.4.337 pin_promptForNumericKey()

```
int pin_promptForNumericKey (
    IN int timeout,
    IN int maskInput,
    IN int minLen,
    IN int maxLen,
    IN char * message,
    IN int messageLen,
    BYTE * signature,
    IN int signatureLen )
```

Capture Numeric Input

Parameters

<i>timeout</i>	Timeout, in seconds. Value of 0 will use system timeout, if any
<i>maskInput</i>	<ul style="list-style-type: none"> • 0 = Display numeric for numeric key on LCD • 1 = Display ?*? for numeric key on LCD
<i>minLen</i>	Minimum input Length
<i>maxLen</i>	Maximum input Length
<i>message</i>	Plaintext Display Message. - 16 chars max
<i>messageLen</i>	Length of 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>signatureLen</i>	Length of signature

Returns

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

11.2.4.338 pin_promptForNumericKeyWithSwipe()

```
int pin_promptForNumericKeyWithSwipe (
    IN int timeout,
    IN BYTE function,
    IN int minLen,
    IN int maxLen,
    IN char * line1,
    IN int line1Len,
    IN char * line2,
    IN int line2Len,
    BYTE * signature,
    IN int signatureLen )
```

Capture Numeric Input

Parameters

<i>timeout</i>	Timeout, in seconds. Value of 0 will use system timeout, if any
<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>line1Len</i>	Length of line1
<i>line2</i>	Line 2 of LCD Message - 16 chars max
<i>line2Len</i>	Length of line2
<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 ?斤??Display Flag><Key max="" length="">< Key Min Length><Plaintext display="" message="">="">?斤?? 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>signatureLen</i>	Length of signature

Returns

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

11.2.4.339 pin_registerCallBk()

```
void pin_registerCallBk (
    pPIN_callback pPINf )
```

*To register the loyalty MSR callback function to get the MSR card data. (Pass NULL to disable the callback.) *To register the loyalty MSR callback function to get the MSR card data pointer. (Pass NULL to disable the callback.) To register the pin callback function to get the PINPad data. (Pass NULL to disable the callback.)

11.2.4.340 pin_sendBeep()

```
int pin_sendBeep (
    int frequency,
    int duration )
```

Send Beep

Executes a beep request.

Parameters

<i>frequency</i>	Frequency, range 200-20000Hz Not used for NEO 2 devices
<i>duration</i>	Duration, range 16-65535ms Not used for NEO 2 devices

Returns

RETURN_CODE: Values can be parsed with `device_getResponseCodeString`

11.2.4.341 pin_setKeyValues()

```
int pin_setKeyValues (
    int mode )
```

Set Key Values

Set return key values on or off

Parameters

<i>mode</i>	On: 1, Off: 0
-------------	---------------

Returns

RETURN_CODE: Values can be parsed with [device_getResponseCodeString\(\)](#)

11.2.4.342 registerHotplugCallBk()

```
void registerHotplugCallBk (
    pMessageHotplug pMsgHotplug )
```

To register the USB HID hot-plug callback function which implemented in the application to monitor the hotplug message from the SDK.

11.2.4.343 registerLogCallBk()

```
void registerLogCallBk (
    pSendDataLog pFSend,
    pReadDataLog pFRead )
```

To register the log callback function which implemented in the application to monitor sending/reading data between application and reader.

11.2.4.344 rs232_device_init()

```
int rs232_device_init (
    int deviceType,
    int port_number,
    int brate )
```

Initial the device by RS232

It will try to connect to the device with provided deviceType, port_number, and brate.

Parameters

<i>deviceType</i>	Device to connect to
<i>port_number</i>	Port number of the device Port nr. Linux Windows

0	ttyS0	COM1
1	ttyS1	COM2
2	ttyS2	COM3
3	ttyS3	COM4
4	ttyS4	COM5
5	ttyS5	COM6
6	ttyS6	COM7
7	ttyS7	COM8
8	ttyS8	COM9
9	ttyS9	COM10
10	ttyS10	COM11
11	ttyS11	COM12
12	ttyS12	COM13
13	ttyS13	COM14
14	ttyS14	COM15
15	ttyS15	COM16
16	ttyUSB0	n.a.
17	ttyUSB1	n.a.
18	ttyUSB2	n.a.
19	ttyUSB3	n.a.
20	ttyUSB4	n.a.
21	ttyUSB5	n.a.
22	ttyAMA0	n.a.
23	ttyAMA1	n.a.
24	ttyACM0	n.a.
25	ttyACM1	n.a.
26	rfcomm0	n.a.
27	rfcomm1	n.a.
28	ircomm0	n.a.
29	ircomm1	n.a.
30	cuau0	n.a.
31	cuau1	n.a.
32	cuau2	n.a.
33	cuau3	n.a.
34	cuaU0	n.a.

35	cuaU1	n.a.
36	cuaU2	n.a.
37	cuaU3	n.a.

Parameters

<i>brate</i>	Bitrate of the device
--------------	-----------------------

Returns

RETURN_CODE: Values can be parsed with [device_getResponseCodeString\(\)](#)

11.2.4.345 SDK_Version()

```
char* SDK_Version ( )
```

To Get SDK version

Returns

return the SDK version string

11.2.4.346 set_open_com_port_timeout()

```
void set_open_com_port_timeout (
    int timeout )
```

Set the timeout value for opening the com port

Parameters

<i>timeout</i>	should be set to greater than 0 in seconds, otherwise there will be no timeout for opening the com port
----------------	---

Returns

RETURN_CODE: Values can be parsed with [device_getResponseCodeString\(\)](#)

11.2.4.347 setAbsoluteLibraryPath()

```
int setAbsoluteLibraryPath (
    const char * absoluteLibraryPath )
```

Set the path to use when searching for ID TECH's libraries. If this is not set, the libraries will be searched for with the system's default procedures.

Parameters

<i>absoluteLibraryPath</i>	The absolute path to ID TECH's libraries.
----------------------------	---

Returns

RETURN_CODE: Values can be parsed with [device_getResponseCodeString\(\)](#)

11.2.4.348 ws_deleteSSLCert()

```
int ws_deleteSSLCert (
    IN char * name,
    IN int nameLen )
```

Delete SSL Certificate Deletes a SSL Certificate by name

Parameters

<i>name</i>	Name of certificate to delete
<i>nameLen</i>	Certificate Name Length

Returns

RETURN_CODE: Values can be parsed with [device_getIDGStatusCodeString\(\)](#)

11.2.4.349 ws_getCertChainType()

```
int ws_getCertChainType (
    OUT int * type )
```

Get Certificate Chain Type Returns indicator for using test/production certificate chain

Parameters

<i>type</i>	0 = test certificate chain, 1 = production certificate chain
-------------	--

Returns

RETURN_CODE: Values can be parsed with [device_getIDGStatusCodeString\(\)](#)

11.2.4.350 ws_loadSSLCert()

```
int ws_loadSSLCert (
    IN char * name,
    IN int nameLen,
    IN char * dataDER,
    IN int dataDERLen )
```

Load SSL Certificate Loads a SSL certificate

Parameters

<i>name</i>	Certificate Name
<i>nameLen</i>	Certificate Name Length

Parameters

<i>dataDER</i>	DER encoded certificate data
<i>dataDERLen</i>	DER encoded certificate data length

Returns

RETURN_CODE: Values can be parsed with [device_getIDGStatusCodeString\(\)](#)

11.2.4.351 ws_requestCSR()

```
int ws_requestCSR (
    OUT RequestCSR * csr )
```

Request CSR Requests 3 sets of public keys: encrypting Keys, signing/validating keys, signing/validating 3rd party apps

Parameters

<i>csr</i>	RequestCSR structure to return the data
------------	---

Returns

RETURN_CODE: Values can be parsed with [device_getIDGStatusCodeString\(\)](#)

11.2.4.352 ws_revokeSSLCert()

```
int ws_revokeSSLCert (
    IN char * name,
    IN int nameLen )
```

Revoke SSL Certificate Revokes a SSL Certificate by name

Parameters

<i>name</i>	Name of certificate to revoke
<i>nameLen</i>	Certificate Name Length

Returns

RETURN_CODE: Values can be parsed with [device_getIDGStatusCodeString\(\)](#)

11.2.4.353 ws_updateRootCertificate()

```
int ws_updateRootCertificate (
    IN char * name,
    IN int nameLen,
    IN char * dataDER,
```

```

    IN int dataDERLen,
    IN char * signature,
    IN int signatureLen )

```

Update Root Certificate Updates the root certificate

Parameters

<i>name</i>	Certificate Name
<i>nameLen</i>	Certificate Name Length
<i>dataDER</i>	DER encoded certificate data
<i>dataDERLen</i>	DER encoded certificate data length
<i>signature</i>	Future Root CA signed (RSASSA PSS SHA256) by current Root CA
<i>signature</i>	length

Returns

RETURN_CODE: Values can be parsed with [device_getIDGStatusCodeString\(\)](#)

11.3 Source_C/libIDT_KioskIII.h File Reference

KioskIII API.

```
#include "IDTDef.h"
```

Macros

- #define [IN](#)
- #define [OUT](#)
- #define [IN_OUT](#)

Typedefs

- typedef void(* [pMessageHotplug](#)) (int, int)
- typedef void(* [pSendDataLog](#)) (BYTE *, int)
- typedef void(* [pReadDataLog](#)) (BYTE *, int)
- typedef void(* [pEMV_callBack](#)) (int, int, BYTE *, int, IDTTransactionData *, EMV_Callback *, int)
- typedef void(* [pMSR_callBack](#)) (int, IDTMSRData)
- typedef void(* [pMSR_callBackp](#)) (int, IDTMSRData *)
- typedef void(* [pPIN_callBack](#)) (int, IDTPINData *)
- typedef void(* [pCMR_callBack](#)) (int, IDTCMRData *)
- typedef void(* [pCSFS_callBack](#)) (BYTE status)
- typedef void(* [ftpComm_callBack](#)) (int, int, int)
- typedef void(* [httpComm_callBack](#)) (BYTE *, int)
- typedef void(* [v4Comm_callBack](#)) (BYTE, BYTE, BYTE *, int)

Functions

- void [registerHotplugCallBk](#) ([pMessageHotplug](#) pMsgHotplug)
- void [registerLogCallBk](#) ([pSendDataLog](#) pFSend, [pReadDataLog](#) pFRead)
- void [emv_registerCallBk](#) ([pEMV_callBack](#) pEMVf)

- void `ctls_registerCallBk` (pMSR_callBack pCTLSf)
- void `ctls_registerCallBkp` (pMSR_callBackp pCTLSf)
- void `pin_registerCallBk` (pPIN_callBack pPINf)
- void `device_registerCameraCallBk` (pCMR_callBack pCMRf)
- void `device_registerCardStatusFrontSwitchCallBk` (pCSFS_callBack pCSFSf)
- char * `SDK_Version` ()
- int `setAbsoluteLibraryPath` (const char *absoluteLibraryPath)
- int `device_init` ()
- int `rs232_device_init` (int deviceType, int port_number, int brate)
- int `device_setCurrentDevice` (int deviceType)
- int `device_close` ()
- void `device_getIDGStatusCodeString` (IN int returnCode, OUT char *despcrition)
- int `device_isConnected` ()
- int `device_isAttached` (int deviceType)
- int `device_getFirmwareVersion` (OUT char *firmwareVersion)
- int `device_getFirmwareVersion_Len` (OUT char *firmwareVersion, IN_OUT int *firmwareVersionLen)
- int `device_pingDevice` ()
- int `device_controlUserInterface` (IN BYTE *values)
- int `device_getCurrentDeviceType` ()
- int `device_SendDataCommandNEO` (IN int cmd, IN int subCmd, IN BYTE *data, IN int dataLen, OUT BYTE *response, IN_OUT int *respLen)
- int `device_enablePassThrough` (int enablePassThrough)
- int `device_setBurstMode` (IN BYTE mode)
- int `device_setPollMode` (IN BYTE mode)
- int `device_setMerchantRecord` (int index, int enabled, char *merchantID, char *merchantURL)
- int `device_getTransactionResults` (IDTMSRData *cardData)
- int `device_getMerchantRecord` (IN int index, OUT BYTE *record)
- int `device_getMerchantRecord_Len` (IN int index, OUT BYTE *record, IN_OUT int *recordLen)
- int `device_getSDKWaitTime` ()
- void `device_setSDKWaitTime` (int waitTime)
- int `device_getThreadStackSize` ()
- void `device_setThreadStackSize` (int threadSize)
- int `config_getSerialNumber` (OUT char *sNumber)
- int `config_getSerialNumber_Len` (OUT char *sNumber, IN_OUT int *sNumberLen)
- int `device_startRKI` (IN const char *caPath, IN int isProduction)
- void `device_setRKI_URL` (IN char *rkiURL, IN int rkiURLLen)
- int `ctls_startTransaction` (IN double amount, IN double amtOther, IN int type, IN const int _timeout, IN BYTE *tags, IN int tagsLen)
- int `ctls_activateTransaction` (IN const int _timeout, IN BYTE *tags, IN int tagsLen)
- int `ctls_cancelTransaction` ()
- int `ctls_retrieveApplicationData` (IN BYTE *AID, IN int AIDLen, OUT BYTE *tlv, IN_OUT int *tlvLen)
- int `ctls_setApplicationData` (IN BYTE *tlv, IN int tlvLen)
- int `ctls_removeApplicationData` (IN BYTE *AID, IN int AIDLen)
- int `ctls_removeAllApplicationData` ()
- int `ctls_retrieveAIDList` (OUT BYTE *AIDList, IN_OUT int *AIDListLen)
- int `ctls_retrieveTerminalData` (OUT BYTE *tlv, IN_OUT int *tlvLen)
- int `ctls_setTerminalData` (IN BYTE *tlv, IN int tlvLen)
- int `ctls_retrieveCAPK` (IN BYTE *capk, IN int capkLen, OUT BYTE *key, IN_OUT int *keyLen)
- int `ctls_setCAPK` (IN BYTE *capk, IN int capkLen)
- int `ctls_removeCAPK` (IN BYTE *capk, IN int capkLen)
- int `ctls_removeAllCAPK` ()
- int `ctls_retrieveCAPKList` (OUT BYTE *keys, IN_OUT int *keysLen)
- int `ctls_setConfigurationGroup` (IN BYTE *tlv, IN int tlvLen)
- int `ctls_getConfigurationGroup` (IN int group, OUT BYTE *tlv, OUT int *tlvLen)
- int `ctls_getAllConfigurationGroups` (OUT BYTE *tlv, IN_OUT int *tlvLen)
- int `ctls_removeConfigurationGroup` (int group)
- void `parseMSRData` (IN BYTE *resData, IN int resLen, IN_OUT IDTMSRData *cardData)

11.3.1 Detailed Description

KioskIII API.

KioskIII Global API methods.

11.3.2 Macro Definition Documentation

11.3.2.1 IN

```
#define IN
```

INPUT parameter.

11.3.2.2 IN_OUT

```
#define IN_OUT
```

INPUT / OUTPUT PARAMETER.

11.3.2.3 OUT

```
#define OUT
```

OUTPUT parameter.

11.3.3 Typedef Documentation

11.3.3.1 ftpComm_callBack

```
typedef void(* ftpComm_callBack) (int, int, int)
```

Define the comm callback function to get FTP file transfer status

It should be passed as a parameter in a FTP request, Signature (int, int, int) = response code, current block, total blocks
 RESPONSE CODES: 100 = FILE DOWNLOAD STARTED 101 = FILE BLOCK XX OF XX RECEIVED 102 = FILE DOWNLOAD COMPLETED 103 = FILE DOWNLOAD TERMINATED PREMATURELY

11.3.3.2 httpComm_callBack

```
typedef void(* httpComm_callBack) (BYTE *, int)
```

Define the comm callback function to get the async url data

It should be registered using the comm_registerHTTPCallback

11.3.3.3 pCMR_callBack

```
typedef void(* pCMR_callBack) (int, IDTCMRData *)
```

Define the camera callback function to get the image data

It should be registered using the device_registerCameraCallBk,

11.3.3.4 pCSFS_callBack

```
typedef void(* pCSFS_callBack) (BYTE status)
```

Define the card status and front switch callback function to get card and front switch status
It should be registered using the device_registerCardStatusFrontSwitchCallBk,

11.3.3.5 pEMV_callBack

```
typedef void(* pEMV_callBack) (int, int, BYTE *, int, IDTTransactionData *, EMV_Callback *,  
int)
```

Define the EMV callback function to get the transaction message/data/result.
It should be registered using the emv_registerCallBk,

11.3.3.6 pMessageHotplug

```
typedef void(* pMessageHotplug) (int, int)
```

Define the USB hot-plug callback function to monitor the info when plug in/out the reader.
It should be registered using the registerHotplugCallBk, The first integer parameter is device type, and the second integer parameter is either 0: Device Plugged Out or 1: Device Plugged In

11.3.3.7 pMSR_callBack

```
typedef void(* pMSR_callBack) (int, IDTMSRData)
```

Define the MSR callback function to get the MSR card data
It should be registered using the msr_registerCallBk, this callback function is for backward compatibility

11.3.3.8 pMSR_callBackp

```
typedef void(* pMSR_callBackp) (int, IDTMSRData *)
```

Define the MSR callback function to get pointer to the MSR card data
It should be registered using the msr_registerCallBk, this callback function is recommended instead of pMSR_callBack

11.3.3.9 pPIN_callBack

```
typedef void(* pPIN_callBack) (int, IDTPINData *)
```

Define the PINPad callback function to get the input PIN Pad data
It should be registered using the pin_registerCallBk,

11.3.3.10 pReadDataLog

```
typedef void(* pReadDataLog) (BYTE *, int)
```

Define the read response callback function to monitor the reading response from the reader.
It should be registered using the registerLogCallBk,

11.3.3.11 pSendDataLog

```
typedef void(* pSendDataLog) (BYTE *, int)
```


Define the send command callback function to monitor the sending command into the reader.
It should be registered using the registerLogCallBk,

11.3.3.12 v4Comm_callBack

```
typedef void(* v4Comm_callBack) (BYTE, BYTE, BYTE *, int)
```

Define the comm callback function to receive the V4 Protocol packets received by the device from an external source (IP/USB/RS-232) It should be registered using the comm_registerV4Callback, Data callback will contain command, sub-command, and data from V4 packet

11.3.4 Function Documentation

11.3.4.1 config_getSerialNumber()

```
int config_getSerialNumber (
    OUT char * sNumber )
```

DEPRECATED : please use [config_getSerialNumber_Len\(OUT char* sNumber, IN_OUT int *sNumberLen\)](#)

Polls device for Serial Number

Parameters

<i>sNumber</i>	Returns Serial Number; needs to have at least 64 bytes of memory
----------------	--

Returns

RETURN_CODE: Values can be parsed with device_getIDGStatusCodeString

11.3.4.2 config_getSerialNumber_Len()

```
int config_getSerialNumber_Len (
    OUT char * sNumber,
    IN_OUT int * sNumberLen )
```

Polls device for Serial Number

Parameters

<i>sNumber</i>	Returns Serial Number
<i>sNumberLen</i>	Length of Serial Number

Returns

RETURN_CODE: Values can be parsed with device_getIDGStatusCodeString

11.3.4.3 ctls_activateTransaction()

```
int ctls_activateTransaction (
    IN const int _timeout,
    IN BYTE * tags,
    IN int tagsLen )
```

Start CTLS Transaction Request

Authorizes the CTLS 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 stream. Example, tag 9F0C with amount 0x0000000000100 would be 0x9F0C06000000000100
<i>tagsLen</i>	The length of tags data buffer.

>>>>IMPORTANT: parameters for amount and amtOther MUST BE PASSED AS A DOUBLE VALUE WITH DECIMAL POINT. Example, do not pass 1, but instead pass 1.0 or 1.00. Otherwise, results will be unpredictable

Returns

RETURN_CODE: Values can be parsed with [device_getIDGStatusCodeString\(\)](#) Note: if auto poll is on, it will return the error IDG_P2_STATUS_CODE_COMMAND_NOT_ALLOWED

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

11.3.4.4 ctls_cancelTransaction()

```
int ctls_cancelTransaction ( )
```

Cancel EMV Transaction

Cancels the currently executing EMV transaction.

Returns

RETURN_CODE: Values can be parsed with [device_getIDGStatusCodeString\(\)](#)

11.3.4.5 ctls_getAllConfigurationGroups()

```
int ctls_getAllConfigurationGroups (
    OUT BYTE * tlv,
    IN_OUT int * tlvLen )
```

Retrieve All Configuration Groups

Returns all the Configuration Groups installed on the terminal for CTLS

Parameters

<i>tlv</i>	The TLV elements data
<i>tlvLen</i>	the length of tlv data buffer.

Returns

RETURN_CODE: Values can be parsed with [device_getIDGStatusCodeString\(\)](#)

11.3.4.6 ctls_getConfigurationGroup()

```
int ctls_getConfigurationGroup (
    IN int group,
    OUT BYTE * tlv,
    OUT int * tlvLen )
```

Get Configuration Group

Retrieves the Configuration for the specified Group.

Parameters

<i>group</i>	Configuration Group
<i>tlv</i>	return data
<i>tlvLen</i>	the length of tlv data buffer

Returns

RETURN_CODE: Values can be parsed with [device_getIDGStatusCodeString\(\)](#)

11.3.4.7 ctls_registerCallBk()

```
void ctls_registerCallBk (
    pMSR_callBack pCTLSf )
```

To register the ctls callback function to get the MSR card data. (Pass NULL to disable the callback.)

11.3.4.8 ctls_registerCallBkp()

```
void ctls_registerCallBkp (
    pMSR_callBackp pCTLSf )
```

To register the ctls callback function to get the MSR card data pointer. (Pass NULL to disable the callback.)

11.3.4.9 ctls_removeAllApplicationData()

```
int ctls_removeAllApplicationData ( )
```

Remove All Application Data

Removes all the Application Data

Returns

RETURN_CODE: Values can be parsed with [device_getIDGStatusCodeString\(\)](#)

11.3.4.10 ctls_removeAllCAPK()

```
int ctls_removeAllCAPK ( )
```

Remove All Certificate Authority Public Key

Removes all the CAPK

Returns

RETURN_CODE: Values can be parsed with [device_getIDGStatusCodeString\(\)](#)

11.3.4.11 ctls_removeApplicationData()

```
int ctls_removeApplicationData (
    IN BYTE * AID,
    IN int AIDLen )
```

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>AIDLen</i>	the length of AID data buffer

Returns

RETURN_CODE: Values can be parsed with [device_getIDGStatusCodeString\(\)](#)

11.3.4.12 ctls_removeCAPK()

```
int ctls_removeCAPK (
    IN BYTE * capk,
    IN int capkLen )
```

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>capkLen</i>	the length of capk data buffer

Returns

RETURN_CODE: Values can be parsed with [device_getIDGStatusCodeString\(\)](#)

11.3.4.13 ctls_removeConfigurationGroup()

```
int ctls_removeConfigurationGroup (
    int group )
```

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>	Values can be parsed with device_getIDGStatusCodeString()
--------------------	---

11.3.4.14 `ctls_retrieveAIDList()`

```
int ctls_retrieveAIDList (
    OUT BYTE * AIDList,
    IN_OUT int * AIDListLen )
```

Retrieve AID list

Returns all the AID names installed on the terminal for CTLS. .

Parameters

<i>AIDList</i>	array of AID name byte arrays
<i>AIDListLen</i>	the length of AIDList array buffer

Returns

RETURN_CODE: Values can be parsed with [device_getIDGStatusCodeString\(\)](#)

11.3.4.15 `ctls_retrieveApplicationData()`

```
int ctls_retrieveApplicationData (
    IN BYTE * AID,
    IN int AIDLen,
    OUT BYTE * tlv,
    IN_OUT int * tlvLen )
```

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>AIDLen</i>	the length of AID data buffer.
<i>tlv</i>	The TLV elements of the requested AID
<i>tlvLen</i>	the length of tlv data buffer.

Returns

RETURN_CODE: Values can be parsed with [device_getIDGStatusCodeString\(\)](#)

11.3.4.16 `ctls_retrieveCAPK()`

```
int ctls_retrieveCAPK (
```

```

    IN BYTE * capk,
    IN int capkLen,
    OUT BYTE * key,
    IN_OUT int * keyLen )

```

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>capkLen</i>	the length of capk data buffer
<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>keyLen</i>	<p>the length of key data buffer</p> <ul style="list-style-type: none"> •

Returns

RETURN_CODE: Values can be parsed with [device_getIDGStatusCodeString\(\)](#)

11.3.4.17 ctls_retrieveCAPKList()

```

int ctls_retrieveCAPKList (
    OUT BYTE * keys,
    IN_OUT int * keysLen )

```

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>keysLen</i>	the length of keys data buffer

Returns

RETURN_CODE: Values can be parsed with [device_getIDGStatusCodeString\(\)](#)

11.3.4.18 ctls_retrieveTerminalData()

```
int ctls_retrieveTerminalData (
    OUT BYTE * tlv,
    IN_OUT int * tlvLen )
```

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>tlvLen</i>	the length of tlv data buffer

Returns

RETURN_CODE: Values can be parsed with [device_getIDGStatusCodeString\(\)](#)

11.3.4.19 ctls_setApplicationData()

```
int ctls_setApplicationData (
    IN BYTE * tlv,
    IN int tlvLen )
```

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>tlvLen</i>	the length of tlv data buffer
---------------	-------------------------------

Returns

RETURN_CODE: Values can be parsed with [device_getIDGStatusCodeString\(\)](#)

11.3.4.20 ctls_setCAPK()

```
int ctls_setCAPK (
    IN BYTE * capk,
    IN int capkLen )
```

Set Certificate Authority Public Key

Sets the CAPK as specified by the CAKey structure

Parameters

<i>capk</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>capkLen</i>	the length of capk data buffer

Returns

RETURN_CODE: Values can be parsed with [device_getIDGStatusCodeString\(\)](#)

11.3.4.21 ctls_setConfigurationGroup()

```
int ctls_setConfigurationGroup (
    IN BYTE * tlv,
    IN int tlvLen )
```

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>tlvLen</i>	the length of tlv data buffer

Returns

RETURN_CODE: Values can be parsed with [device_getIDGStatusCodeString\(\)](#)

11.3.4.22 ctls_setTerminalData()

```
int ctls_setTerminalData (
    IN BYTE * tlv,
    IN int tlvLen )
```

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 [ctls_getConfigurationGroup\(int group\)](#), and deleted with [ctls_removeConfigurationGroup\(int group\)](#). You cannot delete group 0.

Parameters

<i>tlv</i>	TerminalData configuration file
<i>tlvLen</i>	the length of tlv data buffer

Return values

<i>RETURN_CODE</i>	Values can be parsed with device_getIDGStatusCodeString()
--------------------	---

11.3.4.23 ctls_startTransaction()

```
int ctls_startTransaction (
    IN double amount,
    IN double amtOther,
    IN int type,
    IN const int _timeout,
    IN BYTE * tags,
    IN int tagsLen )
```

Start CTLS Transaction Request

Authorizes the CTLS transaction for an ICC card

The tags will be returned in the callback routine.

Parameters

<i>amount</i>	Transaction amount value (tag value 9F02) - SEE IMPORTANT NOTE BELOW
<i>amtOther</i>	Other amount value, if any (tag value 9F03) - SEE IMPORTANT NOTE BELOW
<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 stream. Example, tag 9F0C with amount 0x000000000100 would be 0x9F0C06000000000100 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>tagsLen</i>	The length of tags data buffer.

>>>>>IMPORTANT: parameters for amount and amtOther MUST BE PASSED AS A DOUBLE VALUE WITH DECIMAL POINT. Example, do not pass 1, but instead pass 1.0 or 1.00. Otherwise, results will be unpredictable

Returns

RETURN_CODE: Values can be parsed with [device_getIDGStatusCodeString\(\)](#) Note: if auto poll is on, it will return the error IDG_P2_STATUS_CODE_COMMAND_NOT_ALLOWED

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) DFO1 = 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

11.3.4.24 device_close()

```
int device_close ( )
```

Close the device

Returns

RETURN_CODE: 0: success, 0x0A: failed

11.3.4.25 device_controlUserInterface()

```
int device_controlUserInterface (
    IN BYTE * values )
```

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) • 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
---------------	--

Returns

RETURN_CODE: Values can be parsed with [device_getIDGStatusCodeString\(\)](#)

11.3.4.26 device_enablePassThrough()

```
int device_enablePassThrough (
    int enablePassThrough )
```

Enable Pass Through - NEO

Enables Pass Through Mode for direct communication with L1 interface (power on icc, send apdu, etc).

Parameters

<i>enablePassThrough</i>	1 = pass through ON, 0 = pass through OFF
--------------------------	---

Returns

RETURN_CODE: Values can be parsed with [device_getIDGStatusCodeString\(\)](#)

11.3.4.27 device_getCurrentDeviceType()

```
int device_getCurrentDeviceType ( )
```

Get current active device type

Returns

: return the device type defined as DEVICE_TYPE in the IDTDef.h

11.3.4.28 device_getFirmwareVersion()

```
int device_getFirmwareVersion (
    OUT char * firmwareVersion )
```

DEPRECATED : please use [device_getFirmwareVersion_Len\(\)](#)(OUT char* firmwareVersion, IN_OUT int *firmwareVersionLen)

Polls device for Firmware Version

Parameters

<i>firmwareVersion</i>	Response returned of Firmware Version; needs to have at least 128 bytes of memory
------------------------	---

Returns

RETURN_CODE: Values can be parsed with [device_getIDGStatusCodeString\(\)](#)

11.3.4.29 device_getFirmwareVersion_Len()

```
int device_getFirmwareVersion_Len (
    OUT char * firmwareVersion,
    IN_OUT int * firmwareVersionLen )
```

Polls device for Firmware Version

Parameters

<i>firmwareVersion</i>	Response returned of Firmware Version
<i>firmwareVersionLen</i>	Length of Firmware Version

Returns

RETURN_CODE: Values can be parsed with [device_getIDGStatusCodeString\(\)](#)

11.3.4.30 device_getIDGStatusCodeString()

```
void device_getIDGStatusCodeString (
    IN int  returnCode,
    OUT char * despcriton )
```

Review the return code description.

Parameters

<i>returnCode</i>	the response result.
<i>description</i>	

Return values

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

- 0: "no error, beginning task";
- 1: "no response from reader";
- 2: "invalid response data";
- 01: " Incorrect Header Tag";
- 02: " Unknown Command";
- 03: " Unknown Sub-Command";
- 04: " CRC Error in Frame";
- 05: " Incorrect Parameter";
- 06: " Parameter Not Supported";
- 07: " Mal-formatted Data";
- 08: " Timeout";
- 0A: " Failed / NACK";
- 0B: " Command not Allowed";
- 0C: " Sub-Command not Allowed";
- 0D: " Buffer Overflow (Data Length too large for reader buffer)";

- 0E: " User Interface Event";
- 10: " Need clear firmware(apply in boot loader only)";
- 11: " Communication type not supported, VT-1, burst, etc. Need encrypted firmware (apply in boot loader only)";
- 12: " Secure interface is not functional or is in an intermediate state.";
- 13: " Data field is not mod 8";
- 14: " Pad 0x80 not found where expected";
- 15: " Specified key type is invalid";
- 16: " Could not retrieve key from the SAM (InitSecureComm)";
- 17: " Hash code problem";
- 18: " Could not store the key into the SAM (InstallKey)";
- 19: " Frame is too large";
- 1A: " Unit powered up in authentication state but POS must resend the InitSecureComm command";
- 1B: " The EEPROM may not be initialized because SecCommInterface does not make sense";
- 1C: " Problem encoding APDU Module-Specific Status Codes ";
- 20: " Unsupported Index (ILM) SAM Transceiver error - problem communicating with the SAM (Key Mgr)";
- 21: " Unexpected Sequence Counter in multiple frames for single bitmap (ILM)Length error in data returned from the SAM (Key Mgr) - 22: " Improper bit map (ILM)"; - 23: " Request Online Authorization"; - 24: " ViV↔ OCard3 raw data read successful"; - 25: " Message index not available (ILM) ViVOcomm activate transaction card type (ViVOcomm)"; - 26: " Version Information Mismatch (ILM)"; - 27: " Not sending commands in correct index message index (ILM)"; - 28: " Time out or next expected message not received (ILM)"; - 29: " ILM languages not available for viewing (ILM)"; - 2A: " Other language not supported (ILM)"; - 41: " from 41 to 4F, Module-specific errors for Key Manager";
- 50: " Auto-Switch OK";
- 51: " Auto-Switch failed";
- 70: " Antenna Error 80h Use another card";
- 81: " Insert or swipe card";
- 90: " Data encryption Key does not exist";
- 91: " Data encryption Key KSN exhausted";

11.3.4.31 device_getMerchantRecord()

```
int device_getMerchantRecord (
    IN int index,
    OUT BYTE * record )
```

DEPRECATED : please use device_getMerchantRecord_Len(IN int index, OUT BYTE * record, IN_OUT int *recordLen)

Get Merchant Record

Gets the merchant record for the device.

Parameters

<i>index</i>	Merchant Record index, valid values 1-6
<i>record;</i>	needs to have at least 99 bytes of memory response data from reader. Merchant Record Index: 1 byte enabled: 1 byte Merchant ID: 32 bytes Length of Merchant URL: 1 byte Merchant URL: 64 bytes

Returns

success or error code. Values can be parsed with [device_getIDGStatusCodeString\(\)](#)

See also

ErrorCode

11.3.4.32 device_getMerchantRecord_Len()

```
int device_getMerchantRecord_Len (
    IN int index,
    OUT BYTE * record,
    IN_OUT int * recordLen )
```

Get Merchant Record

Gets the merchant record for the device.

Parameters

<i>index</i>	Merchant Record index, valid values 1-6
<i>record</i>	response data from reader. Merchant Record Index: 1 byte enabled: 1 byte Merchant ID: 32 bytes Length of Merchant URL: 1 byte Merchant URL: 64 bytes
<i>recordLen</i>	Length of record

Returns

success or error code. Values can be parsed with [device_getIDGStatusCodeString\(\)](#)

See also

ErrorCode

11.3.4.33 device_getSDKWaitTime()

```
int device_getSDKWaitTime ( )
```

Get SDK Wait Time

Get the SDK wait time for transactions

Returns

SDK wait time in seconds

11.3.4.34 `device_getThreadStackSize()`

```
int device_getThreadStackSize ( )
```

Get Thread Stack Size

Get the stack size setting for newly created threads

Returns

Thread Stack Size

11.3.4.35 `device_getTransactionResults()`

```
int device_getTransactionResults (
    IDTMSRData * cardData )
```

Get Transaction Results Gets the transaction results when the reader is functioning in "Auto Poll" mode

Parameters

<i>cardData</i>	The transaction results
-----------------	-------------------------

Returns

success or error code. Values can be parsed with `device_getResponseCodeString`

See also

`ErrorCode`

11.3.4.36 `device_init()`

```
int device_init ( )
```

Initial the device by USB

It will detect the device and trying connect.

The connect status can be checked by [device_isConnected\(\)](#).

Note: after the function returns success, the function [device_setCurrentDevice\(\)](#) has to be called to set the device type.

Returns

RETURN_CODE: Values can be parsed with [device_getIDGStatusCodeString\(\)](#)

11.3.4.37 `device_isAttached()`

```
int device_isAttached (
    int deviceType )
```

Check if the device is attached to the USB port The function [device_init\(\)](#) must be called before this function.

Parameters

<i>deviceType,the</i>	device type of the USB device
-----------------------	-------------------------------

Returns

1 if the device is attached, or 0 if the device is not attached

11.3.4.38 device_isConnected()

```
int device_isConnected ( )
```

Check the device connected status

Returns

DEVICE_DISCONNECT=0, or DEVICE_CONNECTED = 1

11.3.4.39 device_pingDevice()

```
int device_pingDevice ( )
```

Ping Device

Pings the reader. If connected, returns success. Otherwise, returns timeout.

Returns

RETURN_CODE: Values can be parsed with [device_getIDGStatusCodeString\(\)](#)

11.3.4.40 device_registerCameraCallBk()

```
void device_registerCameraCallBk (
    pCMR_callback pCMRf )
```

To register the camera callback function to get the image data. (Pass NULL to disable the callback.)

11.3.4.41 device_registerCardStatusFrontSwitchCallBk()

```
void device_registerCardStatusFrontSwitchCallBk (
    pCSFS_callback pCSFSf )
```

To register the card status and front switch callback function to get status. (Pass NULL to disable the callback.)

11.3.4.42 device_SendDataCommandNEO()

```
int device_SendDataCommandNEO (
    IN int cmd,
    IN int subCmd,
    IN BYTE * data,
    IN int dataLen,
```

```
OUT BYTE * response,  
IN_OUT int * respLen )
```

Send a Command to NEO device

Sends a command to the NEO device .

Parameters

<i>cmd</i>	command to execute.
<i>subCmd,sub</i>	command to execute.
<i>data</i>	buffer of NEO command data.
<i>dataLen,the</i>	length of the buffer data.
<i>response</i>	Response data
<i>respLen,the</i>	length of Response data

Returns

RETURN_CODE: Values can be parsed with [device_getIDGStatusCodeString\(\)](#)

11.3.4.43 device_setBurstMode()

```
int device_setBurstMode (  
    IN BYTE mode )
```

Send Burst Mode - NEO

Sets the burst mode for the device.

Parameters

<i>mode</i>	0 = OFF, 1 = Always On, 2 = Auto Exit
-------------	---------------------------------------

Returns

success or error code. Values can be parsed with [device_getIDGStatusCodeString](#)

See also

[ErrorCode](#)

11.3.4.44 device_setCurrentDevice()

```
int device_setCurrentDevice (  
    int deviceType )
```

Sets the current device to talk to

The connect status can be checked by [device_isConnected\(\)](#).

Parameters

<i>deviceType</i>	Device to connect to <pre>enum DEVICE_TYPE { IDT_DEVICE_UNKNOWN=0, IDT_DEVICE_AUGUSTA_HID, IDT_DEVICE_AUGUSTA_KB, IDT_DEVICE_AUGUSTA_S_HID, IDT_DEVICE_AUGUSTA_S_KB, IDT_DEVICE_AUGUSTA_S_TTK_HID, IDT_DEVICE_SPECTRUM_PRO, IDT_DEVICE_MINISMART_II, IDT_DEVICE_L80, IDT_DEVICE_L100, IDT_DEVICE_UNIPAY, IDT_DEVICE_UNIPAY_I_V, IDT_DEVICE_VP3300_AJ, IDT_DEVICE_KIOSK_III, IDT_DEVICE_KIOSK_III_S, IDT_DEVICE_PIP_READER, IDT_DEVICE_VENDI, IDT_DEVICE_VP3300_USB, IDT_DEVICE_UNIPAY_I_V_TTK, IDT_DEVICE_VP3300_BT, IDT_DEVICE_VP8800, IDT_DEVICE_SREDKEY2_HID, IDT_DEVICE_SREDKEY2_KB, IDT_DEVICE_NEO2, IDT_DEVICE_MINISMART_II_COM = IDT_DEVICE_NEO2+5, IDT_DEVICE_SPECTRUM_PRO_COM, IDT_DEVICE_KIOSK_III_COM, IDT_DEVICE_KIOSK_III_S_COM, IDT_DEVICE_PIP_READER_COM, IDT_DEVICE_VP3300_COM, IDT_DEVICE_NEO2_COM, IDT_DEVICE_MAX_DEVICES = IDT_DEVICE_NEO2_COM+5 };</pre>
-------------------	---

Returns

RETURN_CODE: 1: success, 0: failed

11.3.4.45 device_setMerchantRecord()

```
int device_setMerchantRecord (
    int index,
    int enabled,
    char * merchantID,
    char * merchantURL )
```

Set Merchant Record - NEO 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

Returns

RETURN_CODE: Values can be parsed with [device_getIDGStatusCodeString\(\)](#)

11.3.4.46 device_setPollMode()

```
int device_setPollMode (
    IN BYTE mode )
```

Set Poll Mode - NEO

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
-------------	-----------------------------------

Returns

RETURN_CODE: Values can be parsed with [device_getIDGStatusCodeString\(\)](#)

11.3.4.47 device_setRKI_URL()

```
void device_setRKI_URL (
    IN char * rkiURL,
    IN int rkiURLLen )
```

Set the URL for RKI

Parameters

<i>rkiURL</i>	The URL for RKI (less than 100 characters). Pass NULL to reset to default URL
<i>rkiURLLen</i>	The length of rkiURL. Pass 0 to reset to default URL

Returns**11.3.4.48 device_setSDKWaitTime()**

```
void device_setSDKWaitTime (
    int waitTime )
```

Set SDK Wait Time

Set the SDK wait time for transactions

Parameters

<i>waitTime</i>	The SDK wait time for transaction in seconds. The Maximum is 2147483 seconds.
-----------------	---

11.3.4.49 device_setThreadStackSize()

```
void device_setThreadStackSize (
    int threadSize )
```

Set Thread Stack Size

Set the stack size setting for newly created threads

11.3.4.50 device_startRKI()

```
int device_startRKI (
    IN const char * caPath,
    IN int isProduction )
```

Start remote key injection.

Parameters

<i>caPath</i>	The path to ca-certificates.crt. It should be NULL, because the file is not used anymore.
<i>isProduction</i>	1: The reader is a production unit, 0: The reader is not a production unit

Returns

success or error code.

See also

ErrorCode

11.3.4.51 emv_registerCallBk()

```
void emv_registerCallBk (
    pEMV_callback pEMVf )
```

To register the emv callback function to get the EMV processing response. (Pass NULL to disable the callback.)

11.3.4.52 parseMSRData()

```
void parseMSRData (
    IN BYTE * resData,
    IN int resLen,
    IN_OUT IDTMSRData * cardData )
```

Parser the MSR data from the buffer into IDTMSTData structure

Parameters

<i>resData</i>	MSR card data buffer
<i>resLen</i>	the length of resData
<i>cardData</i>	the parser result with IDTMSTData structure

11.3.4.53 pin_registerCallBk()

```
void pin_registerCallBk (
    pPIN_callback pPINf )
```

To register the pin callback function to get the PINPad data. (Pass NULL to disable the callback.)

11.3.4.54 registerHotplugCallBk()

```
void registerHotplugCallBk (
    pMessageHotplug pMsgHotplug )
```

To register the USB HID hot-plug callback function which implemented in the application to monitor the hotplug message from the SDK.

11.3.4.55 registerLogCallBk()

```
void registerLogCallBk (
    pSendDataLog pFSend,
    pReadDataLog pFRead )
```

To register the log callback function which implemented in the application to monitor sending/reading data between application and reader.

11.3.4.56 rs232_device_init()

```
int rs232_device_init (
    int deviceType,
    int port_number,
    int brate )
```

Initial the device by RS232

It will try to connect to the device with provided deviceType, port_number, and brate.

Parameters

<i>deviceType</i>	Device to connect to
<i>port_number</i>	Port number of the device

Port nr. | Linux | Windows

0	ttyS0	COM1	1	ttyS1	COM2	2	ttyS2	COM3	3	ttyS3	COM4	4	ttyS4	COM5	5	ttyS5	COM6	6	ttyS6	COM7	7	ttyS7	COM8	8	ttyS8	COM9	9	ttyS9	COM10	10	ttyS10	COM11	11	ttyS11	COM12	12	ttyS12	COM13	13	ttyS13	COM14	14	ttyS14	COM15	15	ttyS15	COM16	16	ttyUSB0	n.a.	17	ttyUSB1	n.a.	18	ttyUSB2	n.a.	19	ttyUSB3	n.a.	20	ttyUSB4	n.a.	21	ttyUSB5	n.a.	22	ttyAMA0	n.a.	23	ttyAMA1	n.a.	24	ttyACM0	n.a.	25	ttyACM1	n.a.	26	rfcomm0	n.a.	27	rfcomm1	n.a.	28	ircomm0	n.a.	29	ircomm1	n.a.	30	cuau0	n.a.	31	cuau1	n.a.	32	cuau2	n.a.	33	cuau3	n.a.	34	cuaU0	n.a.	35	cuaU1	n.a.	36	cuaU2	n.a.	37	cuaU3	n.a.
---	-------	------	---	-------	------	---	-------	------	---	-------	------	---	-------	------	---	-------	------	---	-------	------	---	-------	------	---	-------	------	---	-------	-------	----	--------	-------	----	--------	-------	----	--------	-------	----	--------	-------	----	--------	-------	----	--------	-------	----	---------	------	----	---------	------	----	---------	------	----	---------	------	----	---------	------	----	---------	------	----	---------	------	----	---------	------	----	---------	------	----	---------	------	----	---------	------	----	---------	------	----	---------	------	----	---------	------	----	-------	------	----	-------	------	----	-------	------	----	-------	------	----	-------	------	----	-------	------	----	-------	------	----	-------	------

Parameters

<i>brate</i>	Bitrate of the device
--------------	-----------------------

Returns

RETURN_CODE: Values can be parsed with [device_getIDGStatusCodeString\(\)](#)

11.3.4.57 SDK_Version()

```
char* SDK_Version ( )
```

To Get SDK version

Returns

return the SDK version string

11.3.4.58 `setAbsoluteLibraryPath()`

```
int setAbsoluteLibraryPath (
    const char * absoluteLibraryPath )
```

Set the path to use when searching for ID TECH's libraries. If this is not set, the libraries will be searched for with the system's default procedures.

Parameters

<i>absoluteLibraryPath</i>	The absolute path to ID TECH's libraries.
----------------------------	---

Returns

RETURN_CODE: Values can be parsed with [device_getIDGStatusCodeString\(\)](#)

11.4 Source_C/libIDT_L100.h File Reference

L100 API.

```
#include "IDTDef.h"
```

Macros

- `#define IN`
- `#define OUT`
- `#define IN_OUT`

Typedefs

- `typedef void(* pMessageHotplug) (int, int)`
- `typedef void(* pSendDataLog) (BYTE *, int)`
- `typedef void(* pReadDataLog) (BYTE *, int)`
- `typedef void(* pEMV_callBack) (int, int, BYTE *, int, IDTTransactionData *, EMV_Callback *, int)`
- `typedef void(* pMSR_callBack) (int, IDTMSRData)`
- `typedef void(* pMSR_callBackp) (int, IDTMSRData *)`
- `typedef void(* pPIN_callBack) (int, IDTPINData *)`
- `typedef void(* pCMR_callBack) (int, IDTCMRData *)`
- `typedef void(* pCSFS_callBack) (BYTE status)`
- `typedef void(* pFW_callBack) (int, int, int, int, int)`
- `typedef void(* ftpComm_callBack) (int, int, int)`
- `typedef void(* httpComm_callBack) (BYTE *, int)`
- `typedef void(* v4Comm_callBack) (BYTE, BYTE, BYTE *, int)`

Functions

- void [registerHotplugCallBk](#) ([pMessageHotplug](#) pMsgHotplug)
- void [registerLogCallBk](#) ([pSendDataLog](#) pFSend, [pReadDataLog](#) pFRead)
- void [emv_registerCallBk](#) ([pEMV_callBack](#) pEMVf)
- void [msr_registerCallBk](#) ([pMSR_callBack](#) pMSRf)
- void [msr_registerCallBkp](#) ([pMSR_callBackp](#) pMSRf)
- void [pin_registerCallBk](#) ([pPIN_callBack](#) pPINf)

- void `device_registerCameraCallBk` (`pCMR_callback` `pCMRf`)
- void `device_registerCardStatusFrontSwitchCallBk` (`pCSFS_callback` `pCSFSf`)
- void `device_registerFWCallBk` (`pFW_callback` `pFWf`)
- char * `SDK_Version` ()
- int `setAbsoluteLibraryPath` (const char *absoluteLibraryPath)
- int `device_init` ()
- int `device_setCurrentDevice` (int deviceType)
- int `device_close` ()
- void `device_getResponseCodeString` (IN int returnCode, OUT char *despcrition)
- int `device_isConnected` ()
- int `device_isAttached` (int deviceType)
- int `device_getFirmwareVersion` (OUT char *firmwareVersion)
- int `device_getFirmwareVersion_Len` (OUT char *firmwareVersion, IN_OUT int *firmwareVersionLen)
- int `device_getDateTime` (OUT BYTE *dateTime)
- int `device_getDateTime_Len` (OUT BYTE *dateTime, IN_OUT int *dateTimeLen)
- int `device_getCurrentDeviceType` ()
- int `device_SendDataCommand` (IN BYTE *cmd, IN int cmdLen, IN BYTE *data, IN int dataLen, OUT BYTE *response, IN_OUT int *respLen)
- int `device_updateFirmware` (IN BYTE *firmwareData, IN int firmwareDataLen, IN char *firmwareName, IN int encryptionType, IN BYTE *keyBlob, IN int keyBlobLen)
- int `device_rebootDevice` ()
- int `device_getKeyStatus` (int *newFormat, BYTE *status, int *statusLen)
- int `device_enterStopMode` ()
- int `device_setSleepModeTime` (int time)
- int `config_getModelNumber` (OUT char *sNumber)
- int `config_getModelNumber_Len` (OUT char *sNumber, IN_OUT int *sNumberLen)
- int `config_getSerialNumber` (OUT char *sNumber)
- int `config_getSerialNumber_Len` (OUT char *sNumber, IN_OUT int *sNumberLen)
- int `pin_getEncryptedPIN` (int keyType, char *PAN, int PANLen, char *message, int messageLen, int timeout)
- int `pin_promptForKeyInput` (int messageID, int languageID, int maskInput, int minLen, int maxLen, int timeout)
- int `pin_promptForAmountInput` (int messageID, int languageID, int minLen, int maxLen, int timeout)
- int `pin_getFunctionKey` (int timeout)
- int `pin_sendBeep` (int frequency, int duration)
- int `pin_setKeyValues` (int mode)
- int `lcd_savePrompt` (int promptNumber, char *prompt, int promptLen)
- int `lcd_displayPrompt` (int promptNumber, int lineNumber)
- int `lcd_displayMessage` (int lineNumber, char *message, int messageLen)
- int `lcd_enableBacklight` (int enable)
- int `lcd_getBacklightStatus` (int *enabled)

11.4.1 Detailed Description

L100 API.

L100 Global API methods.

11.4.2 Macro Definition Documentation

11.4.2.1 IN

```
#define IN
```

INPUT parameter.

11.4.2.2 IN_OUT

```
#define IN_OUT
```

INPUT / OUTPUT PARAMETER.

11.4.2.3 OUT

```
#define OUT
```

OUTPUT parameter.

11.4.3 Typedef Documentation

11.4.3.1 ftpComm_callBack

```
typedef void(* ftpComm_callBack) (int, int, int)
```

Define the comm callback function to get FTP file transfer status

It should be passed as a parameter in a FTP request, Signature (int, int, int) = response code, current block, total blocks
 RESPONSE CODES: 100 = FILE DOWNLOAD STARTED 101 = FILE BLOCK XX OF XX RECEIVED 102 = FILE DOWNLOAD COMPLETED 103 = FILE DOWNLOAD TERMINATED PREMATURELY

11.4.3.2 httpComm_callBack

```
typedef void(* httpComm_callBack) (BYTE *, int)
```

Define the comm callback function to get the async url data

It should be registered using the comm_registerHTTPCallback

11.4.3.3 pCMR_callBack

```
typedef void(* pCMR_callBack) (int, IDTCMRData *)
```

Define the camera callback function to get the image data

It should be registered using the device_registerCameraCallBk,

11.4.3.4 pCSFS_callBack

```
typedef void(* pCSFS_callBack) (BYTE status)
```

Define the card status and front switch callback function to get card and front switch status

It should be registered using the device_registerCardStatusFrontSwitchCallBk,

11.4.3.5 pEMV_callBack

```
typedef void(* pEMV_callBack) (int, int, BYTE *, int, IDTTransactionData *, EMV_Callback *, int)
```

Define the EMV callback function to get the transaction message/data/result.

It should be registered using the emv_registerCallBk,

11.4.3.6 pFW_callBack

```
typedef void(* pFW_callBack) (int, int, int, int, int)
```

Define the firmware update callback function to get the status of firmware update
It should be registered using the device_registerFWCallBk,

11.4.3.7 pMessageHotplug

```
typedef void(* pMessageHotplug) (int, int)
```

Define the USB hot-plug callback function to monitor the info when plug in/out the reader.
It should be registered using the registerHotplugCallBk, The first integer parameter is device type, and the second integer parameter is either 0: Device Plugged Out or 1: Device Plugged In

11.4.3.8 pMSR_callBack

```
typedef void(* pMSR_callBack) (int, IDTMSRData)
```

Define the MSR callback function to get the MSR card data
It should be registered using the msr_registerCallBk, this callback function is for backward compatibility

11.4.3.9 pMSR_callBackp

```
typedef void(* pMSR_callBackp) (int, IDTMSRData *)
```

Define the MSR callback function to get pointer to the MSR card data
It should be registered using the msr_registerCallBk, this callback function is recommended instead of pMSR_callBack

11.4.3.10 pPIN_callBack

```
typedef void(* pPIN_callBack) (int, IDTPINData *)
```

Define the PINPad callback function to get the input PIN Pad data
It should be registered using the pin_registerCallBk,

11.4.3.11 pReadDataLog

```
typedef void(* pReadDataLog) (BYTE *, int)
```

Define the read response callback function to monitor the reading response from the reader.
It should be registered using the registerLogCallBk,

11.4.3.12 pSendDataLog

```
typedef void(* pSendDataLog) (BYTE *, int)
```

Define the send command callback function to monitor the sending command into the reader.
It should be registered using the registerLogCallBk,

11.4.3.13 v4Comm_callBack

```
typedef void(* v4Comm_callBack) (BYTE, BYTE, BYTE *, int)
```

Define the comm callback function to receive the V4 Protocol packets received by the device from an external source (IP/USB/RS-232) It should be registered using the comm_registerV4Callback, Data callback will contain command, sub-command, and data from V4 packet

11.4.4 Function Documentation

11.4.4.1 config_getModelNumber()

```
int config_getModelNumber (
    OUT char * sNumber )
```

DEPRECATED : please use [config_getModelNumber_Len\(OUT char* sNumber, IN_OUT int *sNumberLen\)](#)

Polls device for Model Number

Parameters

<i>sNumber</i>	Returns Model Number; needs to have at least 64 bytes of memory
----------------	---

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString

11.4.4.2 config_getModelNumber_Len()

```
int config_getModelNumber_Len (
    OUT char * sNumber,
    IN_OUT int * sNumberLen )
```

Polls device for Model Number

Parameters

<i>sNumber</i>	Returns Model Number
<i>sNumber</i>	length of Model Number

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString

11.4.4.3 config_getSerialNumber()

```
int config_getSerialNumber (
    OUT char * sNumber )
```

DEPRECATED : please use [config_getSerialNumber_Len\(OUT char* sNumber, IN_OUT int *sNumberLen\)](#)

Polls device for Serial Number

Parameters

<i>sNumber</i>	Returns Serial Number; needs to have at least 64 bytes of memory
----------------	--

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString

11.4.4.4 config_getSerialNumber_Len()

```
int config_getSerialNumber_Len (
    OUT char * sNumber,
    IN_OUT int * sNumberLen )
```

Polls device for Serial Number

Parameters

<i>sNumber</i>	Returns Serial Number
<i>sNumberLen</i>	Length of Serial Number

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString

11.4.4.5 device_close()

```
int device_close ( )
```

Close the device

Returns

RETURN_CODE: 0: success, 0x0A: failed

11.4.4.6 device_enterStopMode()

```
int device_enterStopMode ( )
```

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.

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString

11.4.4.7 device_getCurrentDeviceType()

```
int device_getCurrentDeviceType ( )
```

Get current active device type

Returns

: return the device type defined as DEVICE_TYPE in the IDTDef.h

11.4.4.8 device_getDateTime()

```
int device_getDateTime (
    OUT BYTE * dateTime )
```

Polls device for Date and Time

Parameters

<i>dateTime</i>	Response returned of Date and Time; needs to have at least 6 bytes of memory
-----------------	--

Returns

RETURN_CODE: Values can be parsed with [device_getResponseCodeString\(\)](#)

11.4.4.9 device_getDateTime_Len()

```
int device_getDateTime_Len (
    OUT BYTE * dateTime,
    IN_OUT int * dateTimeLen )
```

Polls device for Date and Time

Parameters

<i>dateTime</i>	Response returned of Date and Time
<i>dateTime</i>	Length of Date and Time

Returns

RETURN_CODE: Values can be parsed with [device_getResponseCodeString\(\)](#)

11.4.4.10 device_getFirmwareVersion()

```
int device_getFirmwareVersion (
    OUT char * firmwareVersion )
```

DEPRECATED : please use [device_getFirmwareVersion_Len](#)(OUT char* firmwareVersion, IN_OUT int *firmwareVersionLen)

Polls device for Firmware Version

Parameters

<i>firmwareVersion</i>	Response returned of Firmware Version; needs to have at least 128 bytes of memory
------------------------	---

Returns

RETURN_CODE: Values can be parsed with [device_getResponseCodeString\(\)](#)

11.4.4.11 `device_getFirmwareVersion_Len()`

```
int device_getFirmwareVersion_Len (
    OUT char * firmwareVersion,
    IN_OUT int * firmwareVersionLen )
```

Polls device for Firmware Version

Parameters

<i>firmwareVersion</i>	Response returned of Firmware Version
<i>firmwareVersionLen</i>	Length of Firmware Version

Returns

RETURN_CODE: Values can be parsed with [device_getResponseCodeString\(\)](#)

11.4.4.12 `device_getKeyStatus()`

```
int device_getKeyStatus (
    int * newFormat,
    BYTE * status,
    int * statusLen )
```

Get Key Status

Gets the status of loaded keys

Parameters

<i>status</i>	newFormat for Augusta and miniSmartII only 1: new format of key status 0: reserved format for support previous device
<i>status</i>	For L80, L100, Augusta and miniSmartII: When the newFormat is 0, data format as follows. For Augusta and miniSmartII: 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 For L80 and L100: byte 0: PIN DUKPT Key byte 1: PIN Master Key byte 2: Standard PIN Session Key byte 3: Desjardins PIN Session Key byte 4: Account/MSR DUKPT Key, 1 Exists, 0 None, 0xFF STOP, Does not support, always 0 byte 5: Account/ICC DUKPT Key, 1 Exists, 0 None, 0xFF STOP, Does not support, always 0 byte 6: Admin DUKPT Key, 1 Exists, 0 None, 0xFF STOP byte 7: Data DUKPT Key, 1 Exists, 0 None, 0xFF STOP byte 8: MAC DUKPT Key, 1 Exists, 0 None, 0xFF STOP

when the newFormat is 1, 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) For NEO2 and SREDKey2: Each unit of three bytes represents one key's parameters (index and slot). Key Name Index (1 byte): 0x14 - LCL-KEK 0x01 - Pin encryption Key (NEO2 only) 0x02 - Data encryption Key 0x05 - MAC DUKPT Key 0x0A - PCI Pairing Key (NEO2 only) Key Slot (2 bytes): Indicate different slots of a certain Key Name Example: slot =5 (0x00 0x05), slot=300 (0x01 0x2C) For BTPay380, slot is always 0 For example, 0x14 0x00 0x00 0x02 0x00 0x00 0x0A 0x00 0x00 will represent [KeyNameIndex=0x14,KeySlot=0x0000], [KeyNameIndex=0x02,KeySlot=0x0000] and [KeyNameIndex=0x0A,KeySlot=0x0000]

Parameters

<i>statusLen</i>	the length of status
------------------	----------------------

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString

11.4.4.13 device_getResponseCodeString()

```
void device_getResponseCodeString (
    IN int  returnCode,
    OUT char * description )
```

Review the return code description.

Parameters

<i>returnCode</i>	the response result.
<i>description</i>	

Return values

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

- 0: "no error, beginning task";
- 1: "no response from reader";
- 2: "invalid response data";
- 3: "time out for task or CMD";
- 4: "wrong parameter";
- 5: "SDK is doing MSR or ICC task";
- 6: "SDK is doing PINPad task";
- 7: "SDK is doing CTLS task";

- 8: "SDK is doing EMV task";
- 9: "SDK is doing Other task";
- 10: "err response or data";
- 11: "no reader attached";
- 12: "mono audio is enabled";
- 13: "did connection";
- 14: "audio volume is too low";
- 15: "task or CMD be canceled";
- 16: "UF wrong string format";
- 17: "UF file not found";
- 18: "UF wrong file format";
- 19: "Attempt to contact online host failed";
- 20: "Attempt to perform RKI failed";
- 22: "Buffer size is not enough";
- 0X300: "Key Type(TDES) of Session Key is not same as the related Master Key.";
- 0X400: "Related Key was not loaded.";
- 0X500: "Key Same.";
- 0X501: "Key is all zero";
- 0X502: "TR-31 format error";
- 0X702: "PAN is Error Key.";
- 0X705: "No Internal MSR PAN (or Internal MSR PAN is erased timeout)";
- 0X0C01: "Incorrect Frame Tag";
- 0X0C02: "Incorrect Frame Type";
- 0X0C03: "Unknown Frame Type";
- 0X0C04: "Unknown Command";
- 0X0C05: "Unknown Sub-Command";
- 0X0C06: "CRC Error";
- 0X0C07: "Failed";
- 0X0C08: "Timeout";
- 0X0C0A: "Incorrect Parameter";
- 0X0C0B: "Command Not Supported";
- 0X0C0C: "Sub-Command Not Supported";
- 0X0C0D: "Parameter Not Supported / Status Abort Command";
- 0X0C0F: "Sub-Command Not Allowed";
- 0X0D01: "Incorrect Header Tag";
- 0X0D02: "Unknown Command";

- 0X0D03: "Unknown Sub-Command";
- 0X0D04: "CRC Error in Frame";
- 0X0D05: "Incorrect Parameter";
- 0X0D06: "Parameter Not Supported";
- 0X0D07: "Mal-formatted Data";
- 0X0D08: "Timeout";
- 0X0D0A: "Failed / NACK";
- 0X0D0B: "Command not Allowed";
- 0X0D0C: "Sub-Command not Allowed";
- 0X0D0D: "Buffer Overflow (Data Length too large for reader buffer)";
- 0X0D0E: "User Interface Event";
- 0X0D11: "Communication type not supported, VT-1, burst, etc.";
- 0X0D12: "Secure interface is not functional or is in an intermediate state.";
- 0X0D13: "Data field is not mod 8";
- 0X0D14: "Pad - 0X80 not found where expected";
- 0X0D15: "Specified key type is invalid";
- 0X0D1: "Could not retrieve key from the SAM(InitSecureComm)";
- 0X0D17: "Hash code problem";
- 0X0D18: "Could not store the key into the SAM(InstallKey)";
- 0X0D19: "Frame is too large";
- 0X0D1A: "Unit powered up in authentication state but POS must resend the InitSecureComm command";
- 0X0D1B: "The EEPROM may not be initialized because SecCommInterface does not make sense";
- 0X0D1C: "Problem encoding APDU";
- 0X0D20: "Unsupported Index(ILM) SAM Transceiver error - problem communicating with the SAM(Key Mgr)";
- 0X0D2: "Unexpected Sequence Counter in multiple frames for single bitmap(ILM) Length error in data returned from the SAM(Key Mgr)";
- 0X0D22: "Improper bit map(ILM)";
- 0X0D23: "Request Online Authorization";
- 0X0D24: "ViVOCard3 raw data read successful";
- 0X0D25: "Message index not available(ILM) ViVOcomm activate transaction card type(ViVOcomm)";
- 0X0D26: "Version Information Mismatch(ILM)";
- 0X0D27: "Not sending commands in correct index message index(ILM)";
- 0X0D28: "Time out or next expected message not received(ILM)";
- 0X0D29: "ILM languages not available for viewing(ILM)";
- 0X0D2A: "Other language not supported(ILM)";
- 0X0D41: "Unknown Error from SAM";

- 0X0D42: "Invalid data detected by SAM";
- 0X0D43: "Incomplete data detected by SAM";
- 0X0D44: "Reserved";
- 0X0D45: "Invalid key hash algorithm";
- 0X0D46: "Invalid key encryption algorithm";
- 0X0D47: "Invalid modulus length";
- 0X0D48: "Invalid exponent";
- 0X0D49: "Key already exists";
- 0X0D4A: "No space for new RID";
- 0X0D4B: "Key not found";
- 0X0D4C: "Crypto not responding";
- 0X0D4D: "Crypto communication error";
- 0X0D4E: "Module-specific error for Key Manager";
- 0X0D4F: "All key slots are full (maximum number of keys has been installed)";
- 0X0D50: "Auto-Switch OK";
- 0X0D51: "Auto-Switch failed";
- 0X0D90: "Account DUKPT Key not exist";
- 0X0D91: "Account DUKPT Key KSN exhausted";
- 0X0D00: "This Key had been loaded.";
- 0X0E00: "Base Time was loaded.";
- 0X0F00: "Encryption Or Decryption Failed.";
- 0X1000: "Battery Low Warning (It is High Priority Response while Battery is Low.)";
- 0X1800: "Send 'Cancel Command' after send 'Get Encrypted PIN' & 'Get Numeric' & 'Get Amount'; - 0↔
- X1900: "Press 'Cancel' key after send 'Get Encrypted PIN' & 'Get Numeric' & 'Get Amount';
- 0X30FF: "Security Chip is not connect";
- 0X3000: "Security Chip is deactivation & Device is In Removal Legally State.";
- 0X3101: "Security Chip is activation & Device is In Removal Legally State.";
- 0X5500: "No Admin DUKPT Key.";
- 0X5501: "Admin DUKPT Key STOP.";
- 0X5502: "Admin DUKPT Key KSN is Error.";
- 0X5503: "Get Authentication Code1 Failed.";
- 0X5504: "Validate Authentication Code Error.";
- 0X5505: "Encrypt or Decrypt data failed.";
- 0X5506: "Not Support the New Key Type.";
- 0X5507: "New Key Index is Error.";
- 0X5508: "Step Error.";

- 0X5509: "KSN Error";
- 0X550A: "MAC Error.";
- 0X550B: "Key Usage Error.";
- 0X550C: "Mode Of Use Error.";
- 0X550F: "Other Error.";
- 0X6000: "Save or Config Failed / Or Read Config Error.";
- 0X6200: "No Serial Number.";
- 0X6900: "Invalid Command - Protocol is right, but task ID is invalid.";
- 0X6A01: "Unsupported Command - Protocol and task ID are right, but command is invalid - In this State";
- 0X6A00: "Unsupported Command - Protocol and task ID are right, but command is invalid.";
- 0X6B00: "Unknown parameter in command - Protocol task ID and command are right, but parameter is invalid.";
- 0X6C00: "Unknown parameter in command - Protocol task ID and command are right, but length is out of the requirement.";
- 0X7200: "Device is suspend (MKSK suspend or press password suspend).";
- 0X7300: "PIN DUKPT is STOP (21 bit 1).";
- 0X7400: "Device is Busy.";
- 0XE100: "Can not enter sleep mode";
- 0XE200: "File has existed";
- 0XE300: "File has not existed";
- 0XE313: "IO line low -- Card error after session start";
- 0XE400: "Open File Error";
- 0XE500: "SmartCard Error";
- 0XE600: "Get MSR Card data is error";
- 0XE700: "Command time out";
- 0XE800: "File read or write is error";
- 0XE900: "Active 1850 error!";
- 0XEA00: "Load bootloader error";
- 0XEF00: "Protocol Error- STX or ETX or check error.";
- 0XEB00: "Picture is not exist";
- 0X2C02: "No Microprocessor ICC seated";
- 0X2C06: "no card seated to request ATR";
- 0X2D01: "Card Not Supported,";
- 0X2D03: "Card Not Supported, wants CRC";
- 0X690D: "Command not supported on reader without ICC support";
- 0X8100: "ICC error time out on power-up";
- 0X8200: "invalid TS character received - Wrong operation step";

- 0X8300: "Decode MSR Error";
- 0X8400: "TriMagII no Response";
- 0X8500: "No Swipe MSR Card";
- 0X8510: "No Financial Card";
- 0X8600: "Unsupported F, D, or combination of F and D";
- 0X8700: "protocol not supported EMV TD1 out of range";
- 0X8800: "power not at proper level";
- 0X8900: "ATR length too long";
- 0X8B01: "EMV invalid TA1 byte value";
- 0X8B02: "EMV TB1 required";
- 0X8B03: "EMV Unsupported TB1 only 00 allowed";
- 0X8B04: "EMV Card Error, invalid BWI or CWI";
- 0X8B06: "EMV TB2 not allowed in ATR";
- 0X8B07: "EMV TC2 out of range";
- 0X8B08: "EMV TC2 out of range";
- 0X8B09: "per EMV96 TA3 must be > - 0XF";
- 0X8B10: "ICC error on power-up";
- 0X8B11: "EMV T=1 then TB3 required";
- 0X8B12: "Card Error, invalid BWI or CWI";
- 0X8B13: "Card Error, invalid BWI or CWI";
- 0X8B17: "EMV TC1/TB3 conflict-";
- 0X8B20: "EMV TD2 out of range must be T=1";
- 0X8C00: "TCK error";
- 0XA304: "connector has no voltage setting";
- 0XA305: "ICC error on power-up invalid (SBLK(IFSD) exchange";
- 0XE301: "ICC error after session start";
- 0XFF00: "Request to go online";
- 0XFF01: "EMV: Accept the offline transaction";
- 0XFF02: "EMV: Decline the offline transaction";
- 0XFF03: "EMV: Accept the online transaction";
- 0XFF04: "EMV: Decline the online transaction";
- 0XFF05: "EMV: Application may fallback to magstripe technology";
- 0XFF06: "EMV: ICC detected tah the conditions of use are not satisfied";
- 0XFF07: "EMV: ICC didn't accept transaction";
- 0XFF08: "EMV: Transaction was cancelled";
- 0XFF09: "EMV: Application was not selected by kernel or ICC format error or ICC missing data error";

- 0XFF0A: "EMV: Transaction is terminated";
- 0XFF0B: "EMV: Other EMV Error";
- 0XFFFF: "NO RESPONSE";
- 0XF002: "ICC communication timeout";
- 0XF003: "ICC communication Error";
- 0XF00F: "ICC Card Seated and Highest Priority, disable MSR work request";
- 0XF200: "AID List / Application Data is not exist";
- 0XF201: "Terminal Data is not exist";
- 0XF202: "TLV format is error";
- 0XF203: "AID List is full";
- 0XF204: "Any CA Key is not exist";
- 0XF205: "CA Key RID is not exist";
- 0XF206: "CA Key Index it not exist";
- 0XF207: "CA Key is full";
- 0XF208: "CA Key Hash Value is Error";
- 0XF209: "Transaction format error";
- 0XF20A: "The command will not be processing";
- 0XF20B: "CRL is not exist";
- 0XF20C: "CRL number exceed max number";
- 0XF20D: "Amount,Other Amount,Trasaction Type are missing";
- 0XF20E: "The Identification of algorithm is mistake";
- 0XF20F: "No Financial Card";
- 0XF210: "In Encrypt Result state, TLV total Length is greater than Max Length";
- 0X1001: "INVALID ARG";
- 0X1002: "FILE_OPEN_FAILED";
- 0X1003: "FILE OPERATION_FAILED";
- 0X2001: "MEMORY_NOT_ENOUGH";
- 0X3002: "SMARTCARD_FAIL";
- 0X3003: "SMARTCARD_INIT_FAILED";
- 0X3004: "FALLBACK_SITUATION";
- 0X3005: "SMARTCARD_ABSENT";
- 0X3006: "SMARTCARD_TIMEOUT";
- 0X3012: "EMV_RESULT_CODE_MSR_CARD_ERROR_FALLBACK";
- 0X5001: "EMV_PARSING_TAGS_FAILED";
- 0X5002: "EMV_DUPLICATE_CARD_DATA_ELEMENT";
- 0X5003: "EMV_DATA_FORMAT_INCORRECT";

- 0X5004: "EMV_NO_TERM_APP";
- 0X5005: "EMV_NO_MATCHING_APP";
- 0X5006: "EMV_MISSING_MANDATORY_OBJECT";
- 0X5007: "EMV_APP_SELECTION_RETRY";
- 0X5008: "EMV_GET_AMOUNT_ERROR";
- 0X5009: "EMV_CARD_REJECTED";
- 0X5010: "EMV_AIP_NOT_RECEIVED";
- 0X5011: "EMV_AFL_NOT_RECEIVED";
- 0X5012: "EMV_AFL_LEN_OUT_OF_RANGE";
- 0X5013: "EMV_SFI_OUT_OF_RANGE";
- 0X5014: "EMV_AFL_INCORRECT";
- 0X5015: "EMV_EXP_DATE_INCORRECT";
- 0X5016: "EMV_EFF_DATE_INCORRECT";
- 0X5017: "EMV_ISS_COD_TBL_OUT_OF_RANGE";
- 0X5018: "EMV_CRYPTOGAM_TYPE_INCORRECT";
- 0X5019: "EMV_PSE_NOT_SUPPORTED_BY_CARD";
- 0X5020: "EMV_USER_SELECTED_LANGUAGE";
- 0X5021: "EMV_SERVICE_NOT_ALLOWED";
- 0X5022: "EMV_NO_TAG_FOUND";
- 0X5023: "EMV_CARD_BLOCKED";
- 0X5024: "EMV_LEN_INCORRECT";
- 0X5025: "CARD_COM_ERROR";
- 0X5026: "EMV_TSC_NOT_INCREASED";
- 0X5027: "EMV_HASH_INCORRECT";
- 0X5028: "EMV_NO_ARC";
- 0X5029: "EMV_INVALID_ARC";
- 0X5030: "EMV_NO_ONLINE_COMM";
- 0X5031: "TRAN_TYPE_INCORRECT";
- 0X5032: "EMV_APP_NO_SUPPORT";
- 0X5033: "EMV_APP_NOT_SELECT";
- 0X5034: "EMV_LANG_NOT_SELECT";
- 0X5035: "EMV_NO_TERM_DATA";
- 0X5039: "EMV_PIN_ENTRY_TIMEOUT";
- 0X6001: "CVM_TYPE_UNKNOWN";
- 0X6002: "CVM_AIP_NOT_SUPPORTED";
- 0X6003: "CVM_TAG_8E_MISSING";

- 0X6004: "CVM_TAG_8E_FORMAT_ERROR";
- 0X6005: "CVM_CODE_IS_NOT_SUPPORTED";
- 0X6006: "CVM_COND_CODE_IS_NOT_SUPPORTED";
- 0X6007: "NO_MORE_CVM";
- 0X6008: "PIN_BYPASSED_BEFORE";
- 0X7001: "PK_BUFFER_SIZE_TOO_BIG";
- 0X7002: "PK_FILE_WRITE_ERROR";
- 0X7003: "PK_HASH_ERROR";
- 0X8001: "NO_CARD_HOLDER_CONFIRMATION";
- 0X8002: "GET_ONLINE_PIN";
- 0XD000: "Data not exist";
- 0XD001: "Data access error";
- 0XD100: "RID not exist";
- 0XD101: "RID existed";
- 0XD102: "Index not exist";
- 0XD200: "Maximum exceeded";
- 0XD201: "Hash error";
- 0XD205: "System Busy";
- 0X0E01: "Unable to go online";
- 0X0E02: "Technical Issue";
- 0X0E03: "Declined";
- 0X0E04: "Issuer Referral transaction";
- 0X0F01: "Decline the online transaction";
- 0X0F02: "Request to go online";
- 0X0F03: "Transaction is terminated";
- 0X0F05: "Application was not selected by kernel or ICC format error or ICC missing data error";
- 0X0F07: "ICC didn't accept transaction";
- 0X0F0A: "Application may fallback to magstripe technology";
- 0X0F0C: "Transaction was cancelled";
- 0X0F0D: "Timeout";
- 0X0F0F: "Other EMV Error";
- 0X0F10: "Accept the offline transaction";
- 0X0F11: "Decline the offline transaction";
- 0X0F21: "ICC detected the conditions of use are not satisfied";
- 0X0F22: "No app were found on card matching terminal configuration";
- 0X0F23: "Terminal file does not exist";

- 0X0F24: "CAPK file does not exist";
- 0X0F25: "CRL Entry does not exist";
- 0X0FFE: "code when blocking is disabled";
- 0X0FFF: "code when command is not applicable on the selected device";
- 0XF005: "ICC Encrypted C-APDU Data Structure Length Error Or Format Error.";
- 0XBBE0: "CM100 Success";
- 0XBBE1: "CM100 Parameter Error";
- 0XBBE2: "CM100 Low Output Buffer";
- 0XBBE3: "CM100 Card Not Found";
- 0XBBE4: "CM100 Collision Card Exists";
- 0XBBE5: "CM100 Too Many Cards Exist";
- 0XBBE6: "CM100 Saved Data Does Not Exist";
- 0XBBE8: "CM100 No Data Available";
- 0XBBE9: "CM100 Invalid CID Returned";
- 0XBBEA: "CM100 Invalid Card Exists";
- 0XBBE C: "CM100 Command Unsupported";
- 0XBBED: "CM100 Error In Command Process";
- 0XB BEE: "CM100 Invalid Command";
- 0X9031: "Unknown command";
- 0X9032: "Wrong parameter (such as the length of the command is incorrect)";
- 0X9038: "Wait (the command couldnt be finished in BWT)";
- 0X9039: "Busy (a previously command has not been finished)";
- 0X903A: "Number of retries over limit";
- 0X9040: "Invalid Manufacturing system data";
- 0X9041: "Not authenticated";
- 0X9042: "Invalid Master DUKPT Key";
- 0X9043: "Invalid MAC Key";
- 0X9044: "Reserved for future use";
- 0X9045: "Reserved for future use";
- 0X9046: "Invalid DATA DUKPT Key";
- 0X9047: "Invalid PIN Pairing DUKPT Key";
- 0X9048: "Invalid DATA Pairing DUKPT Key";
- 0X9049: "No nonce generated";
- 0X9949: "No GUID available. Perform getVersion first.";
- 0X9950: "MAC Calculation unsuccessful. Check BDk value.";
- 0X904A: "Not ready";

- 0X904B: "Not MAC data";
- 0X9050: "Invalid Certificate";
- 0X9051: "Duplicate key detected";
- 0X9052: "AT checks failed";
- 0X9053: "TR34 checks failed";
- 0X9054: "TR31 checks failed";
- 0X9055: "MAC checks failed";
- 0X9056: "Firmware download failed";
- 0X9060: "Log is full";
- 0X9061: "Removal sensor unengaged";
- 0X9062: "Any hardware problems";
- 0X9070: "ICC communication timeout";
- 0X9071: "ICC data error (such check sum error)";
- 0X9072: "Smart Card not powered up";

11.4.4.14 device_init()

```
int device_init ( )
```

Initial the device by USB

It will detect the device and trying connect.

The connect status can be checked by [device_isConnected\(\)](#).

Note: after the function returns success, the function [device_setCurrentDevice\(\)](#) has to be called to set the device type.

Returns

RETURN_CODE: Values can be parsed with [device_getResponseCodeString\(\)](#)

11.4.4.15 device_isAttached()

```
int device_isAttached (
    int deviceType )
```

Check if the device is attached to the USB port The function [device_init\(\)](#) must be called before this function.

Parameters

<i>deviceType,the</i>	device type of the USB device
-----------------------	-------------------------------

Returns

1 if the device is attached, or 0 if the device is not attached

11.4.4.16 device_isConnected()

```
int device_isConnected ( )
```

Check the device connected status

Returns

DEVICE_DISCONNECT=0, or DEVICE_CONNECTED = 1

11.4.4.17 device_rebootDevice()

```
int 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\(\)](#)

11.4.4.18 device_registerCameraCallBk()

```
void device_registerCameraCallBk (
    pCMR_callback pCMRf )
```

To register the camera callback function to get the image data. (Pass NULL to disable the callback.)

11.4.4.19 device_registerCardStatusFrontSwitchCallBk()

```
void device_registerCardStatusFrontSwitchCallBk (
    pCSFS_callback pCSFSf )
```

To register the card status and front switch callback function to get status. (Pass NULL to disable the callback.)

11.4.4.20 device_registerFWCallBk()

```
void device_registerFWCallBk (
    pFW_callback pFWf )
```

To register the firmware update callback function to get the status of firmware update. (Pass NULL to disable the callback.)

11.4.4.21 device_SendDataCommand()

```
int device_SendDataCommand (
    IN BYTE * cmd,
    IN int cmdLen,
```

```

    IN BYTE * data,
    IN int dataLen,
    OUT BYTE * response,
    IN_OUT int * respLen )

```

Send a Command to device

Sends a command to the device .

Parameters

<i>cmd</i>	buffer of command to execute.
<i>cmdLen,the</i>	length of the buffer cmd.
<i>data</i>	buffer of IDG command data.
<i>dataLen,the</i>	length of the buffer data.
<i>response</i>	Response data
<i>respLen,the</i>	length of Response data

Returns

RETURN_CODE: Values can be parsed with [device_getResponseCodeString\(\)](#)

11.4.4.22 device_setCurrentDevice()

```

int device_setCurrentDevice (
    int deviceType )

```

Sets the current device to talk to

The connect status can be checked by [device_isConnected\(\)](#).

Parameters

<i>deviceType</i>	Device to connect to <pre> enum DEVICE_TYPE { IDT_DEVICE_UNKNOWN=0, IDT_DEVICE_AUGUSTA_HID, IDT_DEVICE_AUGUSTA_KB, IDT_DEVICE_AUGUSTA_S_HID, IDT_DEVICE_AUGUSTA_S_KB, IDT_DEVICE_AUGUSTA_S_TTK_HID, IDT_DEVICE_SPECTRUM_PRO, IDT_DEVICE_MINISMART_II, IDT_DEVICE_L80, IDT_DEVICE_L100, IDT_DEVICE_UNIPAY, IDT_DEVICE_UNIPAY_I_V, IDT_DEVICE_VP3300_AJ, IDT_DEVICE_KIOSK_III, IDT_DEVICE_KIOSK_III_S, IDT_DEVICE_PIP_READER, IDT_DEVICE_VENDI, IDT_DEVICE_VP3300_USB, IDT_DEVICE_UNIPAY_I_V_TTK, IDT_DEVICE_VP3300_BT, IDT_DEVICE_VP8800, IDT_DEVICE_SREDKEY2_HID, IDT_DEVICE_SREDKEY2_KB, IDT_DEVICE_NEO2, IDT_DEVICE_MINISMART_II_COM = IDT_DEVICE_NEO2+5, IDT_DEVICE_SPECTRUM_PRO_COM, IDT_DEVICE_KIOSK_III_COM, IDT_DEVICE_KIOSK_III_S_COM, IDT_DEVICE_PIP_READER_COM, IDT_DEVICE_VP3300_COM, IDT_DEVICE_NEO2_COM, IDT_DEVICE_MAX_DEVICES = IDT_DEVICE_NEO2_COM+5 }; </pre>
-------------------	---

Returns

RETURN_CODE: 1: success, 0: failed

11.4.4.23 device_setSleepModeTime()

```

int device_setSleepModeTime (
    int time )

```

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.
-------------	------------------------------------

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString

11.4.4.24 device_updateFirmware()

```
int device_updateFirmware (
    IN BYTE * firmwareData,
    IN int firmwareDataLen,
    IN char * firmwareName,
    IN int encryptionType,
    IN BYTE * keyBlob,
    IN int keyBlobLen )
```

Update Firmware Updates the firmware of Augusta.

Parameters

<i>firmwareData</i>	Signed binary data of a firmware file provided by IDTech
<i>firmwareDataLen</i>	Length of firmwareData
<i>firmwareName</i>	Firmware name. <ul style="list-style-type: none"> For example "Augusta_S_TTK_V1.00.002.fm"
<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)
<i>keyBlobLen</i>	Length of keyBlob

Returns

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

Firmware update status is returned in the callback with the following values: sender = AUGUSTA state = Device↔State.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

11.4.4.25 emv_registerCallBk()

```
void emv_registerCallBk (
    pEMV_callback pEMVf )
```

To register the emv callback function to get the EMV processing response. (Pass NULL to disable the callback.)

11.4.4.26 lcd_displayMessage()

```
int lcd_displayMessage (
    int lineNumber,
    char * message,
    int messageLen )
```

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>messageLen</i>	length of message

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString

11.4.4.27 lcd_displayPrompt()

```
int lcd_displayPrompt (
    int promptNumber,
    int lineNumber )
```

Display Prompt on Line

Displays a message prompt from L80 or L100 memory.

Parameters

<i>promptNumber</i>	Prompt number (0-9)
<i>lineNumber</i>	Line number to display message prompt (1-4)

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString

11.4.4.28 lcd_enableBacklight()

```
int lcd_enableBacklight (
    int enable )
```

Enable/Disable LCD Backlight

Turns on/off the LCD back lighting.

Parameters

<i>enable</i>	TRUE = turn ON backlight, FALSE = turn OFF backlight
---------------	--

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString

11.4.4.29 lcd_getBacklightStatus()

```
int lcd_getBacklightStatus (
    int * enabled )
```

Get Backlight Status

Returns the status of the LCD back lighting.

Parameters

<i>enabled</i>	1 = Backlight is ON, 0 = Backlight is OFF
----------------	---

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString

11.4.4.30 lcd_savePrompt()

```
int lcd_savePrompt (
    int promptNumber,
    char * prompt,
    int promptLen )
```

Save Prompt

Saves a message prompt to L80 or L100 memory.

Parameters

<i>promptNumber</i>	Prompt number (0-9)
<i>prompt</i>	Prompt string (up to 20 characters)
<i>promptLen</i>	length of prompt

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString

11.4.4.31 msr_registerCallBk()

```
void msr_registerCallBk (
    pMSR_callback pMSRf )
```

To register the msr callback function to get the MSR card data. (Pass NULL to disable the callback.)

11.4.4.32 msr_registerCallBkp()

```
void msr_registerCallBkp (
    pMSR_callback pMSRf )
```

To register the msr callback function to get the MSR card data pointer. (Pass NULL to disable the callback.)

11.4.4.33 pin_getEncryptedPIN()

```
int pin_getEncryptedPIN (
    int keyType,
    char * PAN,
    int PANLen,
    char * message,
    int messageLen,
    int timeout )
```

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>PANLen</i>	length of PAN
<i>message</i>	Message to display
<i>messageLen</i>	length of message
<i>timeout</i>	PIN entry timeout

Returns

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

11.4.4.34 pin_getFunctionKey()

```
int pin_getFunctionKey (
    int timeout )
```

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

@param timeout Timeout, in seconds

Returns

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

11.4.4.35 pin_promptForAmountInput()

```
int pin_promptForAmountInput (
    int messageID,
    int languageID,
    int minLen,
    int maxLen,
    int timeout )
```

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 INGR↵ESE	SVP ENTREZ
6	PLEASE REENTER	POR FAVO REENT↵RAR	POR FAVO REINGR↵ESE	SVP RE-ENTREZ
7	PO NUMBER	N↵ERO PO	NUMERO PO	No COMMANDE
8	DRIVER ID	LICEN↵	LICENCIA	ID CONDUCTEUR
9	ODOMETER	ODOMETER	ODOMETRO	ODOMETRE
10	ID NUMBER	N↵ERO 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 N↵ERO	NUMERO EMP	No TRAVAIL
14	WORK ORDER	TRABALHO ORDEM	ORDEN TRABAJO	FICHE TRAVAIL
15	VEHICLE ID	ID VE↵ULO	ID VEHICULO	ID VEHICULE

Msg Id	English Prompt	Portuguese Prompt	Spanish Prompt	French Prompt
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 TRABAJADOR	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 LICENCIA	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 VEHICULO	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 VEICULO	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 USUARIO	INGRESE INFO USUARIO	INFO UTILISATEUR
39	ENTER CUST CODE	ENTER CODE. CLIENTE	INGRESE COD. CLIENTE	ENTR CODE CLIENT
40	ENTER EMPLOYEE	ENTER FUNCIONARIO	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

Msg Id	English Prompt	Portuguese Prompt	Spanish Prompt	French Prompt
47	REENTER VEHICLE	REENTRAR VEICULO	REINGRESE VEHICULO	RE-ENTR VEHICULE
48	ENTER VEHICLE ID	ENTER VEICULO 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 REFERENCIA	INGRESE REFERENCIA	ENTREZ REFERENCE
53	ENTER AUTH NUMBER	ENTER NUMERO 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 CONDUCT
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
67	VEHICLE #	VEICULO #	VEHICULO #	# VEHICULE
68	ENTER VEHICLE #	ENTER VEICULO #	INGRESE VEHICULO #	ENT # VEHICULE
69	JOB #	TRABALHO #	TRABAJO #	# TRAVAIL
70	ENTER JOB #	ENTER JOB #	INGRESE TRABAJO #	ENTREZ # TRAVAIL
71	DEPT NUMBER	NUMERO DEPT	NUMERO DEPTO	No DEPARTEMENT
72	DEPARTMENT #	DEPARTAMENTO #	DEPARTAMENTO #	DEPARTEMENT
73	ENTER DEPT #	ENTER DEPT #	INGRESE DEPTO #	ENT# DEPARTEMENT
74	LICENSE NUMBER	NUMERO DE LICENCA	NUMERO LICENCIA	No PERMIS
75	LICENSE #	LICENCA #	LICENCIA #	# PERMIS
76	ENTER LICENSE #	ENTER LICENCA #	INGRESE LICENCIA #	ENTREZ # PERMIS
77	DATA	INFO	INFO	INFO
78	ENTER DATA	ENTER INFO	INGRESE INFO	ENTREZ INFO

Msg Id	English Prompt	Portuguese Prompt	Spanish Prompt	French Prompt
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 N ^o ERO DE CART ^o	INGRESE NUM TA ^o RJETA	ENTREZ NO CARTE
86	EXP DATE(Yymm)	VALIDADE VAL (AA ^o MM)	FECHA EXP (AAMM)	DATE EXPIR(AAMM)
87	PHONE NUMBER	TELEFONE	NUMERO TELEFONO	NO TEL
88	CVV START DATE	CVV DATA DE IN ^o IO	CVV FECHA INICIO	CVV DATE DE DEB ^o UT
89	ISSUE NUMBER	N ^o ERO DE EMISS ^o	NUMERO DE EMISI ^o ON	NO DEMISSION
90	START DATE (MMYY)	DATA DE IN ^o IO (AA ^o MM)	FECHA INICIO (AA ^o MM)	DATE DE DEBUT-AA ^o MM

```

@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
@param timeout Timeout value, in seconds

```

Returns

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

11.4.4.36 pin_promptForKeyInput()

```

int pin_promptForKeyInput (
    int messageID,
    int languageID,
    int maskInput,
    int minLen,
    int maxLen,
    int timeout )

```

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 INGR ^o ESE	SVP ENTREZ
6	PLEASE REENTER	POR FAVO REENT ^o RAR	POR FAVO REINGR ^o ESE	SVP RE-ENTREZ

Msg Id	English Prompt	Portuguese Prompt	Spanish Prompt	French Prompt
7	PO NUMBER	N ^o PO	NUMERO PO	No COMMANDE
8	DRIVER ID	LICEN ^{ça}	LICENCIA	ID CONDUCTEUR
9	ODOMETER	ODOMETER	ODOMETRO	ODOMETRE
10	ID NUMBER	N ^o 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 N ^o	NUMERO EMP	No TRAVAIL
14	WORK ORDER	TRABALHO ORDEM	ORDEN TRABAJO	FICHE TRAVAIL
15	VEHICLE ID	ID VE ^h ULO	ID VEHICULO	ID VEHICULE
16	ENTER DRIVER	ENTER DRIVER	INGRESE CONDUCTOR	ENTR CONDUCTEUR
17	ENTER DEPT	ENTER DEPT	INGRESE DEPT	ENTR DEPARTEMENT
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	N ^o PATH	RUTA NUMERO	No ROUTE
23	ENTER USER ID	ENTER USER ID	INGRESE ID USUARIO	ID UTILISATEUR
24	FLEET NUMBER	N ^o DE FROTA	FLOTA NUMERO	No PARC AUTO
25	ENTER PRODUCT	ADICIONAR PRODUCTO	INGRESE PRODUCTO	ENTREZ PRODUIT
26	DRIVER NUMBER	N ^o DRIVER	CONDUCTOR NUMERO	No CONDUCTEUR
27	ENTER LICENSE	ENTER LICEN ^{ça}	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 VE ^h ULO	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	N ^o DO VE ^h ULO	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 USUARIO	INGRESE INFO USUARIO	INFO UTILISATEUR
39	ENTER CUST CODE	ENTER CODE. CLIENTE	INGRESE COD. CLIENTE	ENTR CODE CLIENT
40	ENTER EMPLOYEE	ENTER FUNCIONARIO	INGRESE EMPLEADO	ENTREZ EMPLOYE
41	ENTER ID NUMBER	ENTER N ^o ID	INGRESE NUMERO ID	ENTREZ No ID

Msg Id	English Prompt	Portuguese Prompt	Spanish Prompt	French Prompt
42	ENTER DRIVER ID	ENTER ID DRIVER	INGRESE ID COND↵ UCTOR	No CONDUCTEUR
43	ENTER FLEET PIN	ENTER PIN FROTA	INGRESE PIN DE F↵ LOTA	NIP PARC AUTO
44	ODOMETER NUMB↵ ER	N↵ERO ODOMETER	ODOMETRO NUME↵ RO	No ODOMETRE
45	ENTER DRIVER LIC	ENTER DRIVER LIC	INGRESE LIC CON↵ DUCTOR	PERMIS CONDUIRE
46	ENTER TRAILER NO	NRO TRAILER ENT↵ ER	INGRESE NRO TRA↵ ILER	ENT No REMORQUE
47	REENTER VEHICLE	REENTRAR VE↵U↵ LO	REINGRESE VEHIC↵ ULO	RE-ENTR VEHICULE
48	ENTER VEHICLE ID	ENTER VE↵ULO ID	INGRESE ID VEHIC↵ ULO	ENTR ID VEHICULE
49	ENTER BIRTH DATE	INSERIR DATA NAC	INGRESE FECHA N↵ AC	ENT DT NAISSANCE
50	ENTER DOB MMDD↵ YY	ENTER FDN MMDD↵ YY	INGRESE FDN MM↵ DDAA	NAISSANCE MMJJAA
51	ENTER FLEET DATA	ENTER FROTA INFO	INGRESE INFO DE FLOTA	INFO PARC AUTO
52	ENTER REFERENCE	ENTER REFER ↵ CIA	INGRESE REFERE↵ NCIA	ENTREZ REFEREN↵ CE
53	ENTER AUTH NUM↵ BR	ENTER N↵ERO AUT	INGRESE NUMERO AUT	No AUTORISATION
54	ENTER HUB NUMB↵ ER	ENTER HUB NRO	INGRESE NRO HUB	ENTREZ No NOYAU
55	ENTER HUBOMETER	MEDIDA PARA ENT↵ RAR HUB	INGRESE MEDIDO DE HUB	COMPTEUR NOYAU
56	ENTER TRAILER ID	TRAILER ENTER ID	INGRESE ID TRAIL↵ ER	ENT ID REMORQUE
57	ODOMETER READI↵ NG	QUILOMETRAGEM	LECTURA ODOME↵ TRO	LECTURE ODOME↵ TRE
58	REENTER ODOME↵ TER	REENTRAR ODOM↵ ETER	REINGRESE ODO↵ METRO	RE-ENT ODOMETRE
59	REENTER DRIV. ID	REENTRAR DRIVER ID	REINGRESE ID CH↵ OFER	RE-ENT ID CONDUC
60	ENTER CUSTOMER ID	ENTER CLIENTE ID	INGRESE ID CLIEN↵ TE	ENTREZ ID CLIENT
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 # CONDUCTE↵ UR
67	VEHICLE #	VE↵ULO #	VEHICULO #	# VEHICULE
68	ENTER VEHICLE #	ENTER VE↵ULO #	INGRESE VEHICULO #	ENT # VEHICULE
69	JOB #	TRABALHO #	TRABAJO #	# TRAVAIL
70	ENTER JOB #	ENTER JOB #	INGRESE TRABAJO #	ENTREZ # TRAVAIL
71	DEPT NUMBER	N↵ERO DEPT	NUMERO DEPTO	No DEPARTEMENT

Msg Id	English Prompt	Portuguese Prompt	Spanish Prompt	French Prompt
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 TARCJETA	ENTREZ NO CARTE
86	EXP DATE(Yymm)	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 EMISSÃO	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 maskInput 1 = entry is masked with '*', 0 = 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
@param timeout Timeout value, in seconds

```

Returns

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

11.4.4.37 pin_registerCallBk()

```

void pin_registerCallBk (
    pPIN_callback pPINf )

```

To register the pin callback function to get the PINPad data. (Pass NULL to disable the callback.)

11.4.4.38 pin_sendBeep()

```

int pin_sendBeep (
    int frequency,
    int duration )

```

Send Beep

Executes a beep request.

Parameters

<i>frequency</i>	Frequency, range 200-20000Hz Not used for NEO 2 devices
<i>duration</i>	Duration, range 16-65535ms Not used for NEO 2 devices

Returns

RETURN_CODE: Values can be parsed with `device_getResponseCodeString`

11.4.4.39 pin_setKeyValues()

```
int pin_setKeyValues (
    int mode )
```

Set Key Values

Set return key values on or off

Parameters

<i>mode</i>	On: 1, Off: 0
-------------	---------------

Returns

RETURN_CODE: Values can be parsed with [device_getResponseCodeString\(\)](#)

11.4.4.40 registerHotplugCallBk()

```
void registerHotplugCallBk (
    pMessageHotplug pMsgHotplug )
```

To register the USB HID hot-plug callback function which implemented in the application to monitor the hotplug message from the SDK.

11.4.4.41 registerLogCallBk()

```
void registerLogCallBk (
    pSendDataLog pFSend,
    pReadDataLog pFRead )
```

To register the log callback function which implemented in the application to monitor sending/reading data between application and reader.

11.4.4.42 SDK_Version()

```
char* SDK_Version ( )
```

To Get SDK version

Returns

return the SDK version string

11.4.4.43 setAbsoluteLibraryPath()

```
int setAbsoluteLibraryPath (
    const char * absoluteLibraryPath )
```

Set the path to use when searching for ID TECH's libraries. If this is not set, the libraries will be searched for with the system's default procedures.

Parameters

<i>absoluteLibraryPath</i>	The absolute path to ID TECH's libraries.
----------------------------	---

Returns

RETURN_CODE: Values can be parsed with [device_getResponseCodeString\(\)](#)

11.5 Source_C/libIDT_L80.h File Reference

L80 API.

```
#include "IDTDef.h"
```

Macros

- `#define IN`
- `#define OUT`
- `#define IN_OUT`

Typedefs

- `typedef void(* pMessageHotplug) (int, int)`
- `typedef void(* pSendDataLog) (BYTE *, int)`
- `typedef void(* pReadDataLog) (BYTE *, int)`
- `typedef void(* pEMV_callBack) (int, int, BYTE *, int, IDTTransactionData *, EMV_Callback *, int)`
- `typedef void(* pMSR_callBack) (int, IDTMSRData)`
- `typedef void(* pMSR_callBackp) (int, IDTMSRData *)`
- `typedef void(* pPIN_callBack) (int, IDTPINData *)`
- `typedef void(* pCMR_callBack) (int, IDTCMRData *)`
- `typedef void(* pCSFS_callBack) (BYTE status)`
- `typedef void(* pFW_callBack) (int, int, int, int, int)`
- `typedef void(* ftpComm_callBack) (int, int, int)`
- `typedef void(* httpComm_callBack) (BYTE *, int)`
- `typedef void(* v4Comm_callBack) (BYTE, BYTE, BYTE *, int)`

Functions

- void [registerHotplugCallBk](#) (pMessageHotplug pMsgHotplug)
- void [registerLogCallBk](#) (pSendDataLog pFSend, [pReadDataLog](#) pFRead)
- void [emv_registerCallBk](#) (pEMV_callBack pEMVf)
- void [msr_registerCallBk](#) (pMSR_callBack pMSRf)
- void [msr_registerCallBkp](#) (pMSR_callBackp pMSRf)
- void [pin_registerCallBk](#) (pPIN_callBack pPINf)
- void [device_registerCameraCallBk](#) (pCMR_callBack pCMRf)
- void [device_registerCardStatusFrontSwitchCallBk](#) (pCSFS_callBack pCSFSf)
- void [device_registerFWCallBk](#) (pFW_callBack pFWf)
- char * [SDK_Version](#) ()
- int [setAbsoluteLibraryPath](#) (const char *absoluteLibraryPath)
- int [device_init](#) ()
- int [device_setCurrentDevice](#) (int deviceType)
- int [device_close](#) ()
- void [device_getResponseCodeString](#) (IN int returnCode, OUT char *despcrition)
- int [device_isConnected](#) ()
- int [device_isAttached](#) (int deviceType)
- int [device_getFirmwareVersion](#) (OUT char *firmwareVersion)
- int [device_getFirmwareVersion_Len](#) (OUT char *firmwareVersion, IN_OUT int *firmwareVersionLen)
- int [device_getDateTime](#) (OUT BYTE *dateTime)
- int [device_getDateTime_Len](#) (OUT BYTE *dateTime, IN_OUT int *dateTimeLen)
- int [device_getCurrentDeviceType](#) ()
- int [device_SendDataCommand](#) (IN BYTE *cmd, IN int cmdLen, IN BYTE *data, IN int dataLen, OUT BYTE *response, IN_OUT int *respLen)
- int [device_updateFirmware](#) (IN BYTE *firmwareData, IN int firmwareDataLen, IN char *firmwareName, IN int encryptionType, IN BYTE *keyBlob, IN int keyBlobLen)
- int [device_rebootDevice](#) ()
- int [device_getKeyStatus](#) (int *newFormat, BYTE *status, int *statusLen)
- int [device_enterStopMode](#) ()
- int [device_setSleepModeTime](#) (int time)
- int [config_getModelNumber](#) (OUT char *sNumber)
- int [config_getModelNumber_Len](#) (OUT char *sNumber, IN_OUT int *sNumberLen)
- int [config_getSerialNumber](#) (OUT char *sNumber)
- int [config_getSerialNumber_Len](#) (OUT char *sNumber, IN_OUT int *sNumberLen)
- int [pin_getEncryptedPIN](#) (int keyType, char *PAN, int PANLen, char *message, int messageLen, int timeout)
- int [pin_promptForKeyInput](#) (int messageID, int languageID, int maskInput, int minLen, int maxLen, int timeout)
- int [pin_promptForAmountInput](#) (int messageID, int languageID, int minLen, int maxLen, int timeout)
- int [pin_getFunctionKey](#) (int timeout)
- int [pin_sendBeep](#) (int frequency, int duration)
- int [pin_setKeyValues](#) (int mode)
- int [lcd_savePrompt](#) (int promptNumber, char *prompt, int promptLen)
- int [lcd_displayPrompt](#) (int promptNumber, int lineNumber)
- int [lcd_displayMessage](#) (int lineNumber, char *message, int messageLen)
- int [lcd_enableBacklight](#) (int enable)
- int [lcd_getBacklightStatus](#) (int *enabled)

11.5.1 Detailed Description

L80 API.

L80 Global API methods.

11.5.2 Macro Definition Documentation

11.5.2.1 IN

```
#define IN
```

INPUT parameter.

11.5.2.2 IN_OUT

```
#define IN_OUT
```

INPUT / OUTPUT PARAMETER.

11.5.2.3 OUT

```
#define OUT
```

OUTPUT parameter.

11.5.3 Typedef Documentation

11.5.3.1 ftpComm_callBack

```
typedef void(* ftpComm_callBack) (int, int, int)
```

Define the comm callback function to get FTP file transfer status

It should be passed as a parameter in a FTP request, Signature (int, int, int) = response code, current block, total blocks
 RESPONSE CODES: 100 = FILE DOWNLOAD STARTED 101 = FILE BLOCK XX OF XX RECEIVED 102 = FILE DOWNLOAD COMPLETED 103 = FILE DOWNLOAD TERMINATED PREMATURELY

11.5.3.2 httpComm_callBack

```
typedef void(* httpComm_callBack) (BYTE *, int)
```

Define the comm callback function to get the async url data

It should be registered using the comm_registerHTTPCallback

11.5.3.3 pCMR_callBack

```
typedef void(* pCMR_callBack) (int, IDTCMRData *)
```

Define the camera callback function to get the image data

It should be registered using the device_registerCameraCallBk,

11.5.3.4 pCSFS_callBack

```
typedef void(* pCSFS_callBack) (BYTE status)
```

Define the card status and front switch callback function to get card and front switch status

It should be registered using the device_registerCardStatusFrontSwitchCallBk,

11.5.3.5 pEMV_callBack

```
typedef void(* pEMV_callBack) (int, int, BYTE *, int, IDTTransactionData *, EMV_Callback *, int)
```

Define the EMV callback function to get the transaction message/data/result.
It should be registered using the `emv_registerCallBk`,

11.5.3.6 pFW_callBack

```
typedef void(* pFW_callBack) (int, int, int, int, int)
```

Define the firmware update callback function to get the status of firmware update
It should be registered using the `device_registerFWCallBk`,

11.5.3.7 pMessageHotplug

```
typedef void(* pMessageHotplug) (int, int)
```

Define the USB hot-plug callback function to monitor the info when plug in/out the reader.
It should be registered using the `registerHotplugCallBk`, The first integer parameter is device type, and the second integer parameter is either 0: Device Plugged Out or 1: Device Plugged In

11.5.3.8 pMSR_callBack

```
typedef void(* pMSR_callBack) (int, IDTMSRData)
```

Define the MSR callback function to get the MSR card data
It should be registered using the `msr_registerCallBk`, this callback function is for backward compatibility

11.5.3.9 pMSR_callBackp

```
typedef void(* pMSR_callBackp) (int, IDTMSRData *)
```

Define the MSR callback function to get pointer to the MSR card data
It should be registered using the `msr_registerCallBk`, this callback function is recommended instead of `pMSR_callBack`

11.5.3.10 pPIN_callBack

```
typedef void(* pPIN_callBack) (int, IDTPINData *)
```

Define the PINPad callback function to get the input PIN Pad data
It should be registered using the `pin_registerCallBk`,

11.5.3.11 pReadDataLog

```
typedef void(* pReadDataLog) (BYTE *, int)
```

Define the read response callback function to monitor the reading response from the reader.
It should be registered using the `registerLogCallBk`,

11.5.3.12 pSendDataLog

```
typedef void(* pSendDataLog) (BYTE *, int)
```

Define the send command callback function to monitor the sending command into the reader.
It should be registered using the registerLogCallBk,

11.5.3.13 v4Comm_callBack

```
typedef void(* v4Comm_callBack) (BYTE, BYTE, BYTE *, int)
```

Define the comm callback function to receive the V4 Protocol packets received by the device from an external source (IP/USB/RS-232) It should be registered using the comm_registerV4Callback, Data callback will contain command, sub-command, and data from V4 packet

11.5.4 Function Documentation

11.5.4.1 config_getModelNumber()

```
int config_getModelNumber (
    OUT char * sNumber )
```

DEPRECATED : please use [config_getModelNumber_Len\(OUT char* sNumber, IN_OUT int *sNumberLen\)](#)

Polls device for Model Number

Parameters

<i>sNumber</i>	Returns Model Number; needs to have at least 64 bytes of memory
----------------	---

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString

11.5.4.2 config_getModelNumber_Len()

```
int config_getModelNumber_Len (
    OUT char * sNumber,
    IN_OUT int * sNumberLen )
```

Polls device for Model Number

Parameters

<i>sNumber</i>	Returns Model Number
<i>sNumber</i>	length of Model Number

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString

11.5.4.3 config_getSerialNumber()

```
int config_getSerialNumber (
    OUT char * sNumber )
```

DEPRECATED : please use [config_getSerialNumber_Len\(OUT char* sNumber, IN_OUT int *sNumberLen\)](#)

Polls device for Serial Number

Parameters

<i>sNumber</i>	Returns Serial Number; needs to have at least 64 bytes of memory
----------------	--

Returns

RETURN_CODE: Values can be parsed with [device_getResponseCodeString](#)

11.5.4.4 config_getSerialNumber_Len()

```
int config_getSerialNumber_Len (
    OUT char * sNumber,
    IN_OUT int * sNumberLen )
```

Polls device for Serial Number

Parameters

<i>sNumber</i>	Returns Serial Number
<i>sNumberLen</i>	Length of Serial Number

Returns

RETURN_CODE: Values can be parsed with [device_getResponseCodeString](#)

11.5.4.5 device_close()

```
int device_close ( )
```

Close the device

Returns

RETURN_CODE: 0: success, 0x0A: failed

11.5.4.6 device_enterStopMode()

```
int device_enterStopMode ( )
```

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.

Returns

RETURN_CODE: Values can be parsed with `device_getResponseCodeString`

11.5.4.7 device_getCurrentDeviceType()

```
int device_getCurrentDeviceType ( )
```

Get current active device type

Returns

: return the device type defined as DEVICE_TYPE in the IDTDef.h

11.5.4.8 device_getDateTime()

```
int device_getDateTime (
    OUT BYTE * dateTime )
```

Polls device for Date and Time

Parameters

<i>dateTime</i>	Response returned of Date and Time; needs to have at least 6 bytes of memory
-----------------	--

Returns

RETURN_CODE: Values can be parsed with [device_getResponseCodeString\(\)](#)

11.5.4.9 device_getDateTime_Len()

```
int device_getDateTime_Len (
    OUT BYTE * dateTime,
    IN_OUT int * dateTimeLen )
```

Polls device for Date and Time

Parameters

<i>dateTime</i>	Response returned of Date and Time
<i>dateTime</i>	Length of Date and Time

Returns

RETURN_CODE: Values can be parsed with [device_getResponseCodeString\(\)](#)

11.5.4.10 device_getFirmwareVersion()

```
int device_getFirmwareVersion (
    OUT char * firmwareVersion )
```

DEPRECATED : please use [device_getFirmwareVersion_Len](#)(OUT char* firmwareVersion, IN_OUT int *firmwareVersionLen)

Polls device for Firmware Version

Parameters

<i>firmwareVersion</i>	Response returned of Firmware Version; needs to have at least 128 bytes of memory
------------------------	---

Returns

RETURN_CODE: Values can be parsed with [device_getResponseCodeString](#)()

11.5.4.11 device_getFirmwareVersion_Len()

```
int device_getFirmwareVersion_Len (
    OUT char * firmwareVersion,
    IN_OUT int * firmwareVersionLen )
```

Polls device for Firmware Version

Parameters

<i>firmwareVersion</i>	Response returned of Firmware Version
<i>firmwareVersionLen</i>	Length of Firmware Version

Returns

RETURN_CODE: Values can be parsed with [device_getResponseCodeString](#)()

11.5.4.12 device_getKeyStatus()

```
int device_getKeyStatus (
    int * newFormat,
    BYTE * status,
    int * statusLen )
```

Get Key Status

Gets the status of loaded keys

Parameters

<i>status</i>	newFormat for Augusta and miniSmartII only 1: new format of key status 0: reserved format for support previous device
---------------	---

Parameters

<i>status</i>	For L80, L100, Augusta and miniSmartII: When the newFormat is 0, data format as follows. For Augusta and miniSmartII: 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 For L80 and L100: byte 0: PIN DUKPT Key byte 1: PIN Master Key byte 2: Standard PIN Session Key byte 3: Desjardins PIN Session Key byte 4: Account/MSR DUKPT Key, 1 Exists, 0 None, 0xFF STOP, Does not support, always 0 byte 5: Account/ICC DUKPT Key, 1 Exists, 0 None, 0xFF STOP, Does not support, always 0 byte 6: Admin DUKPT Key, 1 Exists, 0 None, 0xFF STOP byte 7: Data DUKPT Key, 1 Exists, 0 None, 0xFF STOP byte 8: MAC DUKPT Key, 1 Exists, 0 None, 0xFF STOP
---------------	--

when the newFormat is 1, 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) For NEO2 and SREDKey2: Each unit of three bytes represents one key's parameters (index and slot). Key Name Index (1 byte): 0x14 - LCL-KEK 0x01 - Pin encryption Key (NEO2 only) 0x02 - Data encryption Key 0x05 - MAC DUKPT Key 0x0A - PCI Pairing Key (NEO2 only) Key Slot (2 bytes): Indicate different slots of a certain Key Name Example: slot =5 (0x00 0x05), slot=300 (0x01 0x2C) For BTPay380, slot is always 0 For example, 0x14 0x00 0x00 0x02 0x00 0x00 0x0A 0x00 0x00 will represent [KeyNameIndex=0x14,KeySlot=0x0000], [KeyNameIndex=0x02,KeySlot=0x0000] and [KeyNameIndex=0x0A,KeySlot=0x0000]

Parameters

<i>statusLen</i>	the length of status
------------------	----------------------

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString

11.5.4.13 device_getResponseCodeString()

```
void device_getResponseCodeString (
    IN int  returnCode,
    OUT char * description )
```

Review the return code description.

Parameters

<i>returnCode</i>	the response result.
<i>description</i>	

Return values

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

- 0: "no error, beginning task";
- 1: "no response from reader";
- 2: "invalid response data";
- 3: "time out for task or CMD";
- 4: "wrong parameter";
- 5: "SDK is doing MSR or ICC task";
- 6: "SDK is doing PINPad task";
- 7: "SDK is doing CTLS task";
- 8: "SDK is doing EMV task";
- 9: "SDK is doing Other task";
- 10: "err response or data";
- 11: "no reader attached";
- 12: "mono audio is enabled";
- 13: "did connection";
- 14: "audio volume is too low";
- 15: "task or CMD be canceled";
- 16: "UF wrong string format";
- 17: "UF file not found";
- 18: "UF wrong file format";
- 19: "Attempt to contact online host failed";
- 20: "Attempt to perform RKL failed";
- 22: "Buffer size is not enough";
- 0X300: "Key Type(TDES) of Session Key is not same as the related Master Key.";
- 0X400: "Related Key was not loaded.";
- 0X500: "Key Same.";
- 0X501: "Key is all zero";
- 0X502: "TR-31 format error";
- 0X702: "PAN is Error Key.";
- 0X705: "No Internal MSR PAN (or Internal MSR PAN is erased timeout)";
- 0X0C01: "Incorrect Frame Tag";
- 0X0C02: "Incorrect Frame Type";
- 0X0C03: "Unknown Frame Type";
- 0X0C04: "Unknown Command";
- 0X0C05: "Unknown Sub-Command";
- 0X0C06: "CRC Error";
- 0X0C07: "Failed";

- 0X0C08: "Timeout";
- 0X0C0A: "Incorrect Parameter";
- 0X0C0B: "Command Not Supported";
- 0X0C0C: "Sub-Command Not Supported";
- 0X0C0D: "Parameter Not Supported / Status Abort Command";
- 0X0C0F: "Sub-Command Not Allowed";
- 0X0D01: "Incorrect Header Tag";
- 0X0D02: "Unknown Command";
- 0X0D03: "Unknown Sub-Command";
- 0X0D04: "CRC Error in Frame";
- 0X0D05: "Incorrect Parameter";
- 0X0D06: "Parameter Not Supported";
- 0X0D07: "Mal-formatted Data";
- 0X0D08: "Timeout";
- 0X0D0A: "Failed / NACK";
- 0X0D0B: "Command not Allowed";
- 0X0D0C: "Sub-Command not Allowed";
- 0X0D0D: "Buffer Overflow (Data Length too large for reader buffer)";
- 0X0D0E: "User Interface Event";
- 0X0D11: "Communication type not supported, VT-1, burst, etc.";
- 0X0D12: "Secure interface is not functional or is in an intermediate state.";
- 0X0D13: "Data field is not mod 8";
- 0X0D14: "Pad - 0X80 not found where expected";
- 0X0D15: "Specified key type is invalid";
- 0X0D1: "Could not retrieve key from the SAM(InitSecureComm)";
- 0X0D17: "Hash code problem";
- 0X0D18: "Could not store the key into the SAM(InstallKey)";
- 0X0D19: "Frame is too large";
- 0X0D1A: "Unit powered up in authentication state but POS must resend the InitSecureComm command";
- 0X0D1B: "The EEPROM may not be initialized because SecCommInterface does not make sense";
- 0X0D1C: "Problem encoding APDU";
- 0X0D20: "Unsupported Index(ILM) SAM Transceiver error - problem communicating with the SAM(Key Mgr)";
- 0X0D2: "Unexpected Sequence Counter in multiple frames for single bitmap(ILM) Length error in data returned from the SAM(Key Mgr)";
- 0X0D22: "Improper bit map(ILM)";
- 0X0D23: "Request Online Authorization";

- 0X0D24: "ViVOCard3 raw data read successful";
- 0X0D25: "Message index not available(ILM) ViVOcomm activate transaction card type(ViVOcomm)";
- 0X0D26: "Version Information Mismatch(ILM)";
- 0X0D27: "Not sending commands in correct index message index(ILM)";
- 0X0D28: "Time out or next expected message not received(ILM)";
- 0X0D29: "ILM languages not available for viewing(ILM)";
- 0X0D2A: "Other language not supported(ILM)";
- 0X0D41: "Unknown Error from SAM";
- 0X0D42: "Invalid data detected by SAM";
- 0X0D43: "Incomplete data detected by SAM";
- 0X0D44: "Reserved";
- 0X0D45: "Invalid key hash algorithm";
- 0X0D46: "Invalid key encryption algorithm";
- 0X0D47: "Invalid modulus length";
- 0X0D48: "Invalid exponent";
- 0X0D49: "Key already exists";
- 0X0D4A: "No space for new RID";
- 0X0D4B: "Key not found";
- 0X0D4C: "Crypto not responding";
- 0X0D4D: "Crypto communication error";
- 0X0D4E: "Module-specific error for Key Manager";
- 0X0D4F: "All key slots are full (maximum number of keys has been installed)";
- 0X0D50: "Auto-Switch OK";
- 0X0D51: "Auto-Switch failed";
- 0X0D90: "Account DUKPT Key not exist";
- 0X0D91: "Account DUKPT Key KSN exhausted";
- 0X0D00: "This Key had been loaded.";
- 0X0E00: "Base Time was loaded.";
- 0X0F00: "Encryption Or Decryption Failed.";
- 0X1000: "Battery Low Warning (It is High Priority Response while Battery is Low.)";
- 0X1800: "Send 'Cancel Command' after send 'Get Encrypted PIN' & 'Get Numeric' & 'Get Amount'; - 0↔
- X1900: "Press 'Cancel' key after send 'Get Encrypted PIN' & 'Get Numeric' & 'Get Amount';
- 0X30FF: "Security Chip is not connect";
- 0X3000: "Security Chip is deactivation & Device is In Removal Legally State.";
- 0X3101: "Security Chip is activation & Device is In Removal Legally State.";
- 0X5500: "No Admin DUKPT Key.";

- 0X5501: "Admin DUKPT Key STOP.";
- 0X5502: "Admin DUKPT Key KSN is Error.";
- 0X5503: "Get Authentication Code1 Failed.";
- 0X5504: "Validate Authentication Code Error.";
- 0X5505: "Encrypt or Decrypt data failed.";
- 0X5506: "Not Support the New Key Type.";
- 0X5507: "New Key Index is Error.";
- 0X5508: "Step Error.";
- 0X5509: "KSN Error";
- 0X550A: "MAC Error.";
- 0X550B: "Key Usage Error.";
- 0X550C: "Mode Of Use Error.";
- 0X550F: "Other Error.";
- 0X6000: "Save or Config Failed / Or Read Config Error.";
- 0X6200: "No Serial Number.";
- 0X6900: "Invalid Command - Protocol is right, but task ID is invalid.";
- 0X6A01: "Unsupported Command - Protocol and task ID are right, but command is invalid - In this State";
- 0X6A00: "Unsupported Command - Protocol and task ID are right, but command is invalid.";
- 0X6B00: "Unknown parameter in command - Protocol task ID and command are right, but parameter is invalid.";
- 0X6C00: "Unknown parameter in command - Protocol task ID and command are right, but length is out of the requirement.";
- 0X7200: "Device is suspend (MKSK suspend or press password suspend).";
- 0X7300: "PIN DUKPT is STOP (21 bit 1).";
- 0X7400: "Device is Busy.";
- 0XE100: "Can not enter sleep mode";
- 0XE200: "File has existed";
- 0XE300: "File has not existed";
- 0XE313: "IO line low -- Card error after session start";
- 0XE400: "Open File Error";
- 0XE500: "SmartCard Error";
- 0XE600: "Get MSR Card data is error";
- 0XE700: "Command time out";
- 0XE800: "File read or write is error";
- 0XE900: "Active 1850 error!";
- 0XEA00: "Load bootloader error";
- 0XEF00: "Protocol Error- STX or ETX or check error.";

- 0XEB00: "Picture is not exist";
- 0X2C02: "No Microprocessor ICC seated";
- 0X2C06: "no card seated to request ATR";
- 0X2D01: "Card Not Supported,";
- 0X2D03: "Card Not Supported, wants CRC";
- 0X690D: "Command not supported on reader without ICC support";
- 0X8100: "ICC error time out on power-up";
- 0X8200: "invalid TS character received - Wrong operation step";
- 0X8300: "Decode MSR Error";
- 0X8400: "TriMagII no Response";
- 0X8500: "No Swipe MSR Card";
- 0X8510: "No Financial Card";
- 0X8600: "Unsupported F, D, or combination of F and D";
- 0X8700: "protocol not supported EMV TD1 out of range";
- 0X8800: "power not at proper level";
- 0X8900: "ATR length too long";
- 0X8B01: "EMV invalid TA1 byte value";
- 0X8B02: "EMV TB1 required";
- 0X8B03: "EMV Unsupported TB1 only 00 allowed";
- 0X8B04: "EMV Card Error, invalid BWI or CWI";
- 0X8B06: "EMV TB2 not allowed in ATR";
- 0X8B07: "EMV TC2 out of range";
- 0X8B08: "EMV TC2 out of range";
- 0X8B09: "per EMV96 TA3 must be > - 0XF";
- 0X8B10: "ICC error on power-up";
- 0X8B11: "EMV T=1 then TB3 required";
- 0X8B12: "Card Error, invalid BWI or CWI";
- 0X8B13: "Card Error, invalid BWI or CWI";
- 0X8B17: "EMV TC1/TB3 conflict-";
- 0X8B20: "EMV TD2 out of range must be T=1";
- 0X8C00: "TCK error";
- 0XA304: "connector has no voltage setting";
- 0XA305: "ICC error on power-up invalid (SBLK(IFSD) exchange";
- 0XE301: "ICC error after session start";
- 0XFF00: "Request to go online";
- 0XFF01: "EMV: Accept the offline transaction";

- 0XFF02: "EMV: Decline the offline transaction";
- 0XFF03: "EMV: Accept the online transaction";
- 0XFF04: "EMV: Decline the online transaction";
- 0XFF05: "EMV: Application may fallback to magstripe technology";
- 0XFF06: "EMV: ICC detected tah the conditions of use are not satisfied";
- 0XFF07: "EMV: ICC didn't accept transaction";
- 0XFF08: "EMV: Transaction was cancelled";
- 0XFF09: "EMV: Application was not selected by kernel or ICC format error or ICC missing data error";
- 0XFF0A: "EMV: Transaction is terminated";
- 0XFF0B: "EMV: Other EMV Error";
- 0XFFFF: "NO RESPONSE";
- 0XF002: "ICC communication timeout";
- 0XF003: "ICC communication Error";
- 0XF00F: "ICC Card Seated and Highest Priority, disable MSR work request";
- 0XF200: "AID List / Application Data is not exist";
- 0XF201: "Terminal Data is not exist";
- 0XF202: "TLV format is error";
- 0XF203: "AID List is full";
- 0XF204: "Any CA Key is not exist";
- 0XF205: "CA Key RID is not exist";
- 0XF206: "CA Key Index it not exist";
- 0XF207: "CA Key is full";
- 0XF208: "CA Key Hash Value is Error";
- 0XF209: "Transaction format error";
- 0XF20A: "The command will not be processing";
- 0XF20B: "CRL is not exist";
- 0XF20C: "CRL number exceed max number";
- 0XF20D: "Amount,Other Amount,Trasaction Type are missing";
- 0XF20E: "The Identification of algorithm is mistake";
- 0XF20F: "No Financial Card";
- 0XF210: "In Encrypt Result state, TLV total Length is greater than Max Length";
- 0X1001: "INVALID ARG";
- 0X1002: "FILE_OPEN_FAILED";
- 0X1003: "FILE OPERATION_FAILED";
- 0X2001: "MEMORY_NOT_ENOUGH";
- 0X3002: "SMARTCARD_FAIL";

- 0X3003: "SMARTCARD_INIT_FAILED";
- 0X3004: "FALLBACK_SITUATION";
- 0X3005: "SMARTCARD_ABSENT";
- 0X3006: "SMARTCARD_TIMEOUT";
- 0X3012: "EMV_RESULT_CODE_MSR_CARD_ERROR_FALLBACK";
- 0X5001: "EMV_PARSING_TAGS_FAILED";
- 0X5002: "EMV_DUPLICATE_CARD_DATA_ELEMENT";
- 0X5003: "EMV_DATA_FORMAT_INCORRECT";
- 0X5004: "EMV_NO_TERM_APP";
- 0X5005: "EMV_NO_MATCHING_APP";
- 0X5006: "EMV_MISSING_MANDATORY_OBJECT";
- 0X5007: "EMV_APP_SELECTION_RETRY";
- 0X5008: "EMV_GET_AMOUNT_ERROR";
- 0X5009: "EMV_CARD_REJECTED";
- 0X5010: "EMV_AIP_NOT_RECEIVED";
- 0X5011: "EMV_AFL_NOT_RECEIVED";
- 0X5012: "EMV_AFL_LEN_OUT_OF_RANGE";
- 0X5013: "EMV_SFI_OUT_OF_RANGE";
- 0X5014: "EMV_AFL_INCORRECT";
- 0X5015: "EMV_EXP_DATE_INCORRECT";
- 0X5016: "EMV_EFF_DATE_INCORRECT";
- 0X5017: "EMV_ISS_COD_TBL_OUT_OF_RANGE";
- 0X5018: "EMV_CRYPTOGAM_TYPE_INCORRECT";
- 0X5019: "EMV_PSE_NOT_SUPPORTED_BY_CARD";
- 0X5020: "EMV_USER_SELECTED_LANGUAGE";
- 0X5021: "EMV_SERVICE_NOT_ALLOWED";
- 0X5022: "EMV_NO_TAG_FOUND";
- 0X5023: "EMV_CARD_BLOCKED";
- 0X5024: "EMV_LEN_INCORRECT";
- 0X5025: "CARD_COM_ERROR";
- 0X5026: "EMV_TSC_NOT_INCREASED";
- 0X5027: "EMV_HASH_INCORRECT";
- 0X5028: "EMV_NO_ARC";
- 0X5029: "EMV_INVALID_ARC";
- 0X5030: "EMV_NO_ONLINE_COMM";
- 0X5031: "TRAN_TYPE_INCORRECT";

- 0X5032: "EMV_APP_NO_SUPPORT";
- 0X5033: "EMV_APP_NOT_SELECT";
- 0X5034: "EMV_LANG_NOT_SELECT";
- 0X5035: "EMV_NO_TERM_DATA";
- 0X5039: "EMV_PIN_ENTRY_TIMEOUT";
- 0X6001: "CVM_TYPE_UNKNOWN";
- 0X6002: "CVM_AIP_NOT_SUPPORTED";
- 0X6003: "CVM_TAG_8E_MISSING";
- 0X6004: "CVM_TAG_8E_FORMAT_ERROR";
- 0X6005: "CVM_CODE_IS_NOT_SUPPORTED";
- 0X6006: "CVM_COND_CODE_IS_NOT_SUPPORTED";
- 0X6007: "NO_MORE_CVM";
- 0X6008: "PIN_BYPASSED_BEFORE";
- 0X7001: "PK_BUFFER_SIZE_TOO_BIG";
- 0X7002: "PK_FILE_WRITE_ERROR";
- 0X7003: "PK_HASH_ERROR";
- 0X8001: "NO_CARD HOLDER CONFIRMATION";
- 0X8002: "GET_ONLINE_PIN";
- 0XD000: "Data not exist";
- 0XD001: "Data access error";
- 0XD100: "RID not exist";
- 0XD101: "RID existed";
- 0XD102: "Index not exist";
- 0XD200: "Maximum exceeded";
- 0XD201: "Hash error";
- 0XD205: "System Busy";
- 0X0E01: "Unable to go online";
- 0X0E02: "Technical Issue";
- 0X0E03: "Declined";
- 0X0E04: "Issuer Referral transaction";
- 0X0F01: "Decline the online transaction";
- 0X0F02: "Request to go online";
- 0X0F03: "Transaction is terminated";
- 0X0F05: "Application was not selected by kernel or ICC format error or ICC missing data error";
- 0X0F07: "ICC didn't accept transaction";
- 0X0F0A: "Application may fallback to magstripe technology";

- 0X0F0C: "Transaction was cancelled";
- 0X0F0D: "Timeout";
- 0X0F0F: "Other EMV Error";
- 0X0F10: "Accept the offline transaction";
- 0X0F11: "Decline the offline transaction";
- 0X0F21: "ICC detected tah the conditions of use are not satisfied";
- 0X0F22: "No app were found on card matching terminal configuration";
- 0X0F23: "Terminal file does not exist";
- 0X0F24: "CAPK file does not exist";
- 0X0F25: "CRL Entry does not exist";
- 0X0FFE: "code when blocking is disabled";
- 0X0FFF: "code when command is not applicable on the selected device";
- 0XF005: "ICC Encrypted C-APDU Data Structure Length Error Or Format Error.";
- 0XBBE0: "CM100 Success";
- 0XBBE1: "CM100 Parameter Error";
- 0XBBE2: "CM100 Low Output Buffer";
- 0XBBE3: "CM100 Card Not Found";
- 0XBBE4: "CM100 Collision Card Exists";
- 0XBBE5: "CM100 Too Many Cards Exist";
- 0XBBE6: "CM100 Saved Data Does Not Exist";
- 0XBBE8: "CM100 No Data Available";
- 0XBBE9: "CM100 Invalid CID Returned";
- 0XBBEA: "CM100 Invalid Card Exists";
- 0XBBE C: "CM100 Command Unsupported";
- 0XBBED: "CM100 Error In Command Process";
- 0XB BEE: "CM100 Invalid Command";
- 0X9031: "Unknown command";
- 0X9032: "Wrong parameter (such as the length of the command is incorrect)";
- 0X9038: "Wait (the command couldnt be finished in BWT)";
- 0X9039: "Busy (a previously command has not been finished)";
- 0X903A: "Number of retries over limit";
- 0X9040: "Invalid Manufacturing system data";
- 0X9041: "Not authenticated";
- 0X9042: "Invalid Master DUKPT Key";
- 0X9043: "Invalid MAC Key";
- 0X9044: "Reserved for future use";

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

11.5.4.14 `device_init()`

```
int device_init ( )
```

Initial the device by USB

It will detect the device and trying connect.

The connect status can be checked by [device_isConnected\(\)](#).

Note: after the function returns success, the function [device_setCurrentDevice\(\)](#) has to be called to set the device type.

Returns

RETURN_CODE: Values can be parsed with [device_getResponseCodeString\(\)](#)

11.5.4.15 `device_isAttached()`

```
int device_isAttached (
    int deviceType )
```

Check if the device is attached to the USB port The function [device_init\(\)](#) must be called before this function.

Parameters

<i>deviceType</i> , the	device type of the USB device
-------------------------	-------------------------------

Returns

1 if the device is attached, or 0 if the device is not attached

11.5.4.16 device_isConnected()

```
int device_isConnected ( )
```

Check the device connected status

Returns

DEVICE_DISCONNECT=0, or DEVICE_CONNECTED = 1

11.5.4.17 device_rebootDevice()

```
int 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\(\)](#)

11.5.4.18 device_registerCameraCallBk()

```
void device_registerCameraCallBk (
    pCMR_callback pCMRf )
```

To register the camera callback function to get the image data. (Pass NULL to disable the callback.)

11.5.4.19 device_registerCardStatusFrontSwitchCallBk()

```
void device_registerCardStatusFrontSwitchCallBk (
    pCSFS_callback pCSFSf )
```

To register the card status and front switch callback function to get status. (Pass NULL to disable the callback.)

11.5.4.20 device_registerFWCallBk()

```
void device_registerFWCallBk (
    pFW_callback pFWf )
```

To register the firmware update callback function to get the status of firmware update. (Pass NULL to disable the callback.)

11.5.4.21 device_SendDataCommand()

```
int device_SendDataCommand (
    IN BYTE * cmd,
    IN int cmdLen,
    IN BYTE * data,
    IN int dataLen,
    OUT BYTE * response,
    IN_OUT int * respLen )
```

Send a Command to device

Sends a command to the device .

Parameters

<i>cmd</i>	buffer of command to execute.
<i>cmdLen,the</i>	length of the buffer cmd.
<i>data</i>	buffer of IDG command data.
<i>dataLen,the</i>	length of the buffer data.
<i>response</i>	Response data
<i>respLen,the</i>	length of Response data

Returns

RETURN_CODE: Values can be parsed with [device_getResponseCodeString\(\)](#)

11.5.4.22 device_setCurrentDevice()

```
int device_setCurrentDevice (
    int deviceType )
```

Sets the current device to talk to

The connect status can be checked by [device_isConnected\(\)](#).

Parameters

<i>deviceType</i>	<p>Device to connect to</p> <pre>enum DEVICE_TYPE { IDT_DEVICE_UNKNOWN=0, IDT_DEVICE_AUGUSTA_HID, IDT_DEVICE_AUGUSTA_KB, IDT_DEVICE_AUGUSTA_S_HID, IDT_DEVICE_AUGUSTA_S_KB, IDT_DEVICE_AUGUSTA_S_TTK_HID, IDT_DEVICE_SPECTRUM_PRO, IDT_DEVICE_MINISMART_II, IDT_DEVICE_L80, IDT_DEVICE_L100, IDT_DEVICE_UNIPAY, IDT_DEVICE_UNIPAY_I_V, IDT_DEVICE_VP3300_AJ, IDT_DEVICE_KIOSK_III, IDT_DEVICE_KIOSK_III_S, IDT_DEVICE_PIP_READER, IDT_DEVICE_VENDI, IDT_DEVICE_VP3300_USB, IDT_DEVICE_UNIPAY_I_V_TTK, IDT_DEVICE_VP3300_BT, IDT_DEVICE_VP8800, IDT_DEVICE_SREDKEY2_HID, IDT_DEVICE_SREDKEY2_KB, IDT_DEVICE_NEO2, IDT_DEVICE_MINISMART_II_COM = IDT_DEVICE_NEO2+5, IDT_DEVICE_SPECTRUM_PRO_COM, IDT_DEVICE_KIOSK_III_COM, IDT_DEVICE_KIOSK_III_S_COM, IDT_DEVICE_PIP_READER_COM, IDT_DEVICE_VP3300_COM, IDT_DEVICE_NEO2_COM, IDT_DEVICE_MAX_DEVICES = IDT_DEVICE_NEO2_COM+5 };</pre>
-------------------	---

Returns

RETURN_CODE: 1: success, 0: failed

11.5.4.23 device_setSleepModeTime()

```
int device_setSleepModeTime (
    int time )
```

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.
-------------	------------------------------------

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString

11.5.4.24 device_updateFirmware()

```
int device_updateFirmware (
    IN BYTE * firmwareData,
    IN int firmwareDataLen,
    IN char * firmwareName,
    IN int encryptionType,
    IN BYTE * keyBlob,
    IN int keyBlobLen )
```

Update Firmware Updates the firmware of Augusta.

Parameters

<i>firmwareData</i>	Signed binary data of a firmware file provided by IDTech
<i>firmwareDataLen</i>	Length of firmwareData
<i>firmwareName</i>	Firmware name. <ul style="list-style-type: none"> For example "Augusta_S_TTK_V1.00.002.fm"
<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)
<i>keyBlobLen</i>	Length of keyBlob

Returns

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

Firmware update status is returned in the callback with the following values: sender = AUGUSTA state = Device↔ State.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

11.5.4.25 emv_registerCallBk()

```
void emv_registerCallBk (
    pEMV_callback pEMVf )
```

To register the emv callback function to get the EMV processing response. (Pass NULL to disable the callback.)

11.5.4.26 lcd_displayMessage()

```
int lcd_displayMessage (
    int lineNumber,
    char * message,
    int messageLen )
```

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>messageLen</i>	length of message

Returns

RETURN_CODE: Values can be parsed with `device_getResponseCodeString`

11.5.4.27 lcd_displayPrompt()

```
int lcd_displayPrompt (
    int promptNumber,
    int lineNumber )
```

Display Prompt on Line

Displays a message prompt from L80 or L100 memory.

Parameters

<i>promptNumber</i>	Prompt number (0-9)
---------------------	---------------------

Parameters

<i>lineNumber</i>	Line number to display message prompt (1-4)
-------------------	---

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString

11.5.4.28 lcd_enableBacklight()

```
int lcd_enableBacklight (
    int enable )
```

Enable/Disable LCD Backlight

Turns on/off the LCD back lighting.

Parameters

<i>enable</i>	TRUE = turn ON backlight, FALSE = turn OFF backlight
---------------	--

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString

11.5.4.29 lcd_getBacklightStatus()

```
int lcd_getBacklightStatus (
    int * enabled )
```

Get Backlight Status

Returns the status of the LCD back lighting.

Parameters

<i>enabled</i>	1 = Backlight is ON, 0 = Backlight is OFF
----------------	---

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString

11.5.4.30 lcd_savePrompt()

```
int lcd_savePrompt (
    int promptNumber,
    char * prompt,
    int promptLen )
```

Save Prompt

Saves a message prompt to L80 or L100 memory.

Parameters

<i>promptNumber</i>	Prompt number (0-9)
<i>prompt</i>	Prompt string (up to 20 characters)
<i>promptLen</i>	length of prompt

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString

11.5.4.31 msr_registerCallBk()

```
void msr_registerCallBk (
    pMSR_callback pMSRf )
```

To register the msr callback function to get the MSR card data. (Pass NULL to disable the callback.)

11.5.4.32 msr_registerCallBkp()

```
void msr_registerCallBkp (
    pMSR_callbackp pMSRf )
```

To register the msr callback function to get the MSR card data pointer. (Pass NULL to disable the callback.)

11.5.4.33 pin_getEncryptedPIN()

```
int pin_getEncryptedPIN (
    int keyType,
    char * PAN,
    int PANLen,
    char * message,
    int messageLen,
    int timeout )
```

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>PANLen</i>	length of PAN
<i>message</i>	Message to display
<i>messageLen</i>	length of message
<i>timeout</i>	PIN entry timeout

Returns

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

11.5.4.34 pin_getFunctionKey()

```
int pin_getFunctionKey (
    int timeout )
```

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

@param timeout Timeout, in seconds

Returns

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

11.5.4.35 pin_promptForAmountInput()

```
int pin_promptForAmountInput (
    int messageID,
    int languageID,
    int minLen,
    int maxLen,
    int timeout )
```

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 INGR↵ESE	SVP ENTREZ
6	PLEASE REENTER	POR FAVO REENT↵RAR	POR FAVO REINGR↵ESE	SVP RE-ENTREZ
7	PO NUMBER	N筋ERO PO	NUMERO PO	No COMMANDE
8	DRIVER ID	LICEN斤墩	LICENCIA	ID CONDUCTEUR
9	ODOMETER	ODOMETER	ODOMETRO	ODOMETRE

Msg Id	English Prompt	Portuguese Prompt	Spanish Prompt	French Prompt
10	ID NUMBER	N ^o 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 N ^o	NUMERO EMP	No TRAVAIL
14	WORK ORDER	TRABALHO ORDEM	ORDEN TRABAJO	FICHE TRAVAIL
15	VEHICLE ID	ID VE ^h ULO	ID VEHICULO	ID VEHICULE
16	ENTER DRIVER	ENTER DRIVER	INGRESE CONDUCTOR	ENTR CONDUCTEUR
17	ENTER DEPT	ENTER DEPT	INGRESE DEPT	ENTR DEPARTEMENT
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	N ^o PATH	RUTA NUMERO	No ROUTE
23	ENTER USER ID	ENTER USER ID	INGRESE ID USUARIO	ID UTILISATEUR
24	FLEET NUMBER	N ^o DE FROTA	FLOTA NUMERO	No PARC AUTO
25	ENTER PRODUCT	ADICIONAR PRODUCTO	INGRESE PRODUCTO	ENTREZ PRODUIT
26	DRIVER NUMBER	N ^o DRIVER	CONDUCTOR NUMERO	No CONDUCTEUR
27	ENTER LICENSE	ENTER LICEN ^{ça}	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 VE ^h ULO	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	N ^o DO VE ^h ULO	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 USUARIO	INGRESE INFO USUARIO	INFO UTILISATEUR
39	ENTER CUST CODE	ENTER CODE. CLIENTE	INGRESE COD. CLIENTE	ENTR CODE CLIENT
40	ENTER EMPLOYEE	ENTER FUNCIONARIO	INGRESE EMPLEADO	ENTREZ EMPLOYE
41	ENTER ID NUMBER	ENTER N ^o 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

Msg Id	English Prompt	Portuguese Prompt	Spanish Prompt	French Prompt
44	ODOMETER NUMB↵ ER	N↵ERO ODOMETER	ODOMETRO NUME↵ RO	No ODOMETRE
45	ENTER DRIVER LIC	ENTER DRIVER LIC	INGRESE LIC CON↵ DUCTOR	PERMIS CONDUIRE
46	ENTER TRAILER NO	NRO TRAILER ENT↵ ER	INGRESE NRO TRA↵ ILER	ENT No REMORQUE
47	REENTER VEHICLE	REENTRAR VE↵U↵ LO	REINGRESE VEHIC↵ ULO	RE-ENTR VEHICULE
48	ENTER VEHICLE ID	ENTER VE↵ULO ID	INGRESE ID VEHIC↵ ULO	ENTR ID VEHICULE
49	ENTER BIRTH DATE	INSERIR DATA NAC	INGRESE FECHA N↵ AC	ENT DT NAISSANCE
50	ENTER DOB MMDD↵ YY	ENTER FDN MMDD↵ YY	INGRESE FDN MM↵ DDAA	NAISSANCE MMJJAA
51	ENTER FLEET DATA	ENTER FROTA INFO	INGRESE INFO DE FLOTA	INFO PARC AUTO
52	ENTER REFERENCE	ENTER REFER ↵ CIA	INGRESE REFERE↵ NCIA	ENTREZ REFEREN↵ CE
53	ENTER AUTH NUM↵ BR	ENTER N↵ERO AUT	INGRESE NUMERO AUT	No AUTORISATION
54	ENTER HUB NUMB↵ ER	ENTER HUB NRO	INGRESE NRO HUB	ENTREZ No NOYAU
55	ENTER HUBOMETER	MEDIDA PARA ENT↵ RAR HUB	INGRESE MEDIDO DE HUB	COMPTEUR NOYAU
56	ENTER TRAILER ID	TRAILER ENTER ID	INGRESE ID TRAIL↵ ER	ENT ID REMORQUE
57	ODOMETER READI↵ NG	QUILOMETRAGEM	LECTURA ODOME↵ TRO	LECTURE ODOME↵ TRE
58	REENTER ODOME↵ TER	REENTRAR ODOM↵ ETER	REINGRESE ODO↵ METRO	RE-ENT ODOMETRE
59	REENTER DRIV. ID	REENTRAR DRIVER ID	REINGRESE ID CH↵ OFER	RE-ENT ID CONDUC
60	ENTER CUSTOMER ID	ENTER CLIENTE ID	INGRESE ID CLIEN↵ TE	ENTREZ ID CLIENT
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 # CONDUCTE↵ UR
67	VEHICLE #	VE↵ULO #	VEHICULO #	# VEHICULE
68	ENTER VEHICLE #	ENTER VE↵ULO #	INGRESE VEHICULO #	ENT # VEHICULE
69	JOB #	TRABALHO #	TRABAJO #	# TRAVAIL
70	ENTER JOB #	ENTER JOB #	INGRESE TRABAJO #	ENTREZ # TRAVAIL
71	DEPT NUMBER	N↵ERO DEPT	NUMERO DEPTO	No DEPARTEMENT
72	DEPARTMENT #	DEPARTAMENTO #	DEPARTAMENTO #	DEPARTEMENT
73	ENTER DEPT #	ENTER DEPT #	INGRESE DEPTO #	ENT# DEPARTEME↵ NT

Msg Id	English Prompt	Portuguese Prompt	Spanish Prompt	French Prompt
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(YYYMM)	VALIDADE VAL (AAAA-MM)	FECHA EXP (AAYMM)	DATE EXPIR(AAYMM)
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 EMISIÓN	NO DEMISSION
90	START DATE (MMYY)	DATA DE INÍCIO (AAYMM)	FECHA INICIO (AAYMM)	DATE DE DEBUT-AAYMM

```

@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
@param timeout Timeout value, in seconds

```

Returns

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

11.5.4.36 pin_promptForKeyInput()

```

int pin_promptForKeyInput (
    int messageID,
    int languageID,
    int maskInput,
    int minLen,
    int maxLen,
    int timeout )

```

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

Msg Id	English Prompt	Portuguese Prompt	Spanish Prompt	French Prompt
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 INGR↵ESE	SVP ENTREZ
6	PLEASE REENTER	POR FAVO REENT↵RAR	POR FAVO REINGR↵ESE	SVP RE-ENTREZ
7	PO NUMBER	N↵ERO PO	NUMERO PO	No COMMANDE
8	DRIVER ID	LICEN斤墩	LICENCIA	ID CONDUCTEUR
9	ODOMETER	ODOMETER	ODOMETRO	ODOMETRE
10	ID NUMBER	N↵ERO 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 N↵ERO	NUMERO EMP	No TRAVAIL
14	WORK ORDER	TRABALHO ORDEM	ORDEN TRABAJO	FICHE TRAVAIL
15	VEHICLE ID	ID VE斤ULO	ID VEHICULO	ID VEHICULE
16	ENTER DRIVER	ENTER DRIVER	INGRESE CONDUC↵TOR	ENTR CONDUCTEUR
17	ENTER DEPT	ENTER DEPT	INGRESE DEPT	ENTR DEPARTEMNT
18	ENTER PHONE	ADICIONAR PHONE	INGRESE TELEFO↵NO	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 TRAB↵AJO	ENTR ID TRAVAIL
22	ROUTE NUMBER	N↵ERO PATH	RUTA NUMERO	No ROUTE
23	ENTER USER ID	ENTER USER ID	INGRESE ID USUA↵RIO	ID UTILISATEUR
24	FLEET NUMBER	N↵ERO DE FROTA	FLOTA NUMERO	No PARC AUTO
25	ENTER PRODUCT	ADICIONAR PROD↵UTO	INGRESE PRODUC↵TO	ENTREZ PRODUIT
26	DRIVER NUMBER	N↵ERO DRIVER	CONDUCTOR NUM↵ERO	No CONDUCTEUR
27	ENTER LICENSE	ENTER LICEN斤墩	INGRESE LICENCIA	ENTREZ PERMIS
28	ENTER FLEET NO	ENTER NRO FROTA	INGRESE NRO FLO↵TA	ENT No PARC AUTO
29	ENTER CAR WASH	WASH ENTER	INGRESE LAVADO	ENTREZ LAVE-AUTO
30	ENTER VEHICLE	ENTER VE斤ULO	INGRESE VEHICULO	ENTREZ VEHICULE
31	ENTER TRAILER	TRAILER ENTER	INGRESE TRAILER	ENTREZ REMORQ↵UE
32	ENTER ODOMETER	ENTER ODOMETER	INGRESE ODOMET↵RO	ENTREZ ODOMETRE
33	DRIVER LICENSE	CARTEIRA DE MOT↵ORISTA	LICENCIA CONDU↵CTOR	PERMIS CONDUIRE
34	ENTER CUSTOMER	ENTER CLIENTE	INGRESE CLIENTE	ENTREZ CLIENT
35	VEHICLE NUMBER	N↵ERO DO VE斤U↵LO	VEHICULO NUMERO	No VEHICULE
36	ENTER CUST DATA	ENTER CLIENTE IN↵FO	INGRESE INFO CLI↵ENTE	INFO CLIENT
37	REENTER DRIVID	REENTRAR DRIVER ID	REINGRESE ID CH↵OFER	RE-ENTR ID COND
38	ENTER USER DATA	ENTER INFO USU↵辣IO	INGRESE INFO US↵UARIO	INFO UTILISATEUR

Msg Id	English Prompt	Portuguese Prompt	Spanish Prompt	French Prompt
39	ENTER CUST CODE	ENTER CODE. CLI↵ ENTE	INGRESE COD. CLI↵ ENTE	ENTR CODE CLIENT
40	ENTER EMPLOYEE	ENTER FUNCION 筋 糸IO	INGRESE EMPLEA↵ DO	ENTREZ EMPLOYE
41	ENTER ID NUMBER	ENTER N筋ERO ID	INGRESE NUMERO ID	ENTREZ No ID
42	ENTER DRIVER ID	ENTER ID DRIVER	INGRESE ID COND↵ UCTOR	No CONDUCTEUR
43	ENTER FLEET PIN	ENTER PIN FROTA	INGRESE PIN DE F↵ LOTA	NIP PARC AUTO
44	ODOMETER NUMB↵ ER	N筋ERO ODOMETER	ODOMETRO NUME↵ RO	No ODOMETRE
45	ENTER DRIVER LIC	ENTER DRIVER LIC	INGRESE LIC CON↵ DUCTOR	PERMIS CONDUIRE
46	ENTER TRAILER NO	NRO TRAILER ENT↵ ER	INGRESE NRO TRA↵ ILER	ENT No REMORQUE
47	REENTER VEHICLE	REENTRAR VE斤U↵ LO	REINGRESE VEHIC↵ ULO	RE-ENTR VEHICULE
48	ENTER VEHICLE ID	ENTER VE斤ULO ID	INGRESE ID VEHIC↵ ULO	ENTR ID VEHICULE
49	ENTER BIRTH DATE	INSERIR DATA NAC	INGRESE FECHA N↵ AC	ENT DT NAISSANCE
50	ENTER DOB MMDD↵ YY	ENTER FDN MMDD↵ YY	INGRESE FDN MM↵ DDAA	NAISSANCE MMJJAA
51	ENTER FLEET DATA	ENTER FROTA INFO	INGRESE INFO DE FLOTA	INFO PARC AUTO
52	ENTER REFERENCE	ENTER REFER 斤 憐CIA	INGRESE REFERE↵ NCIA	ENTREZ REFEREN↵ CE
53	ENTER AUTH NUM↵ BR	ENTER N筋ERO AUT	INGRESE NUMERO AUT	No AUTORISATION
54	ENTER HUB NUMB↵ ER	ENTER HUB NRO	INGRESE NRO HUB	ENTREZ No NOYAU
55	ENTER HUBOMETER	MEDIDA PARA ENT↵ RAR HUB	INGRESE MEDIDO DE HUB	COMPTEUR NOYAU
56	ENTER TRAILER ID	TRAILER ENTER ID	INGRESE ID TRAIL↵ ER	ENT ID REMORQUE
57	ODOMETER READI↵ NG	QUILOMETRAGEM	LECTURA ODOME↵ TRO	LECTURE ODOME↵ TRE
58	REENTER ODOME↵ TER	REENTRAR ODOM↵ ETER	REINGRESE ODO↵ METRO	RE-ENT ODOMETRE
59	REENTER DRIV. ID	REENTRAR DRIVER ID	REINGRESE ID CH↵ OFER	RE-ENT ID CONDUC
60	ENTER CUSTOMER ID	ENTER CLIENTE ID	INGRESE ID CLIEN↵ TE	ENTREZ ID CLIENT
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 CONDUC↵ TOR #	ENT # CONDUCTE↵ UR
67	VEHICLE #	VE斤ULO #	VEHICULO #	# VEHICULE

Msg Id	English Prompt	Portuguese Prompt	Spanish Prompt	French Prompt
68	ENTER VEHICLE #	ENTER VEICULO #	INGRESE VEHICULO #	ENT # VEHICULE
69	JOB #	TRABALHO #	TRABAJO #	# TRAVAIL
70	ENTER JOB #	ENTER JOB #	INGRESE TRABAJO #	ENTREZ # TRAVAIL
71	DEPT NUMBER	NUMERO DEPT	NUMERO DEPTO	No DEPARTEMENT
72	DEPARTMENT #	DEPARTAMENTO #	DEPARTAMENTO #	DEPARTEMENT
73	ENTER DEPT #	ENTER DEPT #	INGRESE DEPTO #	ENT# DEPARTEMENT
74	LICENSE NUMBER	NUMERO DE LICENCA	NUMERO LICENCIA	No PERMIS
75	LICENSE #	LICENCA #	LICENCIA #	# PERMIS
76	ENTER LICENSE #	ENTER LICENCA #	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 NUMERO DE CARTA	INGRESE NUM TARCJETA	ENTREZ NO CARTE
86	EXP DATE(Yymm)	VALIDADE VAL (Aamm)	FECHA EXP (Aamm)	DATE EXPIR(Aamm)
87	PHONE NUMBER	TELEFONE	NUMERO TELEFONO	NO TEL
88	CVV START DATE	CVV DATA DE INICIO	CVV FECHA INICIO	CVV DATE DE DEBUT
89	ISSUE NUMBER	NUMERO DE EMISSAO	NUMERO DE EMISION	NO DEMISSION
90	START DATE (MmYy)	DATA DE INICIO (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 maskInput 1 = entry is masked with '*', 0 = 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
@param timeout Timeout value, in seconds

```

Returns

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

11.5.4.37 pin_registerCallBk()

```

void pin_registerCallBk (
    pPIN_callBack pPINf )

```

To register the pin callback function to get the PINPad data. (Pass NULL to disable the callback.)

11.5.4.38 pin_sendBeep()

```
int pin_sendBeep (
    int frequency,
    int duration )
```

Send Beep

Executes a beep request.

Parameters

<i>frequency</i>	Frequency, range 200-20000Hz Not used for NEO 2 devices
<i>duration</i>	Duration, range 16-65535ms Not used for NEO 2 devices

Returns

RETURN_CODE: Values can be parsed with `device_getResponseCodeString`

11.5.4.39 pin_setKeyValues()

```
int pin_setKeyValues (
    int mode )
```

Set Key Values

Set return key values on or off

Parameters

<i>mode</i>	On: 1, Off: 0
-------------	---------------

Returns

RETURN_CODE: Values can be parsed with [device_getResponseCodeString\(\)](#)

11.5.4.40 registerHotplugCallBk()

```
void registerHotplugCallBk (
    pMessageHotplug pMsgHotplug )
```

To register the USB HID hot-plug callback function which implemented in the application to monitor the hotplug message from the SDK.

11.5.4.41 registerLogCallBk()

```
void registerLogCallBk (
    pSendDataLog pFSend,
    pReadDataLog pFRead )
```

To register the log callback function which implemented in the application to monitor sending/reading data between application and reader.

11.5.4.42 SDK_Version()

```
char* SDK_Version ( )
```

To Get SDK version

Returns

return the SDK version string

11.5.4.43 setAbsoluteLibraryPath()

```
int setAbsoluteLibraryPath (
    const char * absoluteLibraryPath )
```

Set the path to use when searching for ID TECH's libraries. If this is not set, the libraries will be searched for with the system's default procedures.

Parameters

<i>absoluteLibraryPath</i>	The absolute path to ID TECH's libraries.
----------------------------	---

Returns

RETURN_CODE: Values can be parsed with [device_getResponseCodeString\(\)](#)

11.6 Source_C/libIDT_MiniSmartII.h File Reference

MiniSmartII API.

```
#include "IDTDef.h"
```

Macros

- `#define IN`
- `#define OUT`
- `#define IN_OUT`

Typedefs

- `typedef void(* pMessageHotplug) (int, int)`
- `typedef void(* pSendDataLog) (BYTE *, int)`
- `typedef void(* pReadDataLog) (BYTE *, int)`
- `typedef void(* pEMV_callBack) (int, int, BYTE *, int, IDTTransactionData *, EMV_Callback *, int)`
- `typedef void(* pMSR_callBack) (int, IDTMSRData)`
- `typedef void(* pMSR_callBackp) (int, IDTMSRData *)`
- `typedef void(* pPIN_callBack) (int, IDTPINData *)`
- `typedef void(* pCMR_callBack) (int, IDTCMRData *)`
- `typedef void(* pCSFS_callBack) (BYTE status)`
- `typedef void(* ftpComm_callBack) (int, int, int)`
- `typedef void(* httpComm_callBack) (BYTE *, int)`
- `typedef void(* v4Comm_callBack) (BYTE, BYTE, BYTE *, int)`

Functions

- void [registerHotplugCallBk](#) (pMessageHotplug pMsgHotplug)
- void [registerLogCallBk](#) (pSendDataLog pFSend, [pReadDataLog](#) pFRead)
- void [emv_registerCallBk](#) (pEMV_callBack pEMVf)
- void [msr_registerCallBk](#) (pMSR_callBack pMSRf)
- void [msr_registerCallBkp](#) (pMSR_callBackp pMSRf)
- void [pin_registerCallBk](#) (pPIN_callBack pPINf)
- void [device_registerCameraCallBk](#) (pCMR_callBack pCMRf)
- void [device_registerCardStatusFrontSwitchCallBk](#) (pCSFS_callBack pCSFSf)
- void [comm_registerHTTPCallback](#) (httpComm_callBack cBack)
- void [comm_registerV4Callback](#) (v4Comm_callBack cBack)
- char * [SDK_Version](#) ()
- int [setAbsoluteLibraryPath](#) (const char *absoluteLibraryPath)
- int [device_init](#) ()
- int [rs232_device_init](#) (int deviceType, int port_number, int brate)
- int [device_setCurrentDevice](#) (int deviceType)
- int [device_close](#) ()
- void [device_getResponseCodeString](#) (IN int returnCode, OUT char *despcriton)
- int [device_isConnected](#) ()
- int [device_isAttached](#) (int deviceType)
- int [device_getFirmwareVersion](#) (OUT char *firmwareVersion)
- int [device_getFirmwareVersion_Len](#) (OUT char *firmwareVersion, IN_OUT int *firmwareVersionLen)
- int [device_getCurrentDeviceType](#) ()
- int [device_SendDataCommand](#) (IN BYTE *cmd, IN int cmdLen, IN BYTE *data, IN int dataLen, OUT BYTE *response, IN_OUT int *respLen)
- int [device_updateFirmware](#) (IN BYTE *firmwareData, IN int firmwareDataLen, IN char *firmwareName, IN int encryptionType, IN BYTE *keyBlob, IN int keyBlobLen)
- int [device_rebootDevice](#) ()
- int [device_controlLED](#) (byte indexLED, byte control, int intervalOn, int intervalOff)
- int [device_controlLED_ICC](#) (int controlMode, int interval)
- int [device_controlLED_MSR](#) (byte control, int intervalOn, int intervalOff)
- int [device_controlBeep](#) (int index, int frequency, int duration)
- int [device_getKeyStatus](#) (int *newFormat, BYTE *status, int *statusLen)
- int [device_getSDKWaitTime](#) ()
- void [device_setSDKWaitTime](#) (int waitTime)
- int [device_getThreadStackSize](#) ()
- void [device_setThreadStackSize](#) (int threadSize)
- int [config_getModelNumber](#) (OUT char *sNumber)
- int [config_getModelNumber_Len](#) (OUT char *sNumber, IN_OUT int *sNumberLen)
- int [config_getSerialNumber](#) (OUT char *sNumber)
- int [config_getSerialNumber_Len](#) (OUT char *sNumber, IN_OUT int *sNumberLen)
- int [config_setLEDController](#) (int firmwareControlMSRLED, int firmwareControlICCLEd)
- int [config_getLEDController](#) (int *firmwareControlMSRLED, int *firmwareControlICCLEd)
- int [config_setBeeperController](#) (int firmwareControlBeeper)
- int [config_getBeeperController](#) (int *firmwareControlBeeper)
- int [config_setEncryptionControl](#) (int msr, int icc)
- int [config_getEncryptionControl](#) (int *msr, int *icc)
- int [icc_enable](#) (IN int withNotification)
- int [icc_disable](#) ()
- int [icc_powerOnICC](#) (OUT BYTE *ATR, IN_OUT int *inLen)
- int [icc_powerOffICC](#) ()
- int [icc_exchangeAPDU](#) (IN BYTE *c_APDU, IN int cLen, OUT BYTE *reData, IN_OUT int *reLen)
- int [icc_exchangeEncryptedAPDU](#) (IN BYTE *c_APDU, IN int cLen, OUT BYTE *reData, IN_OUT int *reLen)
- int [icc_getAPDU_KSN](#) (OUT BYTE *KSN, IN_OUT int *inLen)

- int [icc_getFunctionStatus](#) (OUT int *enabled, OUT int *withNotification)
- int [icc_getICCRReaderStatus](#) (OUT BYTE *status)
- int [icc_getKeyFormatForICCDUKPT](#) (OUT BYTE *format)
- int [icc_getKeyTypeForICCDUKPT](#) (OUT BYTE *type)
- int [emv_getEMVKernelVersion](#) (OUT char *version)
- int [emv_getEMVKernelVersion_Len](#) (OUT char *version, IN_OUT int *versionLen)
- int [emv_getEMVKernelCheckValue](#) (OUT BYTE *checkValue, IN_OUT int *checkValueLen)
- int [emv_getEMVConfigurationCheckValue](#) (OUT BYTE *checkValue, IN_OUT int *checkValueLen)
- void [emv_setAutoAuthenticateTransaction](#) (IN int authenticate)
- void [emv_setAutoCompleteTransaction](#) (IN int complete)
- int [emv_getAutoAuthenticateTransaction](#) ()
- int [emv_getAutoCompleteTransaction](#) ()
- void [emv_allowFallback](#) (IN int allow)
- int [emv_startTransaction](#) (IN double amount, IN double amtOther, IN int exponent, IN int type, IN int timeout, IN BYTE *tags, IN int tagsLen, IN int forceOnline)
- void [device_setTransactionExponent](#) (int exponent)
- int [device_startTransaction](#) (IN double amount, IN double amtOther, IN int type, IN const int _timeout, IN BYTE *tags, IN int tagsLen)
- int [emv_activateTransaction](#) (IN int timeout, IN BYTE *tags, IN int tagsLen, IN int forceOnline)
- int [emv_authenticateTransaction](#) (IN BYTE *updatedTLV, IN int updatedTLVLen)
- int [emv_authenticateTransactionWithTimeout](#) (IN int timeout, IN BYTE *updatedTLV, IN int updatedTLVLen)
- int [emv_completeTransaction](#) (IN int commError, IN BYTE *authCode, IN int authCodeLen, IN BYTE *iad, IN int iadLen, IN BYTE *tlvScripts, IN int tlvScriptsLen, IN BYTE *tlv, IN int tlvLen)
- int [emv_cancelTransaction](#) ()
- int [device_cancelTransaction](#) ()
- int [emv_retrieveTransactionResult](#) (IN BYTE *tags, IN int tagsLen, IDTTransactionData *cardData)
- int [emv_callbackResponseLCD](#) (IN int type, byte selection)
- int [emv_callbackResponseMSR](#) (IN BYTE *MSR, IN_OUT int MSRLen)
- int [emv_retrieveApplicationData](#) (IN BYTE *AID, IN int AIDLen, OUT BYTE *tlv, IN_OUT int *tlvLen)
- int [emv_setApplicationData](#) (IN BYTE *name, IN int nameLen, IN BYTE *tlv, IN int tlvLen)
- int [emv_removeApplicationData](#) (IN BYTE *AID, IN int AIDLen)
- int [emv_removeAllApplicationData](#) ()
- int [emv_retrieveAIDList](#) (OUT BYTE *AIDList, IN_OUT int *AIDListLen)
- int [emv_retrieveTerminalData](#) (OUT BYTE *tlv, IN_OUT int *tlvLen)
- int [emv_setTerminalData](#) (IN BYTE *tlv, IN int tlvLen)
- int [emv_removeTerminalData](#) ()
- int [emv_retrieveCAPK](#) (IN BYTE *capk, IN int capkLen, OUT BYTE *key, IN_OUT int *keyLen)
- int [emv_setCAPK](#) (IN BYTE *capk, IN int capkLen)
- int [emv_removeCAPK](#) (IN BYTE *capk, IN int capkLen)
- int [emv_removeAllCAPK](#) ()
- int [emv_retrieveCAPKList](#) (OUT BYTE *keys, IN_OUT int *keysLen)
- int [emv_retrieveTerminalID](#) (OUT char *terminalID)
- int [emv_retrieveTerminalID_Len](#) (OUT char *terminalID, IN_OUT int *terminalIDLen)
- int [emv_setTerminalID](#) (IN char *terminalID)
- int [emv_retrieveCRL](#) (OUT BYTE *list, IN_OUT int *lssLen)
- int [emv_setCRL](#) (IN BYTE *list, IN int lssLen)
- int [emv_removeCRL](#) (IN BYTE *list, IN int lssLen)
- int [emv_removeAllCRL](#) ()

11.6.1 Detailed Description

MiniSmartII API.

MiniSmartII Global API methods.

11.6.2 Macro Definition Documentation

11.6.2.1 IN

```
#define IN
```

INPUT parameter.

11.6.2.2 IN_OUT

```
#define IN_OUT
```

INPUT / OUTPUT PARAMETER.

11.6.2.3 OUT

```
#define OUT
```

OUTPUT parameter.

11.6.3 Typedef Documentation

11.6.3.1 ftpComm_callBack

```
typedef void(* ftpComm_callBack) (int, int, int)
```

Define the comm callback function to get FTP file transfer status

It should be passed as a parameter in a FTP request, Signature (int, int, int) = response code, current block, total blocks
 RESPONSE CODES: 100 = FILE DOWNLOAD STARTED 101 = FILE BLOCK XX OF XX RECEIVED 102 = FILE DOWNLOAD COMPLETED 103 = FILE DOWNLOAD TERMINATED PREMATURELY

11.6.3.2 httpComm_callBack

```
typedef void(* httpComm_callBack) (BYTE *, int)
```

Define the comm callback function to get the async url data

It should be registered using the comm_registerHTTPCallback

11.6.3.3 pCMR_callBack

```
typedef void(* pCMR_callBack) (int, IDTCMRData *)
```

Define the camera callback function to get the image data

It should be registered using the device_registerCameraCallBk,

11.6.3.4 pCSFS_callBack

```
typedef void(* pCSFS_callBack) (BYTE status)
```

Define the card status and front switch callback function to get card and front switch status

It should be registered using the device_registerCardStatusFrontSwitchCallBk,

11.6.3.5 pEMV_callBack

```
typedef void(* pEMV_callBack) (int, int, BYTE *, int, IDTTransactionData *, EMV_Callback *, int)
```

Define the EMV callback function to get the transaction message/data/result.
It should be registered using the `emv_registerCallBk`,

11.6.3.6 pMessageHotplug

```
typedef void(* pMessageHotplug) (int, int)
```

Define the USB hot-plug callback function to monitor the info when plug in/out the reader.
It should be registered using the `registerHotplugCallBk`, The first integer parameter is device type, and the second integer parameter is either 0: Device Plugged Out or 1: Device Plugged In

11.6.3.7 pMSR_callBack

```
typedef void(* pMSR_callBack) (int, IDTMSRData)
```

Define the MSR callback function to get the MSR card data
It should be registered using the `msr_registerCallBk`, this callback function is for backward compatibility

11.6.3.8 pMSR_callBackp

```
typedef void(* pMSR_callBackp) (int, IDTMSRData *)
```

Define the MSR callback function to get pointer to the MSR card data
It should be registered using the `msr_registerCallBk`, this callback function is recommended instead of `pMSR_callBack`

11.6.3.9 pPIN_callBack

```
typedef void(* pPIN_callBack) (int, IDTPINData *)
```

Define the PINPad callback function to get the input PIN Pad data
It should be registered using the `pin_registerCallBk`,

11.6.3.10 pReadDataLog

```
typedef void(* pReadDataLog) (BYTE *, int)
```

Define the read response callback function to monitor the reading response from the reader.
It should be registered using the `registerLogCallBk`,

11.6.3.11 pSendDataLog

```
typedef void(* pSendDataLog) (BYTE *, int)
```

Define the send command callback function to monitor the sending command into the reader.
It should be registered using the `registerLogCallBk`,

11.6.3.12 v4Comm_callBack

```
typedef void(* v4Comm_callBack) (BYTE, BYTE, BYTE *, int)
```

Define the comm callback function to receive the V4 Protocol packets received by the device from an external source (IP/USB/RS-232) It should be registered using the `comm_registerV4Callback`, Data callback will contain command, sub-command, and data from V4 packet

11.6.4 Function Documentation

11.6.4.1 `comm_registerHTTPCallback()`

```
void comm_registerHTTPCallback (
    httpComm_callBack cBack )
```

Register Comm HTTP Async Callback

Parameters

<i>cBack</i>	- HTTP Comm callback
--------------	----------------------

11.6.4.2 `comm_registerV4Callback()`

```
void comm_registerV4Callback (
    v4Comm_callBack cBack )
```

Register External V4 Protocol commands Callback

Parameters

<i>cBack</i>	- V4 Protocol Comm callback
--------------	-----------------------------

11.6.4.3 `config_getBeeperController()`

```
int config_getBeeperController (
    int * firmwareControlBeeper )
```

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

Parameters

<i>firmwareControlBeeper</i>	1 means firmware control the beeper, 0 means software control beeper.
------------------------------	---

Returns

RETURN_CODE: Values can be parsed with `device_getResponseCodeString`

11.6.4.4 `config_getEncryptionControl()`

```
int config_getEncryptionControl (
```



```
int * msr,  
int * icc )
```

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">• 1: enabled MSR with Encryption,• 0: disabled MSR with Encryption,
<i>icc</i>	<ul style="list-style-type: none">• 1: enabled ICC with Encryption,• 0: disabled ICC with Encryption,

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString

11.6.4.5 config_getLEDController()

```
int config_getLEDController (  
    int * firmwareControlMSRLED,  
    int * firmwareControlICCLED )
```

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

Parameters

<i>firmwareControlMSRLED</i>	<ul style="list-style-type: none">• 1: firmware control the MSR LED• 0: software control the MSR LED
<i>firmwareControlICCLED</i>	<ul style="list-style-type: none">• 1: firmware control the ICC LED• 0: software control the ICC LED

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString

11.6.4.6 config_getModelNumber()

```
int config_getModelNumber (
    OUT char * sNumber )
```

DEPRECATED : please use [config_getModelNumber_Len](#)(OUT char* sNumber, IN_OUT int *sNumberLen)

Polls device for Model Number

Parameters

<i>sNumber</i>	Returns Model Number; needs to have at least 64 bytes of memory
----------------	---

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString

11.6.4.7 config_getModelNumber_Len()

```
int config_getModelNumber_Len (
    OUT char * sNumber,
    IN_OUT int * sNumberLen )
```

Polls device for Model Number

Parameters

<i>sNumber</i>	Returns Model Number
<i>sNumber</i>	length of Model Number

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString

11.6.4.8 config_getSerialNumber()

```
int config_getSerialNumber (
    OUT char * sNumber )
```

DEPRECATED : please use [config_getSerialNumber_Len](#)(OUT char* sNumber, IN_OUT int *sNumberLen)

Polls device for Serial Number

Parameters

<i>sNumber</i>	Returns Serial Number; needs to have at least 64 bytes of memory
----------------	--

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString

11.6.4.9 config_getSerialNumber_Len()

```
int config_getSerialNumber_Len (
    OUT char * sNumber,
    IN_OUT int * sNumberLen )
```

Polls device for Serial Number

Parameters

<i>sNumber</i>	Returns Serial Number
<i>sNumberLen</i>	Length of Serial Number

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString

11.6.4.10 config_setBeeperController()

```
int config_setBeeperController (
    int firmwareControlBeeper )
```

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

Parameters

<i>firmwareControlBeeper</i>	1 means firmware control the beeper, 0 means software control beeper.
------------------------------	---

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString

11.6.4.11 config_setEncryptionControl()

```
int config_setEncryptionControl (
    int msr,
    int icc )
```

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"> • 1: enable MSR with Encryption, • 0: disable MSR with Encryption,
<i>icc</i>	<ul style="list-style-type: none"> • 1: enable ICC with Encryption, • 0: disable ICC with Encryption,

Returns

RETURN_CODE: Values can be parsed with `device_getResponseCodeString`

11.6.4.12 config_setLEDController()

```
int config_setLEDController (
    int firmwareControlMSRLED,
    int firmwareControlICCLED )
```

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"> • 1: firmware control the MSR LED • 0: software control the MSR LED
<i>firmwareControlICCLED</i>	<ul style="list-style-type: none"> • 1: firmware control the ICC LED • 0: software control the ICC LED

Returns

RETURN_CODE: Values can be parsed with `device_getResponseCodeString`

11.6.4.13 device_cancelTransaction()

```
int device_cancelTransaction ( )
```

Cancel Transaction request.

Returns

RETURN_CODE: Values can be parsed with [device_getResponseCodeString\(\)](#)

11.6.4.14 device_close()

```
int device_close ( )
```

Close the device

Returns

RETURN_CODE: 0: success, 0x0A: failed

11.6.4.15 device_controlBeep()

```
int device_controlBeep (
    int index,
    int frequency,
    int duration )
```

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

11.6.4.16 device_controlLED()

```
int device_controlLED (
    byte indexLED,
    byte control,
    int intervalOn,
    int intervalOff )
```

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

11.6.4.17 device_controlLED_ICC()

```
int device_controlLED_ICC (
    int controlMode,
    int interval )
```

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

11.6.4.18 device_controlLED_MSR()

```
int device_controlLED_MSR (
    byte control,
    int intervalOn,
    int intervalOff )
```

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`

11.6.4.19 `device_getCurrentDeviceType()`

```
int device_getCurrentDeviceType ( )
```

Get current active device type

Returns

: return the device type defined as DEVICE_TYPE in the IDTDef.h

11.6.4.20 `device_getFirmwareVersion()`

```
int device_getFirmwareVersion (
    OUT char * firmwareVersion )
```

DEPRECATED : please use `device_getFirmwareVersion_Len`(OUT char* firmwareVersion, IN_OUT int *firmwareVersionLen)

Polls device for Firmware Version

Parameters

<i>firmwareVersion</i>	Response returned of Firmware Version; needs to have at least 128 bytes of memory
------------------------	---

Returns

RETURN_CODE: Values can be parsed with `device_getResponseCodeString()`

11.6.4.21 device_getFirmwareVersion_Len()

```
int device_getFirmwareVersion_Len (
    OUT char * firmwareVersion,
    IN_OUT int * firmwareVersionLen )
```

Polls device for Firmware Version

Parameters

<i>firmwareVersion</i>	Response returned of Firmware Version
<i>firmwareVersionLen</i>	Length of Firmware Version

Returns

RETURN_CODE: Values can be parsed with [device_getResponseCodeString\(\)](#)

11.6.4.22 device_getKeyStatus()

```
int device_getKeyStatus (
    int * newFormat,
    BYTE * status,
    int * statusLen )
```

Get Key Status

Gets the status of loaded keys

Parameters

<i>status</i>	newFormat for Augusta and miniSmartII only 1: new format of key status 0: reserved format for support previous device
<i>status</i>	For L80, L100, Augusta and miniSmartII: When the newFormat is 0, data format as follows. For Augusta and miniSmartII: 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 For L80 and L100: byte 0: PIN DUKPT Key byte 1: PIN Master Key byte 2: Standard PIN Session Key byte 3: Desjardins PIN Session Key byte 4: Account/MSR DUKPT Key, 1 Exists, 0 None, 0xFF STOP, Does not support, always 0 byte 5: Account/ICC DUKPT Key, 1 Exists, 0 None, 0xFF STOP, Does not support, always 0 byte 6: Admin DUKPT Key, 1 Exists, 0 None, 0xFF STOP byte 7: Data DUKPT Key, 1 Exists, 0 None, 0xFF STOP byte 8: MAC DUKPT Key, 1 Exists, 0 None, 0xFF STOP

when the newFormat is 1, 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) For NEO2 and SREDKey2: Each unit of three bytes represents one key's parameters (index and slot). Key Name Index (1 byte): 0x14 - LCL-KEK 0x01 - Pin encryption Key (NEO2 only) 0x02 - Data encryption Key 0x05 - MAC DUKPT Key 0x0A - PCI Pairing Key (NEO2 only) Key Slot (2 bytes): Indicate different slots of a certain Key Name Example: slot =5 (0x00 0x05), slot=300 (0x01 0x2C) For BTPay380, slot is always 0 For example, 0x14 0x00 0x00 0x02 0x00 0x00 0x0A 0x00 0x00 will represent [KeyNameIndex=0x14,KeySlot=0x0000], [KeyNameIndex=0x02,KeySlot=0x0000]

Slot=0x0000] and [KeyNameIndex=0x0A,KeySlot=0x0000]

Parameters

<i>statusLen</i>	the length of status
------------------	----------------------

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString

11.6.4.23 device_getResponseCodeString()

```
void device_getResponseCodeString (
    IN int returnCode,
    OUT char * description )
```

Review the return code description.

Parameters

<i>returnCode</i>	the response result.
<i>description</i>	

Return values

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

- 0: "no error, beginning task";
- 1: "no response from reader";
- 2: "invalid response data";
- 3: "time out for task or CMD";
- 4: "wrong parameter";
- 5: "SDK is doing MSR or ICC task";
- 6: "SDK is doing PINPad task";
- 7: "SDK is doing CTLS task";
- 8: "SDK is doing EMV task";
- 9: "SDK is doing Other task";
- 10: "err response or data";
- 11: "no reader attached";
- 12: "mono audio is enabled";
- 13: "did connection";
- 14: "audio volume is too low";

- 15: "task or CMD be canceled";
- 16: "UF wrong string format";
- 17: "UF file not found";
- 18: "UF wrong file format";
- 19: "Attempt to contact online host failed";
- 20: "Attempt to perform RKI failed";
- 22: "Buffer size is not enough";
- 0X300: "Key Type(TDES) of Session Key is not same as the related Master Key.";
- 0X400: "Related Key was not loaded.";
- 0X500: "Key Same.";
- 0X501: "Key is all zero";
- 0X502: "TR-31 format error";
- 0X702: "PAN is Error Key.";
- 0X705: "No Internal MSR PAN (or Internal MSR PAN is erased timeout)";
- 0X0C01: "Incorrect Frame Tag";
- 0X0C02: "Incorrect Frame Type";
- 0X0C03: "Unknown Frame Type";
- 0X0C04: "Unknown Command";
- 0X0C05: "Unknown Sub-Command";
- 0X0C06: "CRC Error";
- 0X0C07: "Failed";
- 0X0C08: "Timeout";
- 0X0C0A: "Incorrect Parameter";
- 0X0C0B: "Command Not Supported";
- 0X0C0C: "Sub-Command Not Supported";
- 0X0C0D: "Parameter Not Supported / Status Abort Command";
- 0X0C0F: "Sub-Command Not Allowed";
- 0X0D01: "Incorrect Header Tag";
- 0X0D02: "Unknown Command";
- 0X0D03: "Unknown Sub-Command";
- 0X0D04: "CRC Error in Frame";
- 0X0D05: "Incorrect Parameter";
- 0X0D06: "Parameter Not Supported";
- 0X0D07: "Mal-formatted Data";
- 0X0D08: "Timeout";
- 0X0D0A: "Failed / NACK";

- 0X0D0B: "Command not Allowed";
- 0X0D0C: "Sub-Command not Allowed";
- 0X0D0D: "Buffer Overflow (Data Length too large for reader buffer)";
- 0X0D0E: "User Interface Event";
- 0X0D11: "Communication type not supported, VT-1, burst, etc.";
- 0X0D12: "Secure interface is not functional or is in an intermediate state.";
- 0X0D13: "Data field is not mod 8";
- 0X0D14: "Pad - 0X80 not found where expected";
- 0X0D15: "Specified key type is invalid";
- 0X0D16: "Could not retrieve key from the SAM(InitSecureComm)";
- 0X0D17: "Hash code problem";
- 0X0D18: "Could not store the key into the SAM(InstallKey)";
- 0X0D19: "Frame is too large";
- 0X0D1A: "Unit powered up in authentication state but POS must resend the InitSecureComm command";
- 0X0D1B: "The EEPROM may not be initialized because SecCommInterface does not make sense";
- 0X0D1C: "Problem encoding APDU";
- 0X0D20: "Unsupported Index(ILM) SAM Transceiver error - problem communicating with the SAM(Key Mgr)";
- 0X0D22: "Unexpected Sequence Counter in multiple frames for single bitmap(ILM) Length error in data returned from the SAM(Key Mgr)";
- 0X0D22: "Improper bit map(ILM)";
- 0X0D23: "Request Online Authorization";
- 0X0D24: "ViVOCard3 raw data read successful";
- 0X0D25: "Message index not available(ILM) ViVOcomm activate transaction card type(ViVOcomm)";
- 0X0D26: "Version Information Mismatch(ILM)";
- 0X0D27: "Not sending commands in correct index message index(ILM)";
- 0X0D28: "Time out or next expected message not received(ILM)";
- 0X0D29: "ILM languages not available for viewing(ILM)";
- 0X0D2A: "Other language not supported(ILM)";
- 0X0D41: "Unknown Error from SAM";
- 0X0D42: "Invalid data detected by SAM";
- 0X0D43: "Incomplete data detected by SAM";
- 0X0D44: "Reserved";
- 0X0D45: "Invalid key hash algorithm";
- 0X0D46: "Invalid key encryption algorithm";
- 0X0D47: "Invalid modulus length";
- 0X0D48: "Invalid exponent";

- 0X0D49: "Key already exists";
- 0X0D4A: "No space for new RID";
- 0X0D4B: "Key not found";
- 0X0D4C: "Crypto not responding";
- 0X0D4D: "Crypto communication error";
- 0X0D4E: "Module-specific error for Key Manager";
- 0X0D4F: "All key slots are full (maximum number of keys has been installed)";
- 0X0D50: "Auto-Switch OK";
- 0X0D51: "Auto-Switch failed";
- 0X0D90: "Account DUKPT Key not exist";
- 0X0D91: "Account DUKPT Key KSN exhausted";
- 0X0D00: "This Key had been loaded.";
- 0X0E00: "Base Time was loaded.";
- 0X0F00: "Encryption Or Decryption Failed.";
- 0X1000: "Battery Low Warning (It is High Priority Response while Battery is Low.)";
- 0X1800: "Send 'Cancel Command' after send 'Get Encrypted PIN' & 'Get Numeric' & 'Get Amount'; - 0↵
- X1900: "Press 'Cancel' key after send 'Get Encrypted PIN' & 'Get Numeric' & 'Get Amount';
- 0X30FF: "Security Chip is not connect";
- 0X3000: "Security Chip is deactivation & Device is In Removal Legally State.";
- 0X3101: "Security Chip is activation & Device is In Removal Legally State.";
- 0X5500: "No Admin DUKPT Key.";
- 0X5501: "Admin DUKPT Key STOP.";
- 0X5502: "Admin DUKPT Key KSN is Error.";
- 0X5503: "Get Authentication Code1 Failed.";
- 0X5504: "Validate Authentication Code Error.";
- 0X5505: "Encrypt or Decrypt data failed.";
- 0X5506: "Not Support the New Key Type.";
- 0X5507: "New Key Index is Error.";
- 0X5508: "Step Error.";
- 0X5509: "KSN Error";
- 0X550A: "MAC Error.";
- 0X550B: "Key Usage Error.";
- 0X550C: "Mode Of Use Error.";
- 0X550F: "Other Error.";
- 0X6000: "Save or Config Failed / Or Read Config Error.";
- 0X6200: "No Serial Number.";

- 0X6900: "Invalid Command - Protocol is right, but task ID is invalid.";
- 0X6A01: "Unsupported Command - Protocol and task ID are right, but command is invalid - In this State";
- 0X6A00: "Unsupported Command - Protocol and task ID are right, but command is invalid.";
- 0X6B00: "Unknown parameter in command - Protocol task ID and command are right, but parameter is invalid.";
- 0X6C00: "Unknown parameter in command - Protocol task ID and command are right, but length is out of the requirement.";
- 0X7200: "Device is suspend (MKSK suspend or press password suspend).";
- 0X7300: "PIN DUKPT is STOP (21 bit 1).";
- 0X7400: "Device is Busy.";
- 0XE100: "Can not enter sleep mode";
- 0XE200: "File has existed";
- 0XE300: "File has not existed";
- 0XE313: "IO line low -- Card error after session start";
- 0XE400: "Open File Error";
- 0XE500: "SmartCard Error";
- 0XE600: "Get MSR Card data is error";
- 0XE700: "Command time out";
- 0XE800: "File read or write is error";
- 0XE900: "Active 1850 error!";
- 0XEA00: "Load bootloader error";
- 0XEF00: "Protocol Error- STX or ETX or check error.";
- 0XEB00: "Picture is not exist";
- 0X2C02: "No Microprocessor ICC seated";
- 0X2C06: "no card seated to request ATR";
- 0X2D01: "Card Not Supported,";
- 0X2D03: "Card Not Supported, wants CRC";
- 0X690D: "Command not supported on reader without ICC support";
- 0X8100: "ICC error time out on power-up";
- 0X8200: "invalid TS character received - Wrong operation step";
- 0X8300: "Decode MSR Error";
- 0X8400: "TriMagII no Response";
- 0X8500: "No Swipe MSR Card";
- 0X8510: "No Financial Card";
- 0X8600: "Unsupported F, D, or combination of F and D";
- 0X8700: "protocol not supported EMV TD1 out of range";
- 0X8800: "power not at proper level";

- 0X8900: "ATR length too long";
- 0X8B01: "EMV invalid TA1 byte value";
- 0X8B02: "EMV TB1 required";
- 0X8B03: "EMV Unsupported TB1 only 00 allowed";
- 0X8B04: "EMV Card Error, invalid BWI or CWI";
- 0X8B06: "EMV TB2 not allowed in ATR";
- 0X8B07: "EMV TC2 out of range";
- 0X8B08: "EMV TC2 out of range";
- 0X8B09: "per EMV96 TA3 must be > - 0XF";
- 0X8B10: "ICC error on power-up";
- 0X8B11: "EMV T=1 then TB3 required";
- 0X8B12: "Card Error, invalid BWI or CWI";
- 0X8B13: "Card Error, invalid BWI or CWI";
- 0X8B17: "EMV TC1/TB3 conflict-";
- 0X8B20: "EMV TD2 out of range must be T=1";
- 0X8C00: "TCK error";
- 0XA304: "connector has no voltage setting";
- 0XA305: "ICC error on power-up invalid (SBLK(IFSD) exchange";
- 0XE301: "ICC error after session start";
- 0XFF00: "Request to go online";
- 0XFF01: "EMV: Accept the offline transaction";
- 0XFF02: "EMV: Decline the offline transaction";
- 0XFF03: "EMV: Accept the online transaction";
- 0XFF04: "EMV: Decline the online transaction";
- 0XFF05: "EMV: Application may fallback to magstripe technology";
- 0XFF06: "EMV: ICC detected tah the conditions of use are not satisfied";
- 0XFF07: "EMV: ICC didn't accept transaction";
- 0XFF08: "EMV: Transaction was cancelled";
- 0XFF09: "EMV: Application was not selected by kernel or ICC format error or ICC missing data error";
- 0XFF0A: "EMV: Transaction is terminated";
- 0XFF0B: "EMV: Other EMV Error";
- 0XFFFF: "NO RESPONSE";
- 0XF002: "ICC communication timeout";
- 0XF003: "ICC communication Error";
- 0XF00F: "ICC Card Seated and Highest Priority, disable MSR work request";
- 0XF200: "AID List / Application Data is not exist";

- 0XF201: "Terminal Data is not exist";
- 0XF202: "TLV format is error";
- 0XF203: "AID List is full";
- 0XF204: "Any CA Key is not exist";
- 0XF205: "CA Key RID is not exist";
- 0XF206: "CA Key Index it not exist";
- 0XF207: "CA Key is full";
- 0XF208: "CA Key Hash Value is Error";
- 0XF209: "Transaction format error";
- 0XF20A: "The command will not be processing";
- 0XF20B: "CRL is not exist";
- 0XF20C: "CRL number exceed max number";
- 0XF20D: "Amount,Other Amount,Trasaction Type are missing";
- 0XF20E: "The Identification of algorithm is mistake";
- 0XF20F: "No Financial Card";
- 0XF210: "In Encrypt Result state, TLV total Length is greater than Max Length";
- 0X1001: "INVALID ARG";
- 0X1002: "FILE_OPEN_FAILED";
- 0X1003: "FILE OPERATION_FAILED";
- 0X2001: "MEMORY_NOT_ENOUGH";
- 0X3002: "SMARTCARD_FAIL";
- 0X3003: "SMARTCARD_INIT_FAILED";
- 0X3004: "FALLBACK_SITUATION";
- 0X3005: "SMARTCARD_ABSENT";
- 0X3006: "SMARTCARD_TIMEOUT";
- 0X3012: "EMV_RESULT_CODE_MSR_CARD_ERROR_FALLBACK";
- 0X5001: "EMV_PARSING_TAGS_FAILED";
- 0X5002: "EMV_DUPLICATE_CARD_DATA_ELEMENT";
- 0X5003: "EMV_DATA_FORMAT_INCORRECT";
- 0X5004: "EMV_NO_TERM_APP";
- 0X5005: "EMV_NO_MATCHING_APP";
- 0X5006: "EMV_MISSING_MANDATORY_OBJECT";
- 0X5007: "EMV_APP_SELECTION_RETRY";
- 0X5008: "EMV_GET_AMOUNT_ERROR";
- 0X5009: "EMV_CARD_REJECTED";
- 0X5010: "EMV_AIP_NOT_RECEIVED";

- 0X5011: "EMV_AFL_NOT_RECEIVED";
- 0X5012: "EMV_AFL_LEN_OUT_OF_RANGE";
- 0X5013: "EMV_SFI_OUT_OF_RANGE";
- 0X5014: "EMV_AFL_INCORRECT";
- 0X5015: "EMV_EXP_DATE_INCORRECT";
- 0X5016: "EMV_EFF_DATE_INCORRECT";
- 0X5017: "EMV_ISS_COD_TBL_OUT_OF_RANGE";
- 0X5018: "EMV_CRYPTOGRAM_TYPE_INCORRECT";
- 0X5019: "EMV_PSE_NOT_SUPPORTED_BY_CARD";
- 0X5020: "EMV_USER_SELECTED_LANGUAGE";
- 0X5021: "EMV_SERVICE_NOT_ALLOWED";
- 0X5022: "EMV_NO_TAG_FOUND";
- 0X5023: "EMV_CARD_BLOCKED";
- 0X5024: "EMV_LEN_INCORRECT";
- 0X5025: "CARD_COM_ERROR";
- 0X5026: "EMV_TSC_NOT_INCREASED";
- 0X5027: "EMV_HASH_INCORRECT";
- 0X5028: "EMV_NO_ARC";
- 0X5029: "EMV_INVALID_ARC";
- 0X5030: "EMV_NO_ONLINE_COMM";
- 0X5031: "TRAN_TYPE_INCORRECT";
- 0X5032: "EMV_APP_NO_SUPPORT";
- 0X5033: "EMV_APP_NOT_SELECT";
- 0X5034: "EMV_LANG_NOT_SELECT";
- 0X5035: "EMV_NO_TERM_DATA";
- 0X5039: "EMV_PIN_ENTRY_TIMEOUT";
- 0X6001: "CVM_TYPE_UNKNOWN";
- 0X6002: "CVM_AIP_NOT_SUPPORTED";
- 0X6003: "CVM_TAG_8E_MISSING";
- 0X6004: "CVM_TAG_8E_FORMAT_ERROR";
- 0X6005: "CVM_CODE_IS_NOT_SUPPORTED";
- 0X6006: "CVM_COND_CODE_IS_NOT_SUPPORTED";
- 0X6007: "NO_MORE_CVM";
- 0X6008: "PIN_BYPASSED_BEFORE";
- 0X7001: "PK_BUFFER_SIZE_TOO_BIG";
- 0X7002: "PK_FILE_WRITE_ERROR";

- 0X7003: "PK_HASH_ERROR";
- 0X8001: "NO_CARD HOLDER_CONFIRMATION";
- 0X8002: "GET_ONLINE_PIN";
- 0XD000: "Data not exist";
- 0XD001: "Data access error";
- 0XD100: "RID not exist";
- 0XD101: "RID existed";
- 0XD102: "Index not exist";
- 0XD200: "Maximum exceeded";
- 0XD201: "Hash error";
- 0XD205: "System Busy";
- 0X0E01: "Unable to go online";
- 0X0E02: "Technical Issue";
- 0X0E03: "Declined";
- 0X0E04: "Issuer Referral transaction";
- 0X0F01: "Decline the online transaction";
- 0X0F02: "Request to go online";
- 0X0F03: "Transaction is terminated";
- 0X0F05: "Application was not selected by kernel or ICC format error or ICC missing data error";
- 0X0F07: "ICC didn't accept transaction";
- 0X0F0A: "Application may fallback to magstripe technology";
- 0X0F0C: "Transaction was cancelled";
- 0X0F0D: "Timeout";
- 0X0F0F: "Other EMV Error";
- 0X0F10: "Accept the offline transaction";
- 0X0F11: "Decline the offline transaction";
- 0X0F21: "ICC detected tah the conditions of use are not satisfied";
- 0X0F22: "No app were found on card matching terminal configuration";
- 0X0F23: "Terminal file does not exist";
- 0X0F24: "CAPK file does not exist";
- 0X0F25: "CRL Entry does not exist";
- 0X0FFE: "code when blocking is disabled";
- 0X0FFF: "code when command is not applicable on the selected device";
- 0XF005: "ICC Encrypted C-APDU Data Structure Length Error Or Format Error.";
- 0XBBE0: "CM100 Success";
- 0XBBE1: "CM100 Parameter Error";

- 0XBBE2: "CM100 Low Output Buffer";
- 0XBBE3: "CM100 Card Not Found";
- 0XBBE4: "CM100 Collision Card Exists";
- 0XBBE5: "CM100 Too Many Cards Exist";
- 0XBBE6: "CM100 Saved Data Does Not Exist";
- 0XBBE8: "CM100 No Data Available";
- 0XBBE9: "CM100 Invalid CID Returned";
- 0XBBEA: "CM100 Invalid Card Exists";
- 0XBBE C: "CM100 Command Unsupported";
- 0XBBED: "CM100 Error In Command Process";
- 0XBBEE: "CM100 Invalid Command";
- 0X9031: "Unknown command";
- 0X9032: "Wrong parameter (such as the length of the command is incorrect)";
- 0X9038: "Wait (the command couldnt be finished in BWT)";
- 0X9039: "Busy (a previously command has not been finished)";
- 0X903A: "Number of retries over limit";
- 0X9040: "Invalid Manufacturing system data";
- 0X9041: "Not authenticated";
- 0X9042: "Invalid Master DUKPT Key";
- 0X9043: "Invalid MAC Key";
- 0X9044: "Reserved for future use";
- 0X9045: "Reserved for future use";
- 0X9046: "Invalid DATA DUKPT Key";
- 0X9047: "Invalid PIN Pairing DUKPT Key";
- 0X9048: "Invalid DATA Pairing DUKPT Key";
- 0X9049: "No nonce generated";
- 0X9949: "No GUID available. Perform getVersion first.";
- 0X9950: "MAC Calculation unsuccessful. Check BDK value.";
- 0X904A: "Not ready";
- 0X904B: "Not MAC data";
- 0X9050: "Invalid Certificate";
- 0X9051: "Duplicate key detected";
- 0X9052: "AT checks failed";
- 0X9053: "TR34 checks failed";
- 0X9054: "TR31 checks failed";
- 0X9055: "MAC checks failed";

- 0X9056: "Firmware download failed";
- 0X9060: "Log is full";
- 0X9061: "Removal sensor unengaged";
- 0X9062: "Any hardware problems";
- 0X9070: "ICC communication timeout";
- 0X9071: "ICC data error (such check sum error)";
- 0X9072: "Smart Card not powered up";

11.6.4.24 device_getSDKWaitTime()

```
int device_getSDKWaitTime ( )
```

Get SDK Wait Time

Get the SDK wait time for transactions

Returns

SDK wait time in seconds

11.6.4.25 device_getThreadStackSize()

```
int device_getThreadStackSize ( )
```

Get Thread Stack Size

Get the stack size setting for newly created threads

Returns

Thread Stack Size

11.6.4.26 device_init()

```
int device_init ( )
```

Initial the device by USB

It will detect the device and trying connect.

The connect status can be checked by [device_isConnected\(\)](#).

Note: after the function returns success, the function [device_setCurrentDevice\(\)](#) has to be called to set the device type.

Returns

RETURN_CODE: Values can be parsed with [device_getResponseCodeString\(\)](#)

11.6.4.27 device_isAttached()

```
int device_isAttached (
    int deviceType )
```

Check if the device is attached to the USB port The function [device_init\(\)](#) must be called before this function.

Parameters

<i>deviceType,the</i>	device type of the USB device
-----------------------	-------------------------------

Returns

1 if the device is attached, or 0 if the device is not attached

11.6.4.28 device_isConnected()

```
int device_isConnected ( )
```

Check the device connected status

Returns

DEVICE_DISCONNECT=0, or DEVICE_CONNECTED = 1

11.6.4.29 device_rebootDevice()

```
int 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\(\)](#)

11.6.4.30 device_registerCameraCallBk()

```
void device_registerCameraCallBk (
    pCMR_callback pCMRf )
```

To register the camera callback function to get the image data. (Pass NULL to disable the callback.)

11.6.4.31 device_registerCardStatusFrontSwitchCallBk()

```
void device_registerCardStatusFrontSwitchCallBk (
    pCSFS_callback pCSFSf )
```

To register the card status and front switch callback function to get status. (Pass NULL to disable the callback.)

11.6.4.32 `device_SendDataCommand()`

```
int device_SendDataCommand (
    IN BYTE * cmd,
    IN int cmdLen,
    IN BYTE * data,
    IN int dataLen,
    OUT BYTE * response,
    IN_OUT int * respLen )
```

Send a Command to device

Sends a command to the device .

Parameters

<i>cmd</i>	buffer of command to execute.
<i>cmdLen,the</i>	length of the buffer cmd.
<i>data</i>	buffer of IDG command data.
<i>dataLen,the</i>	length of the buffer data.
<i>response</i>	Response data
<i>respLen,the</i>	length of Response data

Returns

RETURN_CODE: Values can be parsed with [device_getResponseCodeString\(\)](#)

11.6.4.33 `device_setCurrentDevice()`

```
int device_setCurrentDevice (
    int deviceType )
```

Sets the current device to talk to

The connect status can be checked by [device_isConnected\(\)](#).

Parameters

<i>deviceType</i>	Device to connect to <pre>enum DEVICE_TYPE { IDT_DEVICE_UNKNOWN=0, IDT_DEVICE_AUGUSTA_HID, IDT_DEVICE_AUGUSTA_KB, IDT_DEVICE_AUGUSTA_S_HID, IDT_DEVICE_AUGUSTA_S_KB, IDT_DEVICE_AUGUSTA_S_TTK_HID, IDT_DEVICE_SPECTRUM_PRO, IDT_DEVICE_MINISMART_II, IDT_DEVICE_L80, IDT_DEVICE_L100, IDT_DEVICE_UNIPAY, IDT_DEVICE_UNIPAY_I_V, IDT_DEVICE_VP3300_AJ, IDT_DEVICE_KIOSK_III, IDT_DEVICE_KIOSK_III_S, IDT_DEVICE_PIP_READER, IDT_DEVICE_VENDI, IDT_DEVICE_VP3300_USB, IDT_DEVICE_UNIPAY_I_V_TTK, IDT_DEVICE_VP3300_BT, IDT_DEVICE_VP8800, IDT_DEVICE_SREDKEY2_HID, IDT_DEVICE_SREDKEY2_KB, IDT_DEVICE_NEO2, IDT_DEVICE_MINISMART_II_COM = IDT_DEVICE_NEO2+5, IDT_DEVICE_SPECTRUM_PRO_COM, IDT_DEVICE_KIOSK_III_COM, IDT_DEVICE_KIOSK_III_S_COM, IDT_DEVICE_PIP_READER_COM, IDT_DEVICE_VP3300_COM, IDT_DEVICE_NEO2_COM, IDT_DEVICE_MAX_DEVICES = IDT_DEVICE_NEO2_COM+5 };</pre>
-------------------	---

Returns

RETURN_CODE: 1: success, 0: failed

11.6.4.34 device_setSDKWaitTime()

```
void device_setSDKWaitTime (
    int waitTime )
```

Set SDK Wait Time

Set the SDK wait time for transactions

Parameters

<i>waitTime</i>	The SDK wait time for transaction in seconds. The Maximum is 2147483 seconds.
-----------------	---

11.6.4.35 device_setThreadStackSize()

```
void device_setThreadStackSize (
    int threadSize )
```

Set Thread Stack Size

Set the stack size setting for newly created threads

11.6.4.36 device_setTransactionExponent()

```
void device_setTransactionExponent (
    int exponent )
```

Sets the transaction exponent to be used with device_startTransaction. Default value is 2

Parameters

<i>exponent, The</i>	exponent to use when calling device_startTransaction
----------------------	--

11.6.4.37 device_startTransaction()

```
int device_startTransaction (
    IN double amount,
    IN double amtOther,
    IN int type,
    IN const int _timeout,
    IN BYTE * tags,
    IN int tagsLen )
```

Start Transaction Request

The tags will be returned in the callback routine.

Parameters

<i>amount</i>	Transaction amount value (tag value 9F02) • SEE IMPORTANT NOTE BELOW
---------------	---

Parameters

<i>amtOther</i>	Other amount value, if any (tag value 9F03) <ul style="list-style-type: none">• SEE IMPORTANT NOTE BELOW
<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. 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>tagsLen</i>	The length of tags data buffer.

>>>>>IMPORTANT: parameters for amount and amtOther MUST BE PASSED AS A DOUBLE VALUE WITH DECIMAL POINT. Example, do not pass 1, but instead pass 1.0 or 1.00. Otherwise, results will be unpredictable

Returns

RETURN_CODE: Values can be parsed with [device_getResponseCodeString\(\)](#)

11.6.4.38 device_updateFirmware()

```
int device_updateFirmware (
    IN BYTE * firmwareData,
    IN int firmwareDataLen,
    IN char * firmwareName,
    IN int encryptionType,
    IN BYTE * keyBlob,
    IN int keyBlobLen )
```

Update Firmware Updates the firmware of Augusta.

Parameters

<i>firmwareData</i>	Signed binary data of a firmware file provided by IDTech
<i>firmwareDataLen</i>	Length of firmwareData
<i>firmwareName</i>	Firmware name. <ul style="list-style-type: none">• For example "Augusta_S_TTK_V1.00.002.fm"
<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)
<i>keyBlobLen</i>	Length of keyBlob

Returns

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

Firmware update status is returned in the callback with the following values: sender = AUGUSTA state = Device↔ State.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

11.6.4.39 emv_activateTransaction()

```
int emv_activateTransaction (
    IN int timeout,
    IN BYTE * tags,
    IN int tagsLen,
    IN int forceOnline )
```

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>	Tags to be included in the request. Passed as a TLV stream. Example, tag 9F0C with amount 0x000000000100 would be 0x9F0C06000000000100
<i>tagsLen</i>	Length of tags
<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

Returns

RETURN_CODE: Values can be parsed with `device_getResponseCodeString` >>>>IMPORTANT: parameters for amount and amtOther MUST BE PASSED AS A DOUBLE VALUE WITH DECIMAL POINT. Example, do not pass 1, but instead pass 1.0 or 1.00. Otherwise, results will be unpredictable

11.6.4.40 emv_allowFallback()

```
void emv_allowFallback (
    IN int allow )
```

Allow fallback for EMV transactions. Default is TRUE

Parameters

<i>allow</i>	TRUE = allow fallback, FALSE = don't allow fallback
--------------	---

11.6.4.41 emv_authenticateTransaction()

```
int emv_authenticateTransaction (
    IN BYTE * updatedTLV,
    IN int updatedTLVLen )
```

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 = DFEE1A079F029F36959F37
<i>updatedTLVLen</i>	

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString

11.6.4.42 emv_authenticateTransactionWithTimeout()

```
int emv_authenticateTransactionWithTimeout (
    IN int timeout,
    IN BYTE * updatedTLV,
    IN int updatedTLVLen )
```

Authenticate EMV Transaction Request with Timeout

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>timeout</i>	Timeout value in seconds.
----------------	---------------------------

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 = DFEE1A079F029F36959F37
<i>updatedTLVLen</i>	

Returns

RETURN_CODE: Values can be parsed with `device_getResponseCodeString`

11.6.4.43 `emv_callbackResponseLCD()`

```
int emv_callbackResponseLCD (
    IN int type,
    byte selection )
```

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 <code>EMV_LCD_DISPLAY_MODE_CANCEL</code> . Otherwise, value can be <code>EMV_LCD_DISPLAY_MODE_MENU</code> , <code>EMV_LCD_DISPLAY_MODE_PROMPT</code> , or <code>EMV_LCD_DISPLAY_MODE_LANGUAGE_SELECT</code>
<i>selection</i>	If <code>type = EMV_LCD_DISPLAY_MODE_MENU</code> or <code>EMV_LCD_DISPLAY_MODE_LANGUAGE_SELECT</code> , provide the selection ID line number. Otherwise, if <code>type = EMV_LCD_DISPLAY_MODE_PROMPT</code> supply either 0x43 ('C') for Cancel, or 0x45 ('E') for Enter/accept

Returns

RETURN_CODE: Values can be parsed with `device_getResponseCodeString()`

11.6.4.44 `emv_callbackResponseMSR()`

```
int emv_callbackResponseMSR (
    IN BYTE * MSR,
    IN_OUT int MSRLen )
```

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>MSRLen</i>	the length of Swiped track data

Returns

RETURN_CODE: Values can be parsed with [device_getResponseCodeString\(\)](#)

11.6.4.45 emv_cancelTransaction()

```
int emv_cancelTransaction ( )
```

Cancel EMV Transaction

Cancels the currently executing EMV transaction.

Returns

RETURN_CODE: Values can be parsed with [device_getResponseCodeString\(\)](#)

11.6.4.46 emv_completeTransaction()

```
int emv_completeTransaction (
    IN int commError,
    IN BYTE * authCode,
    IN int authCodeLen,
    IN BYTE * iad,
    IN int iadLen,
    IN BYTE * tlvScripts,
    IN int tlvScriptsLen,
    IN BYTE * tlv,
    IN int tlvLen )
```

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(1) if host was unreachable, or FALSE(0) 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>authCodeLen</i>	the length of authCode
<i>iad</i>	Issuer Authentication Data, if any. Example 0x11223344556677883030 (tag value 91).
<i>iadLen</i>	the length of iadLen
<i>tlvScripts</i>	71/72 scripts, if any

Parameters

<i>tlvScriptsLen</i>	the length of tlvScriptsLen
<i>tlv</i>	Additional TLV data to return with transaction results (if any)
<i>tlvLen</i>	the length of tlv

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString

11.6.4.47 emv_getAutoAuthenticateTransaction()

```
int emv_getAutoAuthenticateTransaction ( )
```

Gets auto authenticate value for EMV transactions.

Returns

RETURN_CODE: TRUE = auto authenticate, FALSE = manually authenticate

11.6.4.48 emv_getAutoCompleteTransaction()

```
int emv_getAutoCompleteTransaction ( )
```

Gets auto complete value for EMV transactions.

Returns

RETURN_CODE: TRUE = auto complete, FALSE = manually complete

11.6.4.49 emv_getEMVConfigurationCheckValue()

```
int emv_getEMVConfigurationCheckValue (
    OUT BYTE * checkValue,
    IN_OUT int * checkValueLen )
```

Get EMV Kernel configuration check value info

Parameters

<i>checkValue</i>	Response returned of Kernel configuration check value info
<i>checkValueLen</i>	the length of checkValue

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString

11.6.4.50 emv_getEMVKernelCheckValue()

```
int emv_getEMVKernelCheckValue (
    OUT BYTE * checkValue,
    IN_OUT int * checkValueLen )
```

Get EMV Kernel check value info

Parameters

<i>checkValue</i>	Response returned of Kernel check value info
<i>checkValueLen</i>	the length of checkValue

Returns

RETURN_CODE: Values can be parsed with [device_getResponseCodeString](#)

11.6.4.51 emv_getEMVKernelVersion()

```
int emv_getEMVKernelVersion (
    OUT char * version )
```

DEPRECATED : please use [emv_getEMVKernelVersion_Len](#)(OUT char* version, IN_OUT int *versionLen)

Polls device for EMV Kernel Version

Parameters

<i>version</i>	Response returned of Kernel Version; needs to have at least 128 bytes of memory.
----------------	--

Returns

RETURN_CODE: Values can be parsed with [device_getResponseCodeString\(\)](#)

11.6.4.52 emv_getEMVKernelVersion_Len()

```
int emv_getEMVKernelVersion_Len (
    OUT char * version,
    IN_OUT int * versionLen )
```

Polls device for EMV Kernel Version

Parameters

<i>version</i>	Response returned of Kernel Version
<i>versionLen</i>	Length of version

Returns

RETURN_CODE: Values can be parsed with [device_getResponseCodeString\(\)](#)

11.6.4.53 `emv_registerCallBk()`

```
void emv_registerCallBk (
    pEMV_callback pEMVf )
```

To register the emv callback function to get the EMV processing response. (Pass NULL to disable the callback.)

11.6.4.54 `emv_removeAllApplicationData()`

```
int emv_removeAllApplicationData ( )
```

Remove All Application Data

Removes all the Application Data

Returns

RETURN_CODE: Values can be parsed with [device_getResponseCodeString\(\)](#)

11.6.4.55 `emv_removeAllCAPK()`

```
int emv_removeAllCAPK ( )
```

Remove All Certificate Authority Public Key

Removes all the CAPK

Returns

RETURN_CODE: Values can be parsed with [device_getResponseCodeString\(\)](#)

11.6.4.56 `emv_removeAllCRL()`

```
int emv_removeAllCRL ( )
```

Remove All Certificate Revocation List Entries

Removes all CRLentry entries

Returns

RETURN_CODE: Values can be parsed with [device_getResponseCodeString\(\)](#)

11.6.4.57 `emv_removeApplicationData()`

```
int emv_removeApplicationData (
    IN BYTE * AID,
    IN int AIDLen )
```

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>AIDLen</i>	the length of AID data buffer

Returns

RETURN_CODE: Values can be parsed with [device_getResponseCodeString\(\)](#)

11.6.4.58 emv_removeCAPK()

```
int emv_removeCAPK (
    IN BYTE * capk,
    IN int capkLen )
```

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>capkLen</i>	the length of capk data buffer

Returns

RETURN_CODE: Values can be parsed with [device_getResponseCodeString\(\)](#)

11.6.4.59 emv_removeCRL()

```
int emv_removeCRL (
    IN BYTE * list,
    IN int lsLen )
```

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>lsLen</i>	the length of list data buffer

Returns

RETURN_CODE: Values can be parsed with [device_getResponseCodeString\(\)](#)

11.6.4.60 emv_removeTerminalData()

```
int emv_removeTerminalData ( )
```

Remove Terminal Data

Removes the Terminal Data

Returns

RETURN_CODE: Values can be parsed with [device_getResponseCodeString\(\)](#)

11.6.4.61 emv_retrieveAIDList()

```
int emv_retrieveAIDList (
    OUT BYTE * AIDList,
    IN_OUT int * AIDListLen )
```

Retrieve AID list

Returns all the AID names installed on the terminal.

Parameters

<i>AIDList</i>	array of AID name byte arrays
<i>AIDListLen</i>	the length of AIDList array buffer

Returns

RETURN_CODE: Values can be parsed with [device_getResponseCodeString\(\)](#)

11.6.4.62 emv_retrieveApplicationData()

```
int emv_retrieveApplicationData (
    IN BYTE * AID,
    IN int AIDLen,
    OUT BYTE * tlv,
    IN_OUT int * tlvLen )
```

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>AIDLen</i>	the length of AID data buffer.
<i>tlv</i>	The TLV elements of the requested AID
<i>tlvLen</i>	the length of tlv data buffer.

Returns

RETURN_CODE: Values can be parsed with [device_getResponseCodeString\(\)](#)

11.6.4.63 emv_retrieveCAPK()

```
int emv_retrieveCAPK (
    IN BYTE * capk,
    IN int capkLen,
    OUT BYTE * key,
    IN_OUT int * keyLen )
```

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>capkLen</i>	the length of capk data buffer
<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>keyLen</i>	<p>the length of key data buffer</p> <ul style="list-style-type: none"> •

Returns

RETURN_CODE: Values can be parsed with [device_getResponseCodeString\(\)](#)

11.6.4.64 emv_retrieveCAPKList()

```
int emv_retrieveCAPKList (
    OUT BYTE * keys,
    IN_OUT int * keysLen )
```

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>keysLen</i>	the length of keys data buffer

Returns

RETURN_CODE: Values can be parsed with [device_getResponseCodeString\(\)](#)

11.6.4.65 emv_retrieveCRL()

```
int emv_retrieveCRL (
    OUT BYTE * list,
    IN_OUT int * lssLen )
```

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>lssLen</i>	the length of list data buffer

Returns

RETURN_CODE: Values can be parsed with [device_getResponseCodeString\(\)](#)

11.6.4.66 emv_retrieveTerminalData()

```
int emv_retrieveTerminalData (
    OUT BYTE * tlv,
    IN_OUT int * tlvLen )
```

Retrieve Terminal Data

Retrieves the Terminal Data.

Parameters

<i>tlv</i>	Response returned as a TLV
<i>tlvLen</i>	the length of tlv data buffer

Returns

RETURN_CODE: Values can be parsed with [device_getResponseCodeString\(\)](#)

11.6.4.67 `emv_retrieveTerminalID()`

```
int emv_retrieveTerminalID (
    OUT char * terminalID )
```

DEPRECATED : please use `emv_retrieveTerminalID_Len(OUT char* terminalID, IN_OUT int *terminalIDLen)`

Gets the terminal ID as printable characters .

Parameters

<i>terminalID</i>	Terminal ID string; needs to have at least 30 bytes of memory
-------------------	---

Returns

RETURN_CODE: Values can be parsed with `device_getResponseCodeString()`

11.6.4.68 `emv_retrieveTerminalID_Len()`

```
int emv_retrieveTerminalID_Len (
    OUT char * terminalID,
    IN_OUT int * terminalIDLen )
```

Gets the terminal ID as printable characters .

Parameters

<i>terminalID</i>	Terminal ID string
<i>terminalIDLen</i>	Length of terminalID

Returns

RETURN_CODE: Values can be parsed with `device_getResponseCodeString()`

11.6.4.69 `emv_retrieveTransactionResult()`

```
int emv_retrieveTransactionResult (
    IN BYTE * tags,
    IN int tagsLen,
    IDTTransactionData * cardData )
```

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>tagsLen</i>	Length of tag list
<i>cardData</i>	All requested tags returned as unencrypted, encrypted and masked TLV data in IDTTransactionData object

Returns

RETURN_CODE: Values can be parsed with [device_getResponseCodeString\(\)](#)

11.6.4.70 emv_setApplicationData()

```
int emv_setApplicationData (
    IN BYTE * name,
    IN int nameLen,
    IN BYTE * tlv,
    IN int tlvLen )
```

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>nameLen</i>	the length of name data buffer of Application name,
<i>tlv</i>	Application data in TLV format
<i>tlvLen</i>	the length of tlv data buffer

Returns

RETURN_CODE: Values can be parsed with [device_getResponseCodeString\(\)](#)

11.6.4.71 emv_setAutoAuthenticateTransaction()

```
void emv_setAutoAuthenticateTransaction (
    IN int authenticate )
```

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

Parameters

<i>authenticate</i>	TRUE = auto authenticate, FALSE = manually authenticate
---------------------	---

11.6.4.72 emv_setAutoCompleteTransaction()

```
void emv_setAutoCompleteTransaction (
    IN int complete )
```

Enables complete for EMV transactions. If a `emv_authenticateTransaction` results in code 0x0004 (go online), then `emv_completeTransaction` can automatically execute if parameter is set to TRUE

Parameters

<i>complete</i>	TRUE = auto complete, FALSE = manually complete
-----------------	---

11.6.4.73 `emv_setCAPK()`

```
int emv_setCAPK (
    IN BYTE * capk,
    IN int capkLen )
```

Set Certificate Authority Public Key

Sets the CAPK as specified by the CAKey structure

Parameters

<i>capk</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>capkLen</i>	the length of capk data buffer

Returns

RETURN_CODE: Values can be parsed with [device_getResponseCodeString\(\)](#)

11.6.4.74 `emv_setCRL()`

```
int emv_setCRL (
    IN BYTE * list,
    IN int lsLen )
```

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>lsLen</i>	the length of list data buffer

Returns

RETURN_CODE: Values can be parsed with [device_getResponseCodeString\(\)](#)

11.6.4.75 emv_setTerminalData()

```
int emv_setTerminalData (
    IN BYTE * tlv,
    IN int tlvLen )
```

Set Terminal Data

Sets the Terminal Data as specified by the TerminalData structure passed as a parameter

Parameters

<i>tlv</i>	TerminalData configuration file
<i>tlvLen</i>	the length of tlv data buffer

Return values

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

11.6.4.76 emv_setTerminalID()

```
int emv_setTerminalID (
    IN char * terminalID )
```

Sets the terminal ID as printable characters .

Parameters

<i>terminalID</i>	Terminal ID to set
-------------------	--------------------

Returns

RETURN_CODE: Values can be parsed with [device_getResponseCodeString\(\)](#)

11.6.4.77 emv_startTransaction()

```
int emv_startTransaction (
    IN double amount,
    IN double amtOther,
    IN int exponent,
    IN int type,
    IN int timeout,
    IN BYTE * tags,
    IN int tagsLen,
    IN int forceOnline )
```

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) - SEE IMPORTANT NOTE BELOW
<i>amtOther</i>	Other amount value, if any (tag value 9F03) - SEE IMPORTANT NOTE BELOW
<i>exponent</i>	Number of characters after decimal 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 stream. Example, tag 9F0C with amount 0x000000000100 would be 0x9F0C06000000000100 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>tagsLen</i>	Length of tags
<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 = DFEE1A079F029F36959F37

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString >>>>IMPORTANT: parameters for amount and amtOther MUST BE PASSED AS A DOUBLE VALUE WITH DECIMAL POINT. Example, do not pass 1, but instead pass 1.0 or 1.00. Otherwise, results will be unpredictable

11.6.4.78 `icc_disable()`

```
int icc_disable ( )
```

ICC Function enable/disable Disable ICC function

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString

11.6.4.79 `icc_enable()`

```
int icc_enable (
    IN int withNotification )
```

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

Parameters

<i>withNotification</i>	<ul style="list-style-type: none"> • 1: with notification when ICC seated status changed, • 0: without notification.
-------------------------	--

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString

11.6.4.80 icc_exchangeAPDU()

```
int icc_exchangeAPDU (
    IN BYTE * c_APDU,
    IN int cLen,
    OUT BYTE * reData,
    IN_OUT int * reLen )
```

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>cLen</i>	APDU data packet length
<i>reData</i>	Unencrypted APDU response
<i>reLen</i>	Unencrypted APDU response data length

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString

11.6.4.81 icc_exchangeEncryptedAPDU()

```
int icc_exchangeEncryptedAPDU (
    IN BYTE * c_APDU,
    IN int cLen,
    OUT BYTE * reData,
    IN_OUT int * reLen )
```

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>cLen</i>	data packet length
-------------	--------------------

Parameters

<i>reData</i>	response encrypted APDU response. Can be three options:
---------------	---

[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]

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.

Parameters

<i>reLen</i>	encrypted APDU response data length
--------------	-------------------------------------

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString

11.6.4.82 `icc_getAPDU_KSN()`

```
int icc_getAPDU_KSN (
    OUT BYTE * KSN,
    IN_OUT int * inLen )
```

Get APDU KSN

Retrieves the KSN used in ICC Encrypted APDU usage

Parameters

<i>KSN</i>	Returns the encrypted APDU packet KSN
<i>inLen</i>	KSN data length

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString

11.6.4.83 `icc_getFunctionStatus()`

```
int icc_getFunctionStatus (
    OUT int * enabled,
    OUT int * withNotification )
```

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"> • 1: ICC Function enabled, • 0: means disabled.
<i>withNotification</i>	1 means with notification when ICC seated status changed. 0 means without notification.

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString

11.6.4.84 `icc_getICCReaderStatus()`

```
int icc_getICCReaderStatus (
    OUT BYTE * status )
```

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
---------------	--

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString

11.6.4.85 `icc_getKeyFormatForICCDUKPT()`

```
int icc_getKeyFormatForICCDUKPT (
    OUT BYTE * format )
```

Get Key Format For DUKPT

Specifies how data will be encrypted with Data Key or PIN key (if DUKPT key loaded). This applies to both MSR and ICC

Parameters

<i>format</i>	<p>Response returned from method:</p> <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.
---------------	---

Returns

RETURN_CODE: Values can be parsed with `device_getResponseCodeString`

11.6.4.86 `icc_getKeyTypeForICCDUKPT()`

```
int icc_getKeyTypeForICCDUKPT (
    OUT BYTE * type )
```

Get Key Type for DUKPT

Specifies the key type used for DUKPT encryption This applies to both MSR and ICC

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.
-------------	---

Returns

RETURN_CODE: Values can be parsed with `device_getResponseCodeString`

11.6.4.87 `icc_powerOffICC()`

```
int icc_powerOffICC ( )
```

Power Off ICC

Powers down the ICC

Returns

RETURN_CODE: Values can be parsed with `device_getResponseCodeString()`

If Success, empty If Failure, ASCII encoded data of error string

11.6.4.88 `icc_powerOnICC()`

```
int icc_powerOnICC (
    OUT BYTE * ATR,
    IN_OUT int * inLen )
```

Power On ICC

Power up the currently selected microprocessor card in the ICC reader

Parameters

<i>ATR,the</i>	ATR data response when succeeded power on ICC,
<i>inLen,the</i>	length of ATR data

Returns

RETURN_CODE: Values can be parsed with [device_getResponseCodeString\(\)](#)

11.6.4.89 msr_registerCallBk()

```
void msr_registerCallBk (
    pMSR_callback pMSRf )
```

To register the msr callback function to get the MSR card data. (Pass NULL to disable the callback.)

11.6.4.90 msr_registerCallBkp()

```
void msr_registerCallBkp (
    pMSR_callbackp pMSRf )
```

To register the msr callback function to get the MSR card data pointer. (Pass NULL to disable the callback.)

11.6.4.91 pin_registerCallBk()

```
void pin_registerCallBk (
    pPIN_callback pPINf )
```

To register the pin callback function to get the PINPad data. (Pass NULL to disable the callback.)

11.6.4.92 registerHotplugCallBk()

```
void registerHotplugCallBk (
    pMessageHotplug pMsgHotplug )
```

To register the USB HID hot-plug callback function which implemented in the application to monitor the hotplug message from the SDK.

11.6.4.93 registerLogCallBk()

```
void registerLogCallBk (
    pSendDataLog pFSend,
    pReadDataLog pFRead )
```

To register the log callback function which implemented in the application to monitor sending/reading data between application and reader.

11.6.4.94 rs232_device_init()

```
int rs232_device_init (
    int deviceType,
    int port_number,
    int brate )
```

Initial the device by RS232

It will try to connect to the device with provided deviceType, port_number, and brate.

Parameters

<i>deviceType</i>	Device to connect to
-------------------	----------------------

Parameters

<i>port_number</i>	Port number of the device
--------------------	---------------------------

Port nr. | Linux | Windows

| 0 | ttyS0 | COM1 | | 1 | ttyS1 | COM2 | | 2 | ttyS2 | COM3 | | 3 | ttyS3 | COM4 | | 4 | ttyS4 | COM5 | | 5 | ttyS5 | COM6 | | 6 | ttyS6 | COM7 | | 7 | ttyS7 | COM8 | | 8 | ttyS8 | COM9 | | 9 | ttyS9 | COM10 | | 10 | ttyS10 | COM11 | | 11 | ttyS11 | COM12 | | 12 | ttyS12 | COM13 | | 13 | ttyS13 | COM14 | | 14 | ttyS14 | COM15 | | 15 | ttyS15 | COM16 | | 16 | ttyUSB0 | n.a. | | 17 | ttyUSB1 | n.a. | | 18 | ttyUSB2 | n.a. | | 19 | ttyUSB3 | n.a. | | 20 | ttyUSB4 | n.a. | | 21 | ttyUSB5 | n.a. | | 22 | ttyAMA0 | n.a. | | 23 | ttyAMA1 | n.a. | | 24 | ttyACM0 | n.a. | | 25 | ttyACM1 | n.a. | | 26 | rfcomm0 | n.a. | | 27 | rfcomm1 | n.a. | | 28 | ircomm0 | n.a. | | 29 | ircomm1 | n.a. | | 30 | cuau0 | n.a. | | 31 | cuau1 | n.a. | | 32 | cuau2 | n.a. | | 33 | cuau3 | n.a. | | 34 | cuaU0 | n.a. | | 35 | cuaU1 | n.a. | | 36 | cuaU2 | n.a. | | 37 | cuaU3 | n.a. |

Parameters

<i>brate</i>	Bitrate of the device
--------------	-----------------------

Returns

RETURN_CODE: Values can be parsed with [device_getResponseCodeString\(\)](#)

11.6.4.95 SDK_Version()

```
char* SDK_Version ( )
```

To Get SDK version

Returns

return the SDK version string

11.6.4.96 setAbsoluteLibraryPath()

```
int setAbsoluteLibraryPath (
    const char * absoluteLibraryPath )
```

Set the path to use when searching for ID TECH's libraries. If this is not set, the libraries will be searched for with the system's default procedures.

Parameters

<i>absoluteLibraryPath</i>	The absolute path to ID TECH's libraries.
----------------------------	---

Returns

RETURN_CODE: Values can be parsed with [device_getResponseCodeString\(\)](#)

11.7 Source_C/libIDT_NEO2.h File Reference

NEO2 API.

```
#include "IDTDef.h"
```

Macros

- `#define IN`
- `#define OUT`
- `#define IN_OUT`

Typedefs

- `typedef void(* pMessageHotplug) (int, int)`
- `typedef void(* pSendDataLog) (BYTE *, int)`
- `typedef void(* pReadDataLog) (BYTE *, int)`
- `typedef void(* pEMV_callBack) (int, int, BYTE *, int, IDTTransactionData *, EMV_Callback *, int)`
- `typedef void(* pWP_callBack) (char *, int, int)`
- `typedef void(* pWN_callBack) (char *, int, int)`
- `typedef void(* pFW_callBack) (int, int, int, int, int)`
- `typedef void(* pMSR_callBack) (int, IDTMSRData)`
- `typedef void(* pMSR_callBackp) (int, IDTMSRData *)`
- `typedef void(* pPIN_callBack) (int, IDTPINData *)`
- `typedef void(* pCMR_callBack) (int, IDTCMRData *)`
- `typedef void(* pCSFS_callBack) (BYTE status)`
- `typedef void(* pLCD_callBack) (int, IDTLCDItem *)`
- `typedef void(* pRKI_callBack) (int, char *)`
- `typedef void(* ftpComm_callBack) (int, int, int)`
- `typedef void(* httpComm_callBack) (BYTE *, int)`
- `typedef void(* v4Comm_callBack) (BYTE, BYTE, BYTE *, int)`

Functions

- `void registerHotplugCallBk (pMessageHotplug pMsgHotplug)`
- `void registerLogCallBk (pSendDataLog pFSend, pReadDataLog pFRead)`
- `void device_registerFWCallBk (pFW_callBack pFWf)`
- `void device_registerCameraCallBk (pCMR_callBack pCMRf)`
- `void device_registerCardStatusFrontSwitchCallBk (pCSFS_callBack pCSFSf)`
- `void device_registerRKICallBk (pRKI_callBack pRKIf)`
- `void emv_registerCallBk (pEMV_callBack pEMVf)`
- `void loyalty_registerCallBk (pEMV_callBack pEMVf)`
- `void msr_registerCallBk (pMSR_callBack pMSRf)`
- `void msr_registerCallBkp (pMSR_callBackp pMSRf)`
- `void ctls_registerCallBk (pMSR_callBack pCTLSf)`
- `void ctls_registerCallBkp (pMSR_callBackp pCTLSf)`
- `void pin_registerCallBk (pPIN_callBack pPINf)`
- `void lcd_registerCallBk (pLCD_callBack pLCDf)`
- `void comm_registerHTTPCallback (httpComm_callBack cBack)`
- `void comm_registerV4Callback (v4Comm_callBack cBack)`
- `char * SDK_Version ()`
- `int setAbsoluteLibraryPath (const char *absoluteLibraryPath)`

- int [device_setConfigPath](#) (const char *path)
- int [device_setNEO2DevicesConfigs](#) (IN const char *configs, IN int len)
- int [device_init](#) ()
- void [device_setNEOGen](#) (int gen)
- void [device_setNEOAltDevice](#) (int alt)
- int [device_getNEOAltDevice](#) ()
- int [rs232_device_init](#) (int deviceType, int port_number, int brate)
- void [set_open_com_port_timeout](#) (int timeout)
- int [device_setCurrentDevice](#) (int deviceType)
- int [device_isAttached](#) (int deviceType)
- int [device_close](#) ()
- void [device_getIDGStatusCodeString](#) (IN int returnCode, OUT char *despcrition)
- int [device_isConnected](#) ()
- int [device_getFirmwareVersion](#) (OUT char *firmwareVersion)
- int [device_getFirmwareVersion_Len](#) (OUT char *firmwareVersion, IN_OUT int *firmwareVersionLen)
- int [device_getDeviceTreeVersion](#) (OUT char *deviceTreeVersion, IN_OUT int *deviceTreeVersionLen)
- int [device_pingDevice](#) ()
- int [device_controlUserInterface](#) (IN BYTE *values)
- int [device_getCurrentDeviceType](#) ()
- int [device_SendDataCommandNEO](#) (IN int cmd, IN int subCmd, IN BYTE *data, IN int dataLen, OUT BYTE *response, IN_OUT int *respLen)
- int [device_rebootDevice](#) ()
- int [device_stopAudio](#) ()
- int [device_playAudio](#) (IN char *fileName, IN int fileNameLen, IN int onSD)
- int [device_getAudioVolume](#) (OUT BYTE *volume)
- int [device_setAudioVolume](#) (IN BYTE volume)
- int [device_getCameraParameters](#) (OUT BYTE *isAutoFocus, OUT BYTE *focalLength)
- int [device_setCameraParameters](#) (IN BYTE isAutoFocus, IN BYTE focalLength)
- int [device_getSDKWaitTime](#) ()
- void [device_setSDKWaitTime](#) (int waitTime)
- int [device_getThreadStackSize](#) ()
- void [device_setThreadStackSize](#) (int threadSize)
- void [device_toSDCard](#) (int forSDCard)
- int [device_startRKI](#) (IN const char *caPath, IN int isProduction)
- void [device_setRKI_URL](#) (IN char *rkiURL, IN int rkiURLLen)
- int [ctl_displayOnlineAuthResult](#) (IN int statusCode, IN BYTE *TLV, IN int TLVLen)
- int [device_enablePassThrough](#) (int enablePassThrough)
- int [device_enableL80PassThrough](#) (int enableL80PassThrough)
- int [device_enableL100PassThrough](#) (int enableL100PassThrough)
- int [device_getL80PassThroughMode](#) ()
- int [device_getL100PassThroughMode](#) ()
- int [device_setBurstMode](#) (IN BYTE mode)
- int [device_setPollMode](#) (IN BYTE mode)
- int [device_pollForToken](#) (IN int timeout, OUT BYTE *respData, IN_OUT int *respDataLen)
- int [device_setMerchantRecord](#) (int index, int enabled, char *merchantID, char *merchantURL)
- int [device_getMerchantRecord](#) (IN int index, OUT BYTE *record)
- int [device_getMerchantRecord_Len](#) (IN int index, OUT BYTE *record, IN_OUT int *recordLen)
- int [device_getTransactionResults](#) (IDTMSRData *cardData)
- int [device_startTransaction](#) (IN double amount, IN double amtOther, IN int type, IN const int _timeout, IN BYTE *tags, IN int tagsLen)
- int [loyalty_startTransaction](#) (IN double amount, IN double amtOther, IN int type, IN const int _timeout, IN BYTE *tags, IN int tagsLen, IN const int cardType, IN const int iccReadType)
- void [device_setTransactionExponent](#) (int exponent)
- int [device_activateTransaction](#) (IN const int _timeout, IN BYTE *tags, IN int tagsLen)
- int [device_cancelTransaction](#) ()

- int [loyalty_cancelTransaction](#) ()
- int [device_setCancelTransactionMode](#) (int mode)
- int [device_cancelTransactionSilent](#) (int enable)
- int [loyalty_cancelTransactionSilent](#) (int enable)
- int [device_configureButtons](#) (IN BYTE done, IN BYTE swipe, IN BYTE delay)
- int [device_getButtonConfiguration](#) (OUT BYTE *done, OUT BYTE *swipe, OUT BYTE *delay)
- int [device_disableBlueLED](#) ()
- int [device_enableBlueLED](#) (IN BYTE *data, IN int dataLen)
- int [device_lcdDisplayClear](#) ()
- int [device_enableExternalLCDMessages](#) (IN int enableExtLCDMsg)
- int [device_enableRFAntenna](#) (IN int enableAntenna)
- int [device_turnOffYellowLED](#) ()
- int [device_turnOnYellowLED](#) ()
- int [device_buzzerOnOff](#) ()
- int [device_lcdDisplayLine1Message](#) (IN BYTE *message, IN int messageLen)
- int [device_lcdDisplayLine2Message](#) (IN BYTE *message, IN int messageLen)
- int [device_getKeyStatus](#) (int *newFormat, BYTE *status, int *statusLen)
- int [device_updateFirmware](#) (IN BYTE *firmwareData, IN int firmwareDataLen, IN char *firmwareName, IN int encryptionType, IN BYTE *keyBlob, IN int keyBlobLen)
- int [device_transferFile](#) (IN char *fileName, IN int fileNameLen, IN BYTE *file, IN int fileLen)
- int [device_deleteFile](#) (IN char *fileName, IN int fileNameLen)
- int [device_queryFile](#) (IN char *directoryName, IN int directoryNameLen, IN char *fileName, IN int fileNameLen, OUT int *isExist, OUT BYTE *timeStamp, IN_OUT int *timeStampLen, OUT char *fileSize, IN_OUT int *fileSizeLen)
- int [device_readFileFromSD](#) (IN char *directoryName, IN int directoryNameLen, IN char *fileName, IN int fileNameLen, IN int startingOffset, IN int numBytes, OUT BYTE *fileData, IN_OUT int *fileDataLen)
- int [device_startListenNotifications](#) ()
- int [device_stopListenNotifications](#) ()
- int [device_startQRCodeScan](#) (IN int _timeout)
- int [device_startQRCodeScanWithDisplayWindowInfo](#) (IN int _timeout, IN int x, IN int y, IN int width, IN int height)
- int [device_stopQRCodeScan](#) ()
- int [device_startTakingPhoto](#) (IN int _timeout)
- int [device_stopTakingPhoto](#) ()
- void [device_getResponseCodeString](#) (IN int returnCode, OUT char *description)
- int [device_listDirectory](#) (IN char *directoryName, IN int directoryNameLen, IN int recursive, IN int onSD, OUT char *directory, IN_OUT int *directoryLen)
- int [device_deleteDirectory](#) (IN char *dirName, IN int dirNameLen)
- int [device_getDeviceMemoryUsageInfo](#) (OUT int *freeHeapSize, OUT int *notFreedBlockCnt, OUT int *minEverFreeHeapSize)
- int [device_getDRS](#) (BYTE *codeDRS, int *codeDRSLen)
- int [device_setCoreDumpLogFile](#) (IN char *filename, IN int filenameLen)
- int [device_outputLog](#) (IN char *filename, IN int filenameLen)
- int [device_getTamperStatus](#) (OUT int *isTampered)
- int [device_loadCertCA](#) (IN BYTE CertType, IN BYTE *CACertData, IN int CACertDataLen)
- int [device_rrcDownloadApp](#) (IN char *zipFileName, IN int zipFileNameLen, IN char *appName, IN int appNameLen)
- int [device_rrcUninstallApp](#) (IN char *appName, IN int appNameLen)
- int [device_rrcInstallApp](#) (IN char *appName, IN int appNameLen)
- int [device_rrcRunApp](#) (IN char *appName, IN int appNameLen)
- int [device_rrcConnect](#) ()
- int [device_rrcDisconnect](#) ()
- int [felica_authentication](#) (IN BYTE *key, IN int keyLen)
- int [felica_readWithMac](#) (IN int blockCnt, IN BYTE *blockList, IN int blockListLen, OUT BYTE *blockData, OUT int *blockDataLen)

- int felica_writeWithMac (IN BYTE blockNum, IN BYTE *blockData, IN int blockDataLen)
- int felica_read (IN BYTE *serviceCodeList, IN int serviceCodeListLen, IN int blockCnt, IN BYTE *blockList, IN int blockListLen, OUT BYTE *blockData, OUT int *blockDataLen)
- int felica_write (IN BYTE *serviceCodeList, IN int serviceCodeListLen, IN int blockCnt, IN BYTE *blockList, IN int blockListLen, IN BYTE *blockData, IN int blockDataLen, OUT BYTE *statusFlag, OUT int *statusFlagLen)
- int felica_poll (IN BYTE *systemCode, IN int systemCodeLen, OUT BYTE *respData, OUT int *respDataLen)
- int felica_SendCommand (IN BYTE *command, IN int commandLen, OUT BYTE *respData, OUT int *respDataLen)
- int felica_requestService (IN BYTE *nodeCode, IN int nodeCodeLen, OUT BYTE *respData, OUT int *respDataLen)
- int felica_getCode ()
- int felica_cancelCodeEntry ()
- int config_getSerialNumber (OUT char *sNumber)
- int config_getSerialNumber_Len (OUT char *sNumber, IN_OUT int *sNumberLen)
- int config_getModelNumber (OUT char *sNumber)
- int config_getModelNumber_Len (OUT char *sNumber, IN_OUT int *sNumberLen)
- int config_setCmdTimeOutDuration (IN int millisecond)
- int config_setConfigByJsonFile (IN char *jsonFileName, IN int jsonFileNameLen)
- int ctls_startTransaction (IN double amount, IN double amtOther, IN int type, IN const int _timeout, IN BYTE *tags, IN int tagsLen)
- int ctls_activateTransaction (IN const int _timeout, IN BYTE *tags, IN int tagsLen)
- int ctls_cancelTransaction ()
- int ctls_retrieveApplicationData (IN BYTE *AID, IN int AIDLen, OUT BYTE *tlv, IN_OUT int *tlvLen)
- int ctls_setApplicationData (IN BYTE *tlv, IN int tlvLen)
- int ctls_removeApplicationData (IN BYTE *AID, IN int AIDLen)
- int ctls_removeAllApplicationData ()
- int ctls_retrieveAIDList (OUT BYTE *AIDList, IN_OUT int *AIDListLen)
- int ctls_retrieveTerminalData (OUT BYTE *tlv, IN_OUT int *tlvLen)
- int ctls_setTerminalData (IN BYTE *tlv, IN int tlvLen)
- int ctls_retrieveCAPK (IN BYTE *capk, IN int capkLen, OUT BYTE *key, IN_OUT int *keyLen)
- int ctls_setCAPK (IN BYTE *capk, IN int capkLen)
- int ctls_removeCAPK (IN BYTE *capk, IN int capkLen)
- int ctls_removeAllCAPK ()
- int ctls_retrieveCAPKList (OUT BYTE *keys, IN_OUT int *keysLen)
- int ctls_setConfigurationGroup (IN BYTE *tlv, IN int tlvLen)
- int ctls_getConfigurationGroup (IN int group, OUT BYTE *tlv, OUT int *tlvLen)
- int ctls_getAllConfigurationGroups (OUT BYTE *tlv, IN_OUT int *tlvLen)
- int ctls_removeConfigurationGroup (int group)
- int emv_getEMVKernelVersion (OUT char *version)
- int emv_getEMVKernelVersion_Len (OUT char *version, IN_OUT int *versionLen)
- int emv_getEMVKernelCheckValue (OUT BYTE *checkValue, IN_OUT int *checkValueLen)
- int emv_getEMVConfigurationCheckValue (OUT BYTE *checkValue, IN_OUT int *checkValueLen)
- void emv_allowFallback (IN int allow)
- void emv_setAutoAuthenticateTransaction (IN int authenticate)
- void emv_setAutoCompleteTransaction (IN int complete)
- int emv_getAutoAuthenticateTransaction ()
- int emv_getAutoCompleteTransaction ()
- void emv_setTransactionParameters (IN double amount, IN double amtOther, IN int type, IN int timeout, IN BYTE *tags, IN int tagsLen)
- int emv_startTransaction (IN double amount, IN double amtOther, IN int exponent, IN int type, IN int timeout, IN BYTE *tags, IN int tagsLen, IN int forceOnline)
- int emv_activateTransaction (IN int timeout, IN BYTE *tags, IN int tagsLen, IN int forceOnline)
- int emv_authenticateTransaction (IN BYTE *updatedTLV, IN int updatedTLVLen)
- int emv_authenticateTransactionWithTimeout (IN int timeout, IN BYTE *updatedTLV, IN int updatedTLVLen)

- `int emv_completeTransaction (IN int commError, IN BYTE *authCode, IN int authCodeLen, IN BYTE *iad, IN int iadLen, IN BYTE *tlvScripts, IN int tlvScriptsLen, IN BYTE *tlv, IN int tlvLen)`
- `int emv_cancelTransaction ()`
- `int emv_retrieveTransactionResult (IN BYTE *tags, IN int tagsLen, OUT IDTTransactionData *cardData)`
- `int emv_retrieveApplicationData (IN BYTE *AID, IN int AIDLen, OUT BYTE *tlv, IN_OUT int *tlvLen)`
- `int emv_setApplicationData (IN BYTE *name, IN int nameLen, IN BYTE *tlv, IN int tlvLen)`
- `int emv_setApplicationDataTLV (IN BYTE *tlv, IN int tlvLen)`
- `int emv_removeApplicationData (IN BYTE *AID, IN int AIDLen)`
- `int emv_removeAllApplicationData ()`
- `int emv_retrieveAIDList (OUT BYTE *AIDList, IN_OUT int *AIDListLen)`
- `int emv_retrieveTerminalData (OUT BYTE *tlv, IN_OUT int *tlvLen)`
- `int emv_setTerminalData (IN BYTE *tlv, IN int tlvLen)`
- `int emv_setTerminalMajorConfiguration (IN int configuration)`
- `int emv_retrieveCAPK (IN BYTE *capk, IN int capkLen, OUT BYTE *key, IN_OUT int *keyLen)`
- `int emv_setCAPK (IN BYTE *capk, IN int capkLen)`
- `int emv_removeCAPK (IN BYTE *capk, IN int capkLen)`
- `int emv_removeAllCAPK ()`
- `int emv_retrieveCAPKList (OUT BYTE *keys, IN_OUT int *keysLen)`
- `int emv_retrieveCRL (OUT BYTE *list, IN_OUT int *lssLen)`
- `int emv_setCRL (IN BYTE *list, IN int lsLen)`
- `int emv_removeCRL (IN BYTE *list, IN int lsLen)`
- `int emv_removeAllCRL ()`
- `int icc_getICCRReaderStatus (OUT BYTE *status)`
- `int icc_powerOnICC (OUT BYTE *ATR, IN_OUT int *inLen)`
- `int icc_powerOffICC ()`
- `int icc_exchangeAPDU (IN BYTE *c_APDU, IN int cLen, OUT BYTE *reData, IN_OUT int *reLen)`
- `int lcd_createScreen (IN char *screenName, IN int screenNameLen, OUT int *ScreenID)`
- `int lcd_destroyScreen (IN char *screenName, IN int screenNameLen)`
- `int lcd_getActiveScreen (OUT char *screenName, IN_OUT int *screenNameLen)`
- `int lcd_showScreen (IN char *screenName, IN int screenNameLen)`
- `int lcd_getButtonEvent (OUT int *screenID, OUT int *objectID, OUT char *screenName, IN_OUT int *screenNameLen, OUT char *objectName, IN_OUT int *objectNameLen, OUT int *isLongPress)`
- `int lcd_addButton (IN char *screenName, IN int screenNameLen, IN char *buttonName, IN int buttonNameLen, IN BYTE type, IN BYTE alignment, IN int xCord, IN int yCord, IN char *label, IN int labelLen, OUT IDTLCDItem *returnItem)`
- `int lcd_addEthernet (IN char *screenName, IN int screenNameLen, IN char *objectName, IN int objectNameLen, IN BYTE alignment, IN int xCord, IN int yCord, OUT IDTLCDItem *returnItem)`
- `int lcd_addLED (IN char *screenName, IN int screenNameLen, IN char *objectName, IN int objectNameLen, IN BYTE alignment, IN int xCord, IN int yCord, OUT IDTLCDItem *returnItem, IN BYTE *LED, IN int LEDLen)`
- `int lcd_addText (IN char *screenName, IN int screenNameLen, IN char *objectName, IN int objectNameLen, IN BYTE alignment, IN int xCord, IN int yCord, IN int width, IN int height, IN BYTE fontID, IN BYTE *color, IN int colorLen, IN char *label, IN int labelLen, OUT IDTLCDItem *returnItem)`
- `int lcd_addImage (IN char *screenName, IN int screenNameLen, IN char *objectName, IN int objectNameLen, IN BYTE alignment, IN int xCord, IN int yCord, IN char *filename, IN int filenameLen, OUT IDTLCDItem *returnItem)`
- `int lcd_addVideo (IN char *screenName, IN int screenNameLen, IN char *objectName, IN int objectNameLen, IN BYTE alignment, IN int xCord, IN int yCord, IN char *filename, IN int filenameLen, OUT IDTLCDItem *returnItem)`
- `int lcd_addExtVideo (IN char *screenName, IN int screenNameLen, IN char *objectName, IN int objectNameLen, IN BYTE alignment, IN int xCord, IN int yCord, IN BYTE loop, IN BYTE numVideos, IN char *filenames, IN int filenamesLen, OUT IDTLCDItem *returnItem)`
- `int lcd_cloneScreen (IN char *screenName, IN int screenNameLen, IN char *cloneName, IN int cloneNameLen, OUT int *cloneID)`
- `int lcd_updateLabel (IN char *screenName, IN int screenNameLen, IN char *objectName, IN int objectNameLen, IN char *label, IN int labelLen)`

- int [lcd_updateColor](#) (IN char *screenName, IN int screenNameLen, IN char *objectName, IN int objectNameLen, IN BYTE *color, IN int colorLen)
- int [lcd_updatePosition](#) (IN char *screenName, IN int screenNameLen, IN char *objectName, IN int objectNameLen, IN BYTE alignment, IN int new_xCord, IN int new_yCord)
- int [lcd_removeItem](#) (IN char *screenName, IN int screenNameLen, IN char *objectName, IN int objectNameLen)
- int [lcd_storeScreenInfo](#) ()
- int [lcd_loadScreenInfo](#) ()
- int [lcd_clearScreenInfo](#) ()
- int [lcd_clearDisplay](#) (IN BYTE control)
- int [lcd_displayMessage](#) (int lineNumber, char *message, int messageLen)
- int [lcd_getAllScreens](#) (IN_OUT int *screenNumbers, OUT IDTScreenInfo *screenInfo)
- int [lcd_getAllObjects](#) (IN char *screenName, IN int screenNameLen, IN_OUT int *objectNumbers, OUT IDTObjectInfo *objectInfo)
- int [lcd_queryScreenbyName](#) (IN char *screenName, IN int screenNameLen, OUT int *result)
- int [lcd_linkUIWithTransactionMessageId](#) (IN BYTE MessageId, IN char *screenName, IN int screenNameLen)
- int [lcd_queryObjectbyName](#) (IN char *objectName, IN int objectNameLen, IN_OUT int *objectNumbers, OUT IDTScreenInfo *screenInfo)
- int [lcd_queryScreenbyID](#) (IN int screenID, OUT int *result, OUT int *screenName, IN_OUT int *screenNameLen)
- int [lcd_queryObjectbyID](#) (IN int objectID, OUT int *objectNumbers, OUT IDTScreenInfo *screenInfo)
- int [lcd_setBacklight](#) (IN BYTE isBacklightOn, IN BYTE backlightVal)
- int [msr_cancelMSRSwipe](#) ()
- int [msr_startMSRSwipe](#) (IN int _timeout)
- int [executeTransaction](#) (WorldPayData *data, pWP_callback wpCallback, int requestOnly)
- int [forwardTransaction](#) (IN pWP_callback wpCallback, IN char *forwardID, IN int forwardIDLen, IN char *password, IN int passwordLen, IN int bypassProcessing)
- int [cancelWorldPay](#) ()
- int [executeTransaction_WorldNet](#) (WorldNetData *data, pWN_callback wnCallback, int requestOnly)
- int [forwardTransaction_WorldNet](#) (IN char *apiKey, IN int apiKeyLen, IN pWN_callback wnCallback, IN char *forwardID, IN int forwardIDLen, IN char *password, IN int passwordLen, IN int bypassProcessing)
- int [cancelWorldNet](#) ()
- void [parseMSRData](#) (IN BYTE *resData, IN int resLen, IN_OUT IDTMSRData *cardData)
- int [pin_capturePin](#) (IN int timeout, IN int type, IN char *PAN, IN int PANLen, IN int minPIN, IN int maxPIN, IN char *message, IN int messageLen)
- int [pin_capturePinExt](#) (IN int type, IN char *PAN, IN int PANLen, IN int minPIN, IN int maxPIN, IN char *message, IN int messageLen, IN char *verify, IN int verifyLen)
- int [pin_promptForNumericKeyWithSwipe](#) (IN int timeout, IN BYTE function, IN int minLen, IN int maxLen, IN char *line1, IN int line1Len, IN char *line2, IN int line2Len, BYTE *signature, IN int signatureLen)
- int [pin_promptForNumericKey](#) (IN int timeout, IN int maskInput, IN int minLen, IN int maxLen, IN char *message, IN int messageLen, BYTE *signature, IN int signatureLen)
- int [pin_inputFromPrompt](#) (BYTE mask, BYTE preClearText, BYTE postClearText, int minLen, int maxLen, char *lang, BYTE promptID, char *defaultResponse, int defaultResponseLen, int timeout)
- int [pin_getPanEntry](#) (IN int csc, IN int expDate, IN int ADR, IN int ZIP, IN int mod10CK, IN int timeout, IN int encPANOnly)
- int [pin_cancelPINEntry](#) ()
- int [pin_setKeyValues](#) (int mode)

11.7.1 Detailed Description

NEO2 API.

NEO2 Global API methods.

11.7.2 Macro Definition Documentation

11.7.2.1 IN

```
#define IN
```

INPUT parameter.

11.7.2.2 IN_OUT

```
#define IN_OUT
```

INPUT / OUTPUT PARAMETER.

11.7.2.3 OUT

```
#define OUT
```

OUTPUT parameter.

11.7.3 Typedef Documentation

11.7.3.1 ftpComm_callBack

```
typedef void(* ftpComm_callBack) (int, int, int)
```

Define the comm callback function to get FTP file transfer status

It should be passed as a parameter in a FTP request, Signature (int, int, int) = response code, current block, total blocks
 RESPONSE CODES: 100 = FILE DOWNLOAD STARTED 101 = FILE BLOCK XX OF XX RECEIVED 102 = FILE DOWNLOAD COMPLETED 103 = FILE DOWNLOAD TERMINATED PREMATURELY

11.7.3.2 httpComm_callBack

```
typedef void(* httpComm_callBack) (BYTE *, int)
```

Define the comm callback function to get the async url data

It should be registered using the comm_registerHTTPCallback

11.7.3.3 pCMR_callBack

```
typedef void(* pCMR_callBack) (int, IDTCMRData *)
```

Define the camera callback function to get the image data

It should be registered using the device_registerCameraCallBk,

11.7.3.4 pCSFS_callBack

```
typedef void(* pCSFS_callBack) (BYTE status)
```

Define the card status and front switch callback function to get card and front switch status

It should be registered using the device_registerCardStatusFrontSwitchCallBk,

11.7.3.5 pEMV_callBack

```
typedef void(* pEMV_callBack) (int, int, BYTE *, int, IDTTransactionData *, EMV_Callback *, int)
```

Define the EMV callback function to get the transaction message/data/result.
It should be registered using the `emv_registerCallBk`

11.7.3.6 pFW_callBack

```
typedef void(* pFW_callBack) (int, int, int, int, int)
```

Define the firmware update callback function to get the firmware update status
It should be registered using the `device_registerFWCallBk`

11.7.3.7 pLCD_callBack

```
typedef void(* pLCD_callBack) (int, IDTLCDItem *)
```

Define the LCD callback function to get the input LCDItem
It should be registered using the `lcd_registerCallBk`,

11.7.3.8 pMessageHotplug

```
typedef void(* pMessageHotplug) (int, int)
```

Define the USB hot-plug callback function to monitor the info when plug in/out the reader.
It should be registered using the `registerHotplugCallBk`, The first integer parameter is device type, and the second integer parameter is either 0: Device Plugged Out or 1: Device Plugged In

11.7.3.9 pMSR_callBack

```
typedef void(* pMSR_callBack) (int, IDTMSRData)
```

Define the MSR callback function to get the MSR card data
It should be registered using the `msr_registerCallBk`, this callback function is for backward compatibility

11.7.3.10 pMSR_callBackp

```
typedef void(* pMSR_callBackp) (int, IDTMSRData *)
```

Define the MSR callback function to get pointer to the MSR card data
It should be registered using the `msr_registerCallBk`, this callback function is recommended instead of `pMSR_callBack`

11.7.3.11 pPIN_callBack

```
typedef void(* pPIN_callBack) (int, IDTPINData *)
```

Define the PINPad callback function to get the input PIN Pad data
It should be registered using the `pin_registerCallBk`,

11.7.3.12 pReadDataLog

```
typedef void(* pReadDataLog) (BYTE *, int)
```

Define the read response callback function to monitor the reading response from the reader.
It should be registered using the registerLogCallBk

11.7.3.13 pRKI_callback

```
typedef void(* pRKI_callback) (int, char *)
```

Define the RKI callback function to get the status of the RKI
It should be registered using the device_registerRKICallBk,

11.7.3.14 pSendDataLog

```
typedef void(* pSendDataLog) (BYTE *, int)
```

Define the send command callback function to monitor the sending command into the reader.
It should be registered using the registerLogCallBk

11.7.3.15 pWN_callback

```
typedef void(* pWN_callback) (char *, int, int)
```

Define the Worldnet callback function to get the transaction message/data/result.

11.7.3.16 pWP_callback

```
typedef void(* pWP_callback) (char *, int, int)
```

Define the Worldpay callback function to get the transaction message/data/result.

11.7.3.17 v4Comm_callback

```
typedef void(* v4Comm_callback) (BYTE, BYTE, BYTE *, int)
```

Define the comm callback function to receive the V4 Protocol packets received by the device from an external source (IP/USB/RS-232) It should be registered using the comm_registerV4Callback, Data callback will contain command, sub-command, and data from V4 packet

11.7.4 Function Documentation

11.7.4.1 cancelWorldNet()

```
int cancelWorldNet ( )
```

Cancels WorldNet transaction.

Returns

RETURN_CODE: Values can be parsed with [device_getIDGStatusCodeString\(\)](#)

11.7.4.2 cancelWorldPay()

```
int cancelWorldPay ( )
```

Cancels WorldPay transaction.

Returns

RETURN_CODE: Values can be parsed with [device_getIDGStatusCodeString\(\)](#)

11.7.4.3 comm_registerHTTPCallback()

```
void comm_registerHTTPCallback (
    httpComm_callBack cBack )
```

Register Comm HTTP Async Callback

Parameters

<i>cBack</i>	- HTTP Comm callback
--------------	----------------------

11.7.4.4 comm_registerV4Callback()

```
void comm_registerV4Callback (
    v4Comm_callBack cBack )
```

Register External V4 Protocol commands Callback

Parameters

<i>cBack</i>	- V4 Protocol Comm callback
--------------	-----------------------------

11.7.4.5 config_getModelNumber()

```
int config_getModelNumber (
    OUT char * sNumber )
```

DEPRECATED : please use [config_getModelNumber_Len\(OUT char* sNumber, IN_OUT int *sNumberLen\)](#)

Polls device for Model Number

Parameters

<i>sNumber</i>	Returns Model Number; needs to have at least 64 bytes of memory
----------------	---

Returns

RETURN_CODE: Values can be parsed with [device_getIDGStatusCodeString](#)

11.7.4.6 config_getModelNumber_Len()

```
int config_getModelNumber_Len (
    OUT char * sNumber,
    IN_OUT int * sNumberLen )
```

Polls device for Model Number

Parameters

<i>sNumber</i>	Returns Model Number
<i>sNumber</i>	length of Model Number

Returns

RETURN_CODE: Values can be parsed with device_getIDGStatusCodeString

11.7.4.7 config_getSerialNumber()

```
int config_getSerialNumber (
    OUT char * sNumber )
```

DEPRECATED : please use [config_getSerialNumber_Len\(OUT char* sNumber, IN_OUT int *sNumberLen\)](#)

Polls device for Serial Number

Parameters

<i>sNumber</i>	Returns Serial Number; needs to have at least 64 bytes of memory
----------------	--

Returns

RETURN_CODE: Values can be parsed with device_getIDGStatusCodeString

11.7.4.8 config_getSerialNumber_Len()

```
int config_getSerialNumber_Len (
    OUT char * sNumber,
    IN_OUT int * sNumberLen )
```

Polls device for Serial Number

Parameters

<i>sNumber</i>	Returns Serial Number
<i>sNumberLen</i>	Length of Serial Number

Returns

RETURN_CODE: Values can be parsed with device_getIDGStatusCodeString

11.7.4.9 config_setCmdTimeOutDuration()

```
int config_setCmdTimeOutDuration (
    IN int millisecond )
```

Set the timeout duration for regular commands. The new timeout value will affect all the functions actually send (sync) commands that doesn't need to wait for a callback function, such as [device_getFirmwareVersion\(\)](#), [device_pingDevice\(\)](#), [device_SendDataCommandNEO\(\)](#), [device_enablePassThrough\(\)](#), [device_setBurstMode\(\)](#), [device_setPollMode\(\)](#), [device_updateFirmware\(\)](#) ... etc.

Parameters

<i>millisecond</i>	timeout value in milliseconds. Please consult the firmware team for the proper value.
--------------------	---

Returns

RETURN_CODE: Values can be parsed with [device_getIDGStatusCodeString\(\)](#)

11.7.4.10 config_setConfigByJsonFile()

```
int config_setConfigByJsonFile (
    IN char * jsonFileName,
    IN int jsonFileNameLen )
```

Set Configuration by JSON File

Parameters

<i>jsonFileName</i>	<ul style="list-style-type: none">• The JSON file name to set the configuration
<i>jsonFileNameLen</i>	<ul style="list-style-type: none">• The length of the JSON file name

Returns

RETURN_CODE: Values can be parsed with [device_getIDGStatusCodeString\(\)](#)

11.7.4.11 ctls_activateTransaction()

```
int ctls_activateTransaction (
    IN const int _timeout,
    IN BYTE * tags,
    IN int tagsLen )
```

Start CTLS Transaction Request

Authorizes the CTLS 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 stream. Example, tag 9F02 with amount 0x000000000100 would be 0x9F0206000000000100
<i>tagsLen</i>	The length of tags data buffer.

>>>>>IMPORTANT: parameters for amount and amtOther MUST BE PASSED AS A DOUBLE VALUE WITH DECIMAL POINT. Example, do not pass 1, but instead pass 1.0 or 1.00. Otherwise, results will be unpredictable

Returns

RETURN_CODE: Values can be parsed with [device_getIDGStatusCodeString\(\)](#) Note: if auto poll is on, it will return the error IDG_P2_STATUS_CODE_COMMAND_NOT_ALLOWED

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

11.7.4.12 ctls_cancelTransaction()

```
int ctls_cancelTransaction ( )
```

Cancel EMV Transaction

Cancels the currently executing EMV transaction.

Returns

RETURN_CODE: Values can be parsed with [device_getIDGStatusCodeString\(\)](#)

11.7.4.13 ctls_displayOnlineAuthResult()

```
int ctls_displayOnlineAuthResult (
    IN int statusCode,
    IN BYTE * TLV,
    IN int TLVLen )
```

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. The SDK timeout of the command is set to 7 seconds.

Parameters

<i>statusCode</i>	1 = OK, 0 = NOT OK, 2 = ARC response 89 for Interac
<i>TLV</i>	Optional TLV for AOSA
<i>TLVLen</i>	TLV Length

Returns

RETURN_CODE: Values can be parsed with [device_getIDGStatusCodeString\(\)](#)

11.7.4.14 ctls_getAllConfigurationGroups()

```
int ctls_getAllConfigurationGroups (
    OUT BYTE * tlv,
    IN_OUT int * tlvLen )
```

Retrieve All Configuration Groups

Returns all the Configuration Groups installed on the terminal for CTLS

Parameters

<i>tlv</i>	The TLV elements data
<i>tlvLen</i>	the length of tlv data buffer.

Returns

RETURN_CODE: Values can be parsed with [device_getIDGStatusCodeString\(\)](#)

11.7.4.15 ctls_getConfigurationGroup()

```
int ctls_getConfigurationGroup (
    IN int group,
    OUT BYTE * tlv,
    OUT int * tlvLen )
```

Get Configuration Group

Retrieves the Configuration for the specified Group.

Parameters

<i>group</i>	Configuration Group
<i>tlv</i>	return data
<i>tlvLen</i>	the length of tlv data buffer

Returns

RETURN_CODE: Values can be parsed with [device_getIDGStatusCodeString\(\)](#)

11.7.4.16 ctls_registerCallBk()

```
void ctls_registerCallBk (
    pMSR_callback pCTLSf )
```

*To register the loyalty MSR callback function to get the MSR card data. (Pass NULL to disable the callback.) *To register the loyalty MSR callback function to get the MSR card data pointer. (Pass NULL to disable the callback.) To register the ctls callback function to get the MSR card data. (Pass NULL to disable the callback.)

11.7.4.17 ctls_registerCallBkp()

```
void ctls_registerCallBkp (
    pMSR_callbackp pCTLSf )
```

To register the ctls callback function to get the MSR card data pointer. (Pass NULL to disable the callback.)

11.7.4.18 ctls_removeAllApplicationData()

```
int ctls_removeAllApplicationData ( )
```

Remove All Application Data

Removes all the Application Data

Returns

RETURN_CODE: Values can be parsed with [device_getIDGStatusCodeString\(\)](#)

11.7.4.19 ctls_removeAllCAPK()

```
int ctls_removeAllCAPK ( )
```

Remove All Certificate Authority Public Key

Removes all the CAPK

Returns

RETURN_CODE: Values can be parsed with [device_getIDGStatusCodeString\(\)](#)

11.7.4.20 ctls_removeApplicationData()

```
int ctls_removeApplicationData (
    IN BYTE * AID,
    IN int AIDLen )
```

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>AIDLen</i>	the length of AID data buffer

Returns

RETURN_CODE: Values can be parsed with [device_getIDGStatusCodeString\(\)](#)

11.7.4.21 ctls_removeCAPK()

```
int ctls_removeCAPK (
    IN BYTE * capk,
    IN int capkLen )
```

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>capkLen</i>	the length of capk data buffer

Returns

RETURN_CODE: Values can be parsed with [device_getIDGStatusCodeString\(\)](#)

11.7.4.22 ctls_removeConfigurationGroup()

```
int ctls_removeConfigurationGroup (
    int group )
```

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>	Values can be parsed with device_getIDGStatusCodeString()
--------------------	---

11.7.4.23 ctls_retrieveAIDList()

```
int ctls_retrieveAIDList (
    OUT BYTE * AIDList,
    IN_OUT int * AIDListLen )
```

Retrieve AID list

Returns all the AID names installed on the terminal for CTLS. .

Parameters

<i>AIDList</i>	array of AID name byte arrays
<i>AIDListLen</i>	the length of AIDList array buffer

Returns

RETURN_CODE: Values can be parsed with [device_getIDGStatusCodeString\(\)](#)

11.7.4.24 ctls_retrieveApplicationData()

```
int ctls_retrieveApplicationData (
    IN BYTE * AID,
    IN int AIDLen,
    OUT BYTE * tlv,
    IN_OUT int * tlvLen )
```

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>AIDLen</i>	the length of AID data buffer.
<i>tlv</i>	The TLV elements of the requested AID
<i>tlvLen</i>	the length of tlv data buffer.

Returns

RETURN_CODE: Values can be parsed with [device_getIDGStatusCodeString\(\)](#)

11.7.4.25 ctls_retrieveCAPK()

```
int ctls_retrieveCAPK (
    IN BYTE * capk,
    IN int capkLen,
    OUT BYTE * key,
    IN_OUT int * keyLen )
```

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>capkLen</i>	the length of capk data buffer
<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>keyLen</i>	<p>the length of key data buffer</p> <ul style="list-style-type: none"> •

Returns

RETURN_CODE: Values can be parsed with [device_getIDGStatusCodeString\(\)](#)

11.7.4.26 ctls_retrieveCAPKList()

```
int ctls_retrieveCAPKList (
    OUT BYTE * keys,
    IN_OUT int * keysLen )
```

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>keysLen</i>	the length of keys data buffer

Returns

RETURN_CODE: Values can be parsed with [device_getIDGStatusCodeString\(\)](#)

11.7.4.27 ctls_retrieveTerminalData()

```
int ctls_retrieveTerminalData (
    OUT BYTE * tlv,
    IN_OUT int * tlvLen )
```

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>tlvLen</i>	the length of tlv data buffer

Returns

RETURN_CODE: Values can be parsed with [device_getIDGStatusCodeString\(\)](#)

11.7.4.28 ctls_setApplicationData()

```
int ctls_setApplicationData (
    IN BYTE * tlv,
    IN int tlvLen )
```

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>tlvLen</i>	the length of tlv data buffer
---------------	-------------------------------

Returns

RETURN_CODE: Values can be parsed with [device_getIDGStatusCodeString\(\)](#)

11.7.4.29 ctls_setCAPK()

```
int ctls_setCAPK (
    IN BYTE * capk,
    IN int capkLen )
```

Set Certificate Authority Public Key

Sets the CAPK as specified by the CAKey structure

Parameters

<i>capk</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>capkLen</i>	the length of capk data buffer

Returns

RETURN_CODE: Values can be parsed with [device_getIDGStatusCodeString\(\)](#)

11.7.4.30 ctls_setConfigurationGroup()

```
int ctls_setConfigurationGroup (
    IN BYTE * tlv,
    IN int tlvLen )
```

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>tlvLen</i>	the length of tlv data buffer

Returns

RETURN_CODE: Values can be parsed with [device_getIDGStatusCodeString\(\)](#)

11.7.4.31 ctls_setTerminalData()

```
int ctls_setTerminalData (
    IN BYTE * tlv,
    IN int tlvLen )
```

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 [ctls_getConfigurationGroup\(int group\)](#), and deleted with [ctls_removeConfigurationGroup\(int group\)](#). You cannot delete group 0.

Parameters

<i>tlv</i>	TerminalData configuration file
<i>tlvLen</i>	the length of tlv data buffer

Return values

<i>RETURN_CODE</i>	Values can be parsed with device_getIDGStatusCodeString()
--------------------	---

11.7.4.32 ctls_startTransaction()

```
int ctls_startTransaction (
    IN double amount,
    IN double amtOther,
    IN int type,
    IN const int _timeout,
    IN BYTE * tags,
    IN int tagsLen )
```

Start CTLS Transaction Request

Authorizes the CTLS transaction for an ICC card

The tags will be returned in the callback routine.

Parameters

<i>amount</i>	Transaction amount value (tag value 9F02) • SEE IMPORTANT NOTE BELOW
<i>amtOther</i>	Other amount value, if any (tag value 9F03) • SEE IMPORTANT NOTE BELOW
<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 stream. 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>tagsLen</i>	The length of tags data buffer.

>>>>>IMPORTANT: parameters for amount and amtOther MUST BE PASSED AS A DOUBLE VALUE WITH DECIMAL POINT. Example, do not pass 1, but instead pass 1.0 or 1.00. Otherwise, results will be unpredictable

Returns

RETURN_CODE: Values can be parsed with [device_getIDGStatusCodeString\(\)](#) Note: if auto poll is on, it will return the error IDG_P2_STATUS_CODE_COMMAND_NOT_ALLOWED

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) 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

11.7.4.33 device_activateTransaction()

```
int device_activateTransaction (
    IN const int _timeout,
    IN BYTE * tags,
    IN int tagsLen )
```

Activate Transaction Request

Authorizes the transaction for an MSR/CTLS/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. Example, tag 9F02 with amount 0x0000000000100 would be 0x9F0206000000000100 Be sure to include 9F02 (amount)and 9C (transaction type).
<i>tagsLen</i>	The length of tags data buffer.

>>>>>IMPORTANT: parameters for amount and amtOther MUST BE PASSED AS A DOUBLE VALUE WITH DECIMAL POINT. Example, do not pass 1, but instead pass 1.0 or 1.00. Otherwise, results will be unpredictable

Returns

RETURN_CODE: Values can be parsed with [device_getIDGStatusCodeString\(\)](#) Note: if auto poll is on, it will return the error IDG_P2_STATUS_CODE_COMMAND_NOT_ALLOWED

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 device_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

11.7.4.34 device_buzzerOnOff()

```
int device_buzzerOnOff ( )
```

Use this function to make the buzzer beep once

Returns

RETURN_CODE: Values can be parsed with [device_getIDGStatusCodeString\(\)](#)

11.7.4.35 device_cancelTransaction()

```
int device_cancelTransaction ( )
```

Cancel Transaction request.

Returns

RETURN_CODE: Values can be parsed with [device_getIDGStatusCodeString\(\)](#)

11.7.4.36 device_cancelTransactionSilent()

```
int device_cancelTransactionSilent (
    int enable )
```

Cancel Transaction Silent

Cancel transaction with or without showing the LCD message

Parameters

<i>enable</i>	0: With LCD message 1: Without LCD message
---------------	--

Returns

success or error code. Values can be parsed with `device_getIDGStatusCodeString`

11.7.4.37 device_close()

```
int device_close ( )
```

Close the device

Returns

RETURN_CODE: 0: success, 0x0A: failed

11.7.4.38 device_configureButtons()

```
int device_configureButtons (
    IN BYTE done,
    IN BYTE swipe,
    IN BYTE delay )
```

Configures the buttons on the ViVOpay Vendi reader

Parameters

<i>done</i>	0x01: the Done switch is enabled 0x00: the Done switch is disabled
<i>swipe</i>	0x01: the Swipe Card switch is enabled 0x00: the Swipe Card switch is disabled
<i>delay</i>	an unsigned delay value (<= 30) in seconds

Returns

RETURN_CODE: Values can be parsed with [device_getIDGStatusCodeString\(\)](#)

11.7.4.39 device_controlUserInterface()

```
int device_controlUserInterface (
    IN BYTE * values )
```

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) • 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
---------------	--

Returns

RETURN_CODE: Values can be parsed with [device_getIDGStatusCodeString\(\)](#)

11.7.4.40 device_deleteDirectory()

```
int device_deleteDirectory (
    IN char * dirName,
    IN int dirNameLen )
```

Delete Directory This command deletes an empty directory. For NEO 2 devices, it will delete the directory even the directory is not empty.

Parameters

<i>dirName</i>	Complete path of the directory you want to delete. You do not need to specify the root directory. Indicate subdirectories with a forward slash (/). For NEO 2 devices, to delete the root directory, simply pass "" with 0 for dirNameLen.
<i>dirNameLen</i>	Directory Name Length.

Returns

RETURN_CODE: Values can be parsed with [device_getIDGStatusCodeString\(\)](#)

11.7.4.41 device_deleteFile()

```
int device_deleteFile (
    IN char * fileName,
    IN int fileNameLen )
```

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>filenameLen</i>	File Name Length.

Returns

RETURN_CODE: Values can be parsed with [device_getIDGStatusCodeString\(\)](#)

11.7.4.42 device_disableBlueLED()

```
int device_disableBlueLED ( )
```

Stops the blue LEDs on the ViVOPay Vendi reader from flashing in left to right sequence and turns the LEDs off, and contactless function is disabled at the same time

Returns

RETURN_CODE: Values can be parsed with [device_getIDGStatusCodeString\(\)](#)

11.7.4.43 device_enableBlueLED()

```
int device_enableBlueLED (
    IN BYTE * data,
    IN int dataLen )
```

Use this function to control the blue LED behavior on the Vendi reader

Parameters

<i>data</i>	Sequence data Byte 0 (Cycle): 0 = Cycle once, 1 = Repeat Byte 1 (LEDs): LED State Bitmap Byte 2-3 (Duration): Given in multiples of 10 millisecond Byte 4 (LED): LED State Bitmap Byte 5-6 (Duration): Given in multiples of 10 millisecond Byte 7-24 (Additional LED/Durations): Define up to 8 LED and duration pars
-------------	--

LED State Bitmap: Bit 8: Left blue LED, 0 = off, 1 = on Bit 7: Center Blue LED, 0 = off, 1 = on Bit 6: Right Blue LED 0 = off, 1 = on Bit 5: Yellow LED, 0 = off, 1 = on Bit 4: Reserved for future use Bit 3: Reserved for future use Bit 2: Reserved for future use Bit 1: Reserved for future use

Parameters

<i>dataLen</i>	Length of the sequence data: 0 or 4 to 25 bytes
----------------	---

Returns

RETURN_CODE: Values can be parsed with [device_getIDGStatusCodeString\(\)](#)

11.7.4.44 device_enableExternalLCDMessages()

```
int device_enableExternalLCDMessages (
    IN int enableExtLCDMsg )
```

Enable or disable the external LCD message It will turn off the external LCD messages including EMV transactions. (For the users who only need MSR and/or CTLS transactions.) The function only works for VP5300

Parameters

<i>enableExtLCDMsg</i>	1 = ON, 0 = OFF
------------------------	-----------------

Returns

success or error code. Values can be parsed with [device_getIDGStatusCodeString](#)

See also

[ErrorCode](#)

11.7.4.45 device_enableL100PassThrough()

```
int device_enableL100PassThrough (
    int enableL100PassThrough )
```

Enable L100 Pass Through

Enables Pass Through Mode for direct communication to L100 hook up to NEO II device

Parameters

<i>enableL100PassThrough</i>	1 = pass through ON, 0 = pass through OFF
------------------------------	---

Returns

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

11.7.4.46 `device_enableL80PassThrough()`

```
int device_enableL80PassThrough (
    int enableL80PassThrough )
```

Enable L80 Pass Through

Enables Pass Through Mode for direct communication to L80 hook up to NEO II device

Parameters

<i>enableL80PassThrough</i>	1 = pass through ON, 0 = pass through OFF
-----------------------------	---

Returns

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

11.7.4.47 `device_enablePassThrough()`

```
int device_enablePassThrough (
    int enablePassThrough )
```

Enable Pass Through

Enables Pass Through Mode for direct communication with L1 interface (power on icc, send apdu, etc).

Parameters

<i>enablePassThrough</i>	1 = pass through ON, 0 = pass through OFF
--------------------------	---

Returns

RETURN_CODE: Values can be parsed with [device_getIDGStatusCodeString\(\)](#)

11.7.4.48 `device_enableRFAntenna()`

```
int device_enableRFAntenna (
    IN int enableAntenna )
```

Enable or disable the RF Antenna

Parameters

<i>enableAntenna</i>	1 = ON, 0 = OFF
----------------------	-----------------

Returns

success or error code. Values can be parsed with `device_getIDGStatusCodeString`

See also

`ErrorCode`

11.7.4.49 `device_getAudioVolume()`

```
int device_getAudioVolume (
    OUT BYTE * volume )
```

Get Audio Volume This command retrieves the reader's audio volume.

Parameters

<i>Value</i>	0-20, where 0 is silent and 20 is full volume
--------------	---

Returns

RETURN_CODE: Values can be parsed with `device_getIDGStatusCodeString()`

11.7.4.50 `device_getButtonConfiguration()`

```
int device_getButtonConfiguration (
    OUT BYTE * done,
    OUT BYTE * swipe,
    OUT BYTE * delay )
```

Reads the button configuration from the ViVOpay Vendi reader

Parameters

<i>done</i>	0x01: the Done switch is enabled 0x00: the Done switch is disabled
<i>swipe</i>	0x01: the Swipe Card switch is enabled 0x00: the Swipe Card switch is disabled
<i>delay</i>	an unsigned delay value in seconds

Returns

RETURN_CODE: Values can be parsed with `device_getIDGStatusCodeString()`

11.7.4.51 device_getCameraParameters()

```
int device_getCameraParameters (
    OUT BYTE * isAutoFocus,
    OUT BYTE * focalLength )
```

Get Camera Parameters This command is used to get the camera parameters (e.g., auto/fixed focal length as focus).

Parameters

<i>isAutoFocus</i>	0: fixed focus, 1: auto focus
<i>focalLength</i>	focal length Value 0x00-0xFF, where 0x00 is the farthest, 0xFF is nearest. Not used for auto focus.

Returns

RETURN_CODE: Values can be parsed with [device_getIDGStatusCodeString\(\)](#)

11.7.4.52 device_getCurrentDeviceType()

```
int device_getCurrentDeviceType ( )
```

Get current active device type

Returns

: return the device type defined as DEVICE_TYPE in the IDTDef.h

11.7.4.53 device_getDeviceMemoryUsageInfo()

```
int device_getDeviceMemoryUsageInfo (
    OUT int * freeHeapSize,
    OUT int * notFreedBlockCnt,
    OUT int * minEverFreeHeapSize )
```

Get Device Memory Usage Information

Parameters

<i>freeHeapSize</i>	Free Heap Size: Available heap size
<i>notFreedBlockCnt</i>	Memory Not Freed Block Count: Memory in use block count
<i>minEverFreeHeapSize</i>	Minimum Ever Free Heap Size: The lowest ever available heap size

Returns

RETURN_CODE: Values can be parsed with [device_getIDGStatusCodeString\(\)](#)

11.7.4.54 device_getDeviceTreeVersion()

```
int device_getDeviceTreeVersion (
```

```
OUT char * deviceTreeVersion,
IN_OUT int * deviceTreeVersionLen )
```

Polls device for Device Tree Version

Parameters

<i>deviceTreeVersion</i>	Response returned of Device Tree Version
<i>deviceTreeVersionLen</i>	Length of Device Tree Version

Returns

RETURN_CODE: Values can be parsed with [device_getResponseCodeString\(\)](#)

11.7.4.55 device_getDRS()

```
int device_getDRS (
    BYTE * codeDRS,
    int * codeDRSLen )
```

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

Parameters

<i>codeDRSLen</i>	the length of codeDRS
-------------------	-----------------------

Returns

RETURN_CODE: Values can be parsed with [device_getIDGStatusCodeString\(\)](#)

11.7.4.56 device_getFirmwareVersion()

```
int device_getFirmwareVersion (
    OUT char * firmwareVersion )
```

DEPRECATED : please use [device_getFirmwareVersion_Len](#)(OUT char* firmwareVersion, IN_OUT int *firmwareVersionLen)

Polls device for Firmware Version

Parameters

<i>firmwareVersion</i>	Response returned of Firmware Version; needs to have at least 128 bytes of memory
------------------------	---

Returns

RETURN_CODE: Values can be parsed with [device_getIDGStatusCodeString](#)()

11.7.4.57 device_getFirmwareVersion_Len()

```
int device_getFirmwareVersion_Len (
    OUT char * firmwareVersion,
    IN_OUT int * firmwareVersionLen )
```

Polls device for Firmware Version

Parameters

<i>firmwareVersion</i>	Response returned of Firmware Version
<i>firmwareVersionLen</i>	Length of Firmware Version

Returns

RETURN_CODE: Values can be parsed with [device_getIDGStatusCodeString](#)()

11.7.4.58 device_getIDGStatusCodeString()

```
void device_getIDGStatusCodeString (
    IN int returnCode,
    OUT char * description )
```

Review the return code description.

Parameters

<i>returnCode</i>	the response result.
<i>description</i>	

Return values

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

- 0: "no error, beginning task";

- 1: "no response from reader";
- 2: "invalid response data";
- 01: " Incorrect Header Tag";
- 02: " Unknown Command";
- 03: " Unknown Sub-Command";
- 04: " CRC Error in Frame";
- 05: " Incorrect Parameter";
- 06: " Parameter Not Supported";
- 07: " Mal-formatted Data";
- 08: " Timeout";
- 0A: " Failed / NACK";
- 0B: " Command not Allowed";
- 0C: " Sub-Command not Allowed";
- 0D: " Buffer Overflow (Data Length too large for reader buffer)";
- 0E: " User Interface Event";
- 10: " Need clear firmware(apply in boot loader only)";
- 11: " Communication type not supported, VT-1, burst, etc. Need encrypted firmware (apply in boot loader only)";
- 12: " Secure interface is not functional or is in an intermediate state.";
- 13: " Data field is not mod 8";
- 14: " Pad 0x80 not found where expected";
- 15: " Specified key type is invalid";
- 16: " Could not retrieve key from the SAM (InitSecureComm)";
- 17: " Hash code problem";
- 18: " Could not store the key into the SAM (InstallKey)";
- 19: " Frame is too large";
- 1A: " Unit powered up in authentication state but POS must resend the InitSecureComm command";
- 1B: " The EEPROM may not be initialized because SecCommInterface does not make sense";
- 1C: " Problem encoding APDU Module-Specific Status Codes ";
- 20: " Unsupported Index (ILM) SAM Transceiver error - problem communicating with the SAM (Key Mgr)";
- 21: " Unexpected Sequence Counter in multiple frames for single bitmap (ILM)Length error in data returned from the SAM (Key Mgr) - 22: " Improper bit map (ILM)"; - 23: " Request Online Authorization"; - 24: " ViV← OCard3 raw data read successful"; - 25: " Message index not available (ILM) ViVOcomm activate transaction card type (ViVOcomm)"; - 26: " Version Information Mismatch (ILM)"; - 27: " Not sending commands in correct index message index (ILM)"; - 28: " Time out or next expected message not received (ILM)"; - 29: " ILM languages not available for viewing (ILM)"; - 2A: " Other language not supported (ILM)"; - 41: " from 41 to 4F, Module-specific errors for Key Manager";
- 50: " Auto-Switch OK";

- 51: " Auto-Switch failed";
- 70: " Antenna Error 80h Use another card";
- 81: " Insert or swipe card";
- 90: " Data encryption Key does not exist";
- 91: " Data encryption Key KSN exhausted";

11.7.4.59 device_getKeyStatus()

```
int device_getKeyStatus (
    int * newFormat,
    BYTE * status,
    int * statusLen )
```

Get Key Status

Gets the status of loaded keys

Parameters

<i>status</i>	newFormat for Augusta and miniSmartII only 1: new format of key status 0: reserved format for support previous device
<i>status</i>	For L80, L100, Augusta and miniSmartII: When the newFormat is 0, data format as follows. For Augusta and miniSmartII: 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 For L80 and L100: byte 0: PIN DUKPT Key byte 1: PIN Master Key byte 2: Standard PIN Session Key byte 3: Desjardins PIN Session Key byte 4: Account/MSR DUKPT Key, 1 Exists, 0 None, 0xFF STOP, Does not support, always 0 byte 5: Account/ICC DUKPT Key, 1 Exists, 0 None, 0xFF STOP, Does not support, always 0 byte 6: Admin DUKPT Key, 1 Exists, 0 None, 0xFF STOP byte 7: Data DUKPT Key, 1 Exists, 0 None, 0xFF STOP byte 8: MAC DUKPT Key, 1 Exists, 0 None, 0xFF STOP

when the newFormat is 1, data format as follows. [Block Length] [KeyStatusBlock1] [KeyStatusBlock2]...[Key↵StatusBlockN] Where: [Block Length] is 2 bytes, format is Len_L Len_H, is KeyStatusBlock Number [KeyStatus↵BlockX] 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) For NEO2 and SREDKey2: Each unit of three bytes represents one key's parameters (index and slot). Key Name Index (1 byte): 0x14 - LCL-KEK 0x01 - Pin encryption Key (NEO2 only) 0x02 - Data encryption Key 0x05 - MAC DUKPT Key 0x0A - PCI Pairing Key (NEO2 only) Key Slot (2 bytes): Indicate different slots of a certain Key Name Example: slot =5 (0x00 0x05), slot=300 (0x01 0x2C) For BTPay380, slot is always 0 For example, 0x14 0x00 0x00 0x02 0x00 0x00 0x0A 0x00 0x00 will represent [KeyNameIndex=0x14,KeySlot=0x0000], [KeyNameIndex=0x02,Key↵Slot=0x0000] and [KeyNameIndex=0x0A,KeySlot=0x0000]

Parameters

<i>statusLen</i>	the length of status
------------------	----------------------

Returns

RETURN_CODE: Values can be parsed with `device_getIDGStatusCodeString`

11.7.4.60 device_getL100PassThroughMode()

```
int device_getL100PassThroughMode ( )
```

Get L100 Pass Through Mode

Get current Pass Through Mode for direct communication to L100 hook up to NEO II device

Returns

RETURN_CODE: return 1 if L100 Pass Through Mode is TRUE, 0 if L100 Pass Through Mode is FALSE

11.7.4.61 device_getL80PassThroughMode()

```
int device_getL80PassThroughMode ( )
```

Get L80 Pass Through Mode

Get current Pass Through Mode for direct communication to L80 hook up to NEO II device

Returns

RETURN_CODE: return 1 if L80 Pass Through Mode is TRUE, 0 if L80 Pass Through Mode is FALSE

11.7.4.62 device_getMerchantRecord()

```
int device_getMerchantRecord (
    IN int index,
    OUT BYTE * record )
```

DEPRECATED : please use `device_getMerchantRecord_Len(IN int index, OUT BYTE * record, IN_OUT int *recordLen)`

Get Merchant Record

Gets the merchant record for the device.

Parameters

<i>index</i>	Merchant Record index, valid values 1-6
<i>record;</i>	needs to have at least 99 bytes of memory response data from reader. Merchant Record Index: 1 byte enabled: 1 byte Merchant ID: 32 bytes Length of Merchant URL: 1 byte Merchant URL: 64 bytes

Returns

success or error code. Values can be parsed with `device_getIDGStatusCodeString()`

See also[ErrorCode](#)**11.7.4.63 device_getMerchantRecord_Len()**

```
int device_getMerchantRecord_Len (
    IN int index,
    OUT BYTE * record,
    IN_OUT int * recordLen )
```

Get Merchant Record

Gets the merchant record for the device.

Parameters

<i>index</i>	Merchant Record index, valid values 1-6
<i>record</i>	response data from reader. Merchant Record Index: 1 byte enabled: 1 byte Merchant ID: 32 bytes Length of Merchant URL: 1 byte Merchant URL: 64 bytes
<i>recordLen</i>	Length of record

Returns

success or error code. Values can be parsed with [device_getIDGStatusCodeString\(\)](#)

See also[ErrorCode](#)**11.7.4.64 device_getNEOAltDevice()**

```
int device_getNEOAltDevice ( )
```

Get the alternative device type for the NEO 2 or 3 readers

Returns

RETURN_CODE: The alternative device type of the NEO 2/3 readers.

11.7.4.65 device_getResponseCodeString()

```
void device_getResponseCodeString (
    IN int returnCode,
    OUT char * description )
```

Review the return code description.

Parameters

<i>returnCode</i>	the response result.
<i>description</i>	

Return values

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

- 0: "no error, beginning task";
- 1: "no response from reader";
- 2: "invalid response data";
- 3: "time out for task or CMD";
- 4: "wrong parameter";
- 5: "SDK is doing MSR or ICC task";
- 6: "SDK is doing PINPad task";
- 7: "SDK is doing CTLS task";
- 8: "SDK is doing EMV task";
- 9: "SDK is doing Other task";
- 10: "err response or data";
- 11: "no reader attached";
- 12: "mono audio is enabled";
- 13: "did connection";
- 14: "audio volume is too low";
- 15: "task or CMD be canceled";
- 16: "UF wrong string format";
- 17: "UF file not found";
- 18: "UF wrong file format";
- 19: "Attempt to contact online host failed";
- 20: "Attempt to perform RKI failed";
- 22: "Buffer size is not enough";
- 0X300: "Key Type(TDES) of Session Key is not same as the related Master Key.";
- 0X400: "Related Key was not loaded.";
- 0X500: "Key Same.";
- 0X501: "Key is all zero";
- 0X502: "TR-31 format error";
- 0X702: "PAN is Error Key.";

- 0X705: "No Internal MSR PAN (or Internal MSR PAN is erased timeout)";
- 0X0C01: "Incorrect Frame Tag";
- 0X0C02: "Incorrect Frame Type";
- 0X0C03: "Unknown Frame Type";
- 0X0C04: "Unknown Command";
- 0X0C05: "Unknown Sub-Command";
- 0X0C06: "CRC Error";
- 0X0C07: "Failed";
- 0X0C08: "Timeout";
- 0X0C0A: "Incorrect Parameter";
- 0X0C0B: "Command Not Supported";
- 0X0C0C: "Sub-Command Not Supported";
- 0X0C0D: "Parameter Not Supported / Status Abort Command";
- 0X0C0F: "Sub-Command Not Allowed";
- 0X0D01: "Incorrect Header Tag";
- 0X0D02: "Unknown Command";
- 0X0D03: "Unknown Sub-Command";
- 0X0D04: "CRC Error in Frame";
- 0X0D05: "Incorrect Parameter";
- 0X0D06: "Parameter Not Supported";
- 0X0D07: "Mal-formatted Data";
- 0X0D08: "Timeout";
- 0X0D0A: "Failed / NACK";
- 0X0D0B: "Command not Allowed";
- 0X0D0C: "Sub-Command not Allowed";
- 0X0D0D: "Buffer Overflow (Data Length too large for reader buffer)";
- 0X0D0E: "User Interface Event";
- 0X0D11: "Communication type not supported, VT-1, burst, etc.";
- 0X0D12: "Secure interface is not functional or is in an intermediate state.";
- 0X0D13: "Data field is not mod 8";
- 0X0D14: "Pad - 0X80 not found where expected";
- 0X0D15: "Specified key type is invalid";
- 0X0D1: "Could not retrieve key from the SAM(InitSecureComm)";
- 0X0D17: "Hash code problem";
- 0X0D18: "Could not store the key into the SAM(InstallKey)";
- 0X0D19: "Frame is too large";

- 0X0D1A: "Unit powered up in authentication state but POS must resend the InitSecureComm command";
- 0X0D1B: "The EEPROM may not be initialized because SecCommInterface does not make sense";
- 0X0D1C: "Problem encoding APDU";
- 0X0D20: "Unsupported Index(ILM) SAM Transceiver error - problem communicating with the SAM(Key Mgr)";
- 0X0D2: "Unexpected Sequence Counter in multiple frames for single bitmap(ILM) Length error in data returned from the SAM(Key Mgr)";
- 0X0D22: "Improper bit map(ILM)";
- 0X0D23: "Request Online Authorization";
- 0X0D24: "ViVOCard3 raw data read successful";
- 0X0D25: "Message index not available(ILM) ViVOcomm activate transaction card type(ViVOcomm)";
- 0X0D26: "Version Information Mismatch(ILM)";
- 0X0D27: "Not sending commands in correct index message index(ILM)";
- 0X0D28: "Time out or next expected message not received(ILM)";
- 0X0D29: "ILM languages not available for viewing(ILM)";
- 0X0D2A: "Other language not supported(ILM)";
- 0X0D41: "Unknown Error from SAM";
- 0X0D42: "Invalid data detected by SAM";
- 0X0D43: "Incomplete data detected by SAM";
- 0X0D44: "Reserved";
- 0X0D45: "Invalid key hash algorithm";
- 0X0D46: "Invalid key encryption algorithm";
- 0X0D47: "Invalid modulus length";
- 0X0D48: "Invalid exponent";
- 0X0D49: "Key already exists";
- 0X0D4A: "No space for new RID";
- 0X0D4B: "Key not found";
- 0X0D4C: "Crypto not responding";
- 0X0D4D: "Crypto communication error";
- 0X0D4E: "Module-specific error for Key Manager";
- 0X0D4F: "All key slots are full (maximum number of keys has been installed)";
- 0X0D50: "Auto-Switch OK";
- 0X0D51: "Auto-Switch failed";
- 0X0D90: "Account DUKPT Key not exist";
- 0X0D91: "Account DUKPT Key KSN exhausted";
- 0X0D00: "This Key had been loaded.";
- 0X0E00: "Base Time was loaded.";

- 0X0F00: "Encryption Or Decryption Failed.";
- 0X1000: "Battery Low Warning (It is High Priority Response while Battery is Low.)";
- 0X1800: "Send 'Cancel Command' after send 'Get Encrypted PIN' & 'Get Numeric' & 'Get Amount'; - 0↵
X1900: "Press 'Cancel' key after send 'Get Encrypted PIN' & 'Get Numeric' & 'Get Amount';
- 0X30FF: "Security Chip is not connect";
- 0X3000: "Security Chip is deactivation & Device is In Removal Legally State.";
- 0X3101: "Security Chip is activation & Device is In Removal Legally State.";
- 0X5500: "No Admin DUKPT Key.";
- 0X5501: "Admin DUKPT Key STOP.";
- 0X5502: "Admin DUKPT Key KSN is Error.";
- 0X5503: "Get Authentication Code1 Failed.";
- 0X5504: "Validate Authentication Code Error.";
- 0X5505: "Encrypt or Decrypt data failed.";
- 0X5506: "Not Support the New Key Type.";
- 0X5507: "New Key Index is Error.";
- 0X5508: "Step Error.";
- 0X5509: "KSN Error";
- 0X550A: "MAC Error.";
- 0X550B: "Key Usage Error.";
- 0X550C: "Mode Of Use Error.";
- 0X550F: "Other Error.";
- 0X6000: "Save or Config Failed / Or Read Config Error.";
- 0X6200: "No Serial Number.";
- 0X6900: "Invalid Command - Protocol is right, but task ID is invalid.";
- 0X6A01: "Unsupported Command - Protocol and task ID are right, but command is invalid - In this State";
- 0X6A00: "Unsupported Command - Protocol and task ID are right, but command is invalid.";
- 0X6B00: "Unknown parameter in command - Protocol task ID and command are right, but parameter is invalid.";
- 0X6C00: "Unknown parameter in command - Protocol task ID and command are right, but length is out of the requirement.";
- 0X7200: "Device is suspend (MKSK suspend or press password suspend).";
- 0X7300: "PIN DUKPT is STOP (21 bit 1).";
- 0X7400: "Device is Busy.";
- 0XE100: "Can not enter sleep mode";
- 0XE200: "File has existed";
- 0XE300: "File has not existed";
- 0XE313: "IO line low -- Card error after session start";

- 0XE400: "Open File Error";
- 0XE500: "SmartCard Error";
- 0XE600: "Get MSR Card data is error";
- 0XE700: "Command time out";
- 0XE800: "File read or write is error";
- 0XE900: "Active 1850 error!";
- 0XEA00: "Load bootloader error";
- 0XEF00: "Protocol Error- STX or ETX or check error.";
- 0XEB00: "Picture is not exist";
- 0X2C02: "No Microprocessor ICC seated";
- 0X2C06: "no card seated to request ATR";
- 0X2D01: "Card Not Supported, ";
- 0X2D03: "Card Not Supported, wants CRC";
- 0X690D: "Command not supported on reader without ICC support";
- 0X8100: "ICC error time out on power-up";
- 0X8200: "invalid TS character received - Wrong operation step";
- 0X8300: "Decode MSR Error";
- 0X8400: "TriMagII no Response";
- 0X8500: "No Swipe MSR Card";
- 0X8510: "No Financial Card";
- 0X8600: "Unsupported F, D, or combination of F and D";
- 0X8700: "protocol not supported EMV TD1 out of range";
- 0X8800: "power not at proper level";
- 0X8900: "ATR length too long";
- 0X8B01: "EMV invalid TA1 byte value";
- 0X8B02: "EMV TB1 required";
- 0X8B03: "EMV Unsupported TB1 only 00 allowed";
- 0X8B04: "EMV Card Error, invalid BWI or CWI";
- 0X8B06: "EMV TB2 not allowed in ATR";
- 0X8B07: "EMV TC2 out of range";
- 0X8B08: "EMV TC2 out of range";
- 0X8B09: "per EMV96 TA3 must be > - 0XF";
- 0X8B10: "ICC error on power-up";
- 0X8B11: "EMV T=1 then TB3 required";
- 0X8B12: "Card Error, invalid BWI or CWI";
- 0X8B13: "Card Error, invalid BWI or CWI";

- 0X8B17: "EMV TC1/TB3 conflict-";
- 0X8B20: "EMV TD2 out of range must be T=1";
- 0X8C00: "TCK error";
- 0XA304: "connector has no voltage setting";
- 0XA305: "ICC error on power-up invalid (SBLK(IFSD) exchange";
- 0XE301: "ICC error after session start";
- 0XFF00: "Request to go online";
- 0XFF01: "EMV: Accept the offline transaction";
- 0XFF02: "EMV: Decline the offline transaction";
- 0XFF03: "EMV: Accept the online transaction";
- 0XFF04: "EMV: Decline the online transaction";
- 0XFF05: "EMV: Application may fallback to magstripe technology";
- 0XFF06: "EMV: ICC detected tah the conditions of use are not satisfied";
- 0XFF07: "EMV: ICC didn't accept transaction";
- 0XFF08: "EMV: Transaction was cancelled";
- 0XFF09: "EMV: Application was not selected by kernel or ICC format error or ICC missing data error";
- 0XFF0A: "EMV: Transaction is terminated";
- 0XFF0B: "EMV: Other EMV Error";
- 0XFFFF: "NO RESPONSE";
- 0XF002: "ICC communication timeout";
- 0XF003: "ICC communication Error";
- 0XF00F: "ICC Card Seated and Highest Priority, disable MSR work request";
- 0XF200: "AID List / Application Data is not exist";
- 0XF201: "Terminal Data is not exist";
- 0XF202: "TLV format is error";
- 0XF203: "AID List is full";
- 0XF204: "Any CA Key is not exist";
- 0XF205: "CA Key RID is not exist";
- 0XF206: "CA Key Index it not exist";
- 0XF207: "CA Key is full";
- 0XF208: "CA Key Hash Value is Error";
- 0XF209: "Transaction format error";
- 0XF20A: "The command will not be processing";
- 0XF20B: "CRL is not exist";
- 0XF20C: "CRL number exceed max number";
- 0XF20D: "Amount, Other Amount, Trasaction Type are missing";

- 0XF20E: "The Identification of algorithm is mistake";
- 0XF20F: "No Financial Card";
- 0XF210: "In Encrypt Result state, TLV total Length is greater than Max Length";
- 0X1001: "INVALID ARG";
- 0X1002: "FILE_OPEN_FAILED";
- 0X1003: "FILE OPERATION_FAILED";
- 0X2001: "MEMORY_NOT_ENOUGH";
- 0X3002: "SMARTCARD_FAIL";
- 0X3003: "SMARTCARD_INIT_FAILED";
- 0X3004: "FALLBACK_SITUATION";
- 0X3005: "SMARTCARD_ABSENT";
- 0X3006: "SMARTCARD_TIMEOUT";
- 0X3012: "EMV_RESULT_CODE_MSR_CARD_ERROR_FALLBACK";
- 0X5001: "EMV_PARSING_TAGS_FAILED";
- 0X5002: "EMV_DUPLICATE_CARD_DATA_ELEMENT";
- 0X5003: "EMV_DATA_FORMAT_INCORRECT";
- 0X5004: "EMV_NO_TERM_APP";
- 0X5005: "EMV_NO_MATCHING_APP";
- 0X5006: "EMV_MISSING_MANDATORY_OBJECT";
- 0X5007: "EMV_APP_SELECTION_RETRY";
- 0X5008: "EMV_GET_AMOUNT_ERROR";
- 0X5009: "EMV_CARD_REJECTED";
- 0X5010: "EMV_AIP_NOT_RECEIVED";
- 0X5011: "EMV_AFL_NOT_RECEIVED";
- 0X5012: "EMV_AFL_LEN_OUT_OF_RANGE";
- 0X5013: "EMV_SFI_OUT_OF_RANGE";
- 0X5014: "EMV_AFL_INCORRECT";
- 0X5015: "EMV_EXP_DATE_INCORRECT";
- 0X5016: "EMV_EFF_DATE_INCORRECT";
- 0X5017: "EMV_ISS_COD_TBL_OUT_OF_RANGE";
- 0X5018: "EMV_CRYPTOGram_TYPE_INCORRECT";
- 0X5019: "EMV_PSE_NOT_SUPPORTED_BY_CARD";
- 0X5020: "EMV_USER_SELECTED_LANGUAGE";
- 0X5021: "EMV_SERVICE_NOT_ALLOWED";
- 0X5022: "EMV_NO_TAG_FOUND";
- 0X5023: "EMV_CARD_BLOCKED";

- 0X5024: "EMV_LEN_INCORRECT";
- 0X5025: "CARD_COM_ERROR";
- 0X5026: "EMV_TSC_NOT_INCREASED";
- 0X5027: "EMV_HASH_INCORRECT";
- 0X5028: "EMV_NO_ARC";
- 0X5029: "EMV_INVALID_ARC";
- 0X5030: "EMV_NO_ONLINE_COMM";
- 0X5031: "TRAN_TYPE_INCORRECT";
- 0X5032: "EMV_APP_NO_SUPPORT";
- 0X5033: "EMV_APP_NOT_SELECT";
- 0X5034: "EMV_LANG_NOT_SELECT";
- 0X5035: "EMV_NO_TERM_DATA";
- 0X5039: "EMV_PIN_ENTRY_TIMEOUT";
- 0X6001: "CVM_TYPE_UNKNOWN";
- 0X6002: "CVM_AIP_NOT_SUPPORTED";
- 0X6003: "CVM_TAG_8E_MISSING";
- 0X6004: "CVM_TAG_8E_FORMAT_ERROR";
- 0X6005: "CVM_CODE_IS_NOT_SUPPORTED";
- 0X6006: "CVM_COND_CODE_IS_NOT_SUPPORTED";
- 0X6007: "NO_MORE_CVM";
- 0X6008: "PIN_BYPASSED_BEFORE";
- 0X7001: "PK_BUFFER_SIZE_TOO_BIG";
- 0X7002: "PK_FILE_WRITE_ERROR";
- 0X7003: "PK_HASH_ERROR";
- 0X8001: "NO_CARD_HOLDER_CONFIRMATION";
- 0X8002: "GET_ONLINE_PIN";
- 0XD000: "Data not exist";
- 0XD001: "Data access error";
- 0XD100: "RID not exist";
- 0XD101: "RID existed";
- 0XD102: "Index not exist";
- 0XD200: "Maximum exceeded";
- 0XD201: "Hash error";
- 0XD205: "System Busy";
- 0X0E01: "Unable to go online";
- 0X0E02: "Technical Issue";

- 0X0E03: "Declined";
- 0X0E04: "Issuer Referral transaction";
- 0X0F01: "Decline the online transaction";
- 0X0F02: "Request to go online";
- 0X0F03: "Transaction is terminated";
- 0X0F05: "Application was not selected by kernel or ICC format error or ICC missing data error";
- 0X0F07: "ICC didn't accept transaction";
- 0X0F0A: "Application may fallback to magstripe technology";
- 0X0F0C: "Transaction was cancelled";
- 0X0F0D: "Timeout";
- 0X0F0F: "Other EMV Error";
- 0X0F10: "Accept the offline transaction";
- 0X0F11: "Decline the offline transaction";
- 0X0F21: "ICC detected tah the conditions of use are not satisfied";
- 0X0F22: "No app were found on card matching terminal configuration";
- 0X0F23: "Terminal file does not exist";
- 0X0F24: "CAPK file does not exist";
- 0X0F25: "CRL Entry does not exist";
- 0X0FFE: "code when blocking is disabled";
- 0X0FFF: "code when command is not applicable on the selected device";
- 0XF005: "ICC Encrypted C-APDU Data Structure Length Error Or Format Error.";
- 0XBBE0: "CM100 Success";
- 0XBBE1: "CM100 Parameter Error";
- 0XBBE2: "CM100 Low Output Buffer";
- 0XBBE3: "CM100 Card Not Found";
- 0XBBE4: "CM100 Collision Card Exists";
- 0XBBE5: "CM100 Too Many Cards Exist";
- 0XBBE6: "CM100 Saved Data Does Not Exist";
- 0XBBE8: "CM100 No Data Available";
- 0XBBE9: "CM100 Invalid CID Returned";
- 0XBBEA: "CM100 Invalid Card Exists";
- 0BBEC: "CM100 Command Unsupported";
- 0BBED: "CM100 Error In Command Process";
- 0BBEE: "CM100 Invalid Command";
- 0X9031: "Unknown command";
- 0X9032: "Wrong parameter (such as the length of the command is incorrect)";

- 0X9038: "Wait (the command couldnt be finished in BWT)";
- 0X9039: "Busy (a previously command has not been finished)";
- 0X903A: "Number of retries over limit";
- 0X9040: "Invalid Manufacturing system data";
- 0X9041: "Not authenticated";
- 0X9042: "Invalid Master DUKPT Key";
- 0X9043: "Invalid MAC Key";
- 0X9044: "Reserved for future use";
- 0X9045: "Reserved for future use";
- 0X9046: "Invalid DATA DUKPT Key";
- 0X9047: "Invalid PIN Pairing DUKPT Key";
- 0X9048: "Invalid DATA Pairing DUKPT Key";
- 0X9049: "No nonce generated";
- 0X9949: "No GUID available. Perform getVersion first.";
- 0X9950: "MAC Calculation unsuccessful. Check BDK value.";
- 0X904A: "Not ready";
- 0X904B: "Not MAC data";
- 0X9050: "Invalid Certificate";
- 0X9051: "Duplicate key detected";
- 0X9052: "AT checks failed";
- 0X9053: "TR34 checks failed";
- 0X9054: "TR31 checks failed";
- 0X9055: "MAC checks failed";
- 0X9056: "Firmware download failed";
- 0X9060: "Log is full";
- 0X9061: "Removal sensor unengaged";
- 0X9062: "Any hardware problems";
- 0X9070: "ICC communication timeout";
- 0X9071: "ICC data error (such check sum error)";
- 0X9072: "Smart Card not powered up";

11.7.4.66 device_getSDKWaitTime()

```
int device_getSDKWaitTime ( )
```

Get SDK Wait Time

Get the SDK wait time for transactions

Returns

SDK wait time in seconds

11.7.4.67 device_getTamperStatus()

```
int device_getTamperStatus (
    OUT int * isTampered )
```

Get Tamper Status This command check the tamper status for NEO 2 readers.

Parameters

<i>isTampered</i>	1: the reader is tampered. 0: the reader is not tampered.
-------------------	---

Returns

RETURN_CODE: Values can be parsed with [device_getIDGStatusCodeString\(\)](#)

11.7.4.68 device_getThreadStackSize()

```
int device_getThreadStackSize ( )
```

Get Thread Stack Size

Get the stack size setting for newly created threads

Returns

Thread Stack Size

11.7.4.69 device_getTransactionResults()

```
int device_getTransactionResults (
    IDTMSRData * cardData )
```

Get Transaction Results Gets the transaction results when the reader is functioning in "Auto Poll" mode

Parameters

<i>cardData</i>	The transaction results
-----------------	-------------------------

Returns

success or error code. Values can be parsed with [device_getIDGStatusCodeString](#)

See also

[ErrorCode](#)

11.7.4.70 device_init()

```
int device_init ( )
```

Initial the device by USB

It will detect the device and trying connect.

The connect status can be checked by [device_isConnected\(\)](#).

Note: after the function returns success, the function [device_setCurrentDevice\(\)](#) has to be called to set the device type.

Returns

RETURN_CODE: Values can be parsed with [device_getIDGStatusCodeString\(\)](#)

11.7.4.71 device_isAttached()

```
int device_isAttached (
    int deviceType )
```

Check if the device is attached to the USB port The function [device_init\(\)](#) must be called before this function.

Parameters

<i>deviceType,the</i>	device type of the USB device
-----------------------	-------------------------------

Returns

1 if the device is attached, or 0 if the device is not attached

11.7.4.72 device_isConnected()

```
int device_isConnected ( )
```

Check the device connected status

Returns

DEVICE_DISCONNECT=0, or DEVICE_CONNECTED = 1

11.7.4.73 device_lcdDisplayClear()

```
int device_lcdDisplayClear ( )
```

Use this function to clear the LCD display

Returns

RETURN_CODE: Values can be parsed with [device_getIDGStatusCodeString\(\)](#)

11.7.4.74 device_lcdDisplayLine1Message()

```
int device_lcdDisplayLine1Message (  
    IN BYTE * message,  
    IN int messageLen )
```

Use this function to display text on the LCD display. On the Vendi reader the LCD is a 2-line character display.

Parameters

<i>message</i>	Valid messages for the first line of text are between 1 and 16 printable characters long. If the text message is greater than 16 bytes but not more than 32 bytes, byte 17 and onward are displayed as a second row of text. All messages are left justified on the LCD display.
<i>messageLen</i>	Length of the message: 1 to 32 bytes

Returns

RETURN_CODE: Values can be parsed with [device_getIDGStatusCodeString\(\)](#)

11.7.4.75 device_lcdDisplayLine2Message()

```
int device_lcdDisplayLine2Message (  
    IN BYTE * message,  
    IN int messageLen )
```

Use this function to display the message on line 2 of the LCD display. On the Vendi reader the LCD is a 2-line character display.

Parameters

<i>message</i>	Valid messages are between 1 and 16 printable characters long. All messages are left justified on the LCD display.
<i>messageLen</i>	Length of the message: 1 to 16 bytes

Returns

RETURN_CODE: Values can be parsed with [device_getIDGStatusCodeString\(\)](#)

11.7.4.76 device_listDirectory()

```
int device_listDirectory (  
    IN char * directoryName,  
    IN int directoryNameLen,
```

```

    IN int recursive,
    IN int onSD,
    OUT char * directory,
    IN_OUT int * directoryLen )

```

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>directoryNameLen</i>	Directory Name Length. If null, root directory is listed
<i>recursive</i>	Included sub-directories
<i>onSD</i>	0: use internal memory, 1: use SD card The returned directory information The returned directory information length

Returns

RETURN_CODE: Values can be parsed with [device_getIDGStatusCodeString\(\)](#)

11.7.4.77 device_loadCertCA()

```

int device_loadCertCA (
    IN BYTE CertType,
    IN BYTE * CACertData,
    IN int CACertDataLen )

```

Use this function to load CA Cert (Intermediate Certificate)

Parameters

<i>CertType</i>	Application CA: 0x00 TLS CA: 0x01
<i>CACertData</i>	Multi bytes CA cert data
<i>CACertDataLen</i>	Length of the CACertData

Returns

RETURN_CODE: Values can be parsed with [device_getIDGStatusCodeString\(\)](#)

11.7.4.78 device_outputLog()

```

int device_outputLog (
    IN char * filename,
    IN int filenameLen )

```

Output Log

Save the log to a file

Parameters

<i>filename</i>	the file name including the path
<i>filenameLen</i>	the length of filename

Returns

RETURN_CODE: Values can be parsed with [device_getIDGStatusCodeString\(\)](#)

11.7.4.79 device_pingDevice()

```
int device_pingDevice ( )
```

Ping Device

Pings the reader. If connected, returns success. Otherwise, returns timeout.

Returns

RETURN_CODE: Values can be parsed with [device_getIDGStatusCodeString\(\)](#)

11.7.4.80 device_playAudio()

```
int device_playAudio (
    IN char * fileName,
    IN int fileNameLen,
    IN int onSD )
```

Play Audio This command plays an audio file loaded from the inserted SD card. The VP6800 supports 16bit PCM format .WAV files.

Parameters

<i>filename</i>	Complete path and file name of the wav file you want to play. You do not need to specify the root directory. Indicate subdirectories with a forward slash (/).
<i>filenameLen</i>	File Name Length. Maximum file name length is 30.
<i>onSD</i>	0: use internal memory (the Maximum audio file in Flash is 5M and only 2 audio file is supported), 1: use SD card

Returns

RETURN_CODE: Values can be parsed with [device_getIDGStatusCodeString\(\)](#)

11.7.4.81 device_pollForToken()

```
int device_pollForToken (
    IN int timeout,
    OUT BYTE * respData,
    IN_OUT int * respDataLen )
```

Poll for Token

Polls for a PICC

Parameters

<i>timeout</i>	timeout in milliseconds, must be multiple of 10 milliseconds. 30, 120, 630, or 1150 for example.
----------------	--

Parameters

<i>respData</i>	Response data will be stored in respData. 1 byte of card type, and the Serial Number (or the UID) of the PICC if available.
<i>respDataLen</i>	Length of systemCode.

Returns

RETURN_CODE: Values can be parsed with [device_getIDGStatusCodeString\(\)](#)

11.7.4.82 device_queryFile()

```
int device_queryFile (
    IN char * directoryName,
    IN int directoryNameLen,
    IN char * fileName,
    IN int fileNameLen,
    OUT int * isExist,
    OUT BYTE * timeStamp,
    IN_OUT int * timeStampLen,
    OUT char * fileSize,
    IN_OUT int * fileSizeLen )
```

Query File This command checks if the specified file exists in NAND Flash..

Parameters

<i>directoryName</i>	Directory name string. No longer than 32 bytes. ASCII string, terminated by 0x00.
<i>directoryNameLen</i>	Directory Name Length.
<i>fileName</i>	File name string. No longer than 32 bytes. ASCII string, terminated by 0x00.
<i>fileNameLen</i>	File Name Length.
<i>isExist</i>	File exists: 1, File not exists 0.
<i>timeStamp</i>	Latest time stamp of the file. 6 bytes BCD code if the file exists.
<i>timeStampLen</i>	Length of timeStamp. 6 if the file exists, 0 if the file does not exist.
<i>fileSize</i>	Zero-terminated ASCII string of the file size.
<i>fileSizeLen</i>	Length of fileSize.

Returns

RETURN_CODE: Values can be parsed with [device_getIDGStatusCodeString\(\)](#)

11.7.4.83 device_readFileFromSD()

```
int device_readFileFromSD (
    IN char * directoryName,
    IN int directoryNameLen,
    IN char * fileName,
    IN int fileNameLen,
    IN int startingOffset,
```

```

    IN int numBytes,
    OUT BYTE * fileData,
    IN_OUT int * fileDataLen )

```

Read Out File from SD Card This command retrieves a file from the SD card of the reader.

Parameters

<i>directoryName</i>	Directory name string. No longer than 32 bytes. ASCII string, terminated by 0x00.
<i>directoryNameLen</i>	Directory Name Length.
<i>fileName</i>	File name string. No longer than 32 bytes. ASCII string, terminated by 0x00.
<i>fileNameLen</i>	File Name Length.
<i>startingOffset</i>	Starting offset in the file to retrieve
<i>numBytes</i>	Number of bytes of file data to retrieve
<i>fileData</i>	the file data read from the SD card
<i>fileDataLen</i>	Length of fileData.

Returns

RETURN_CODE: Values can be parsed with [device_getIDGStatusCodeString\(\)](#)

11.7.4.84 device_rebootDevice()

```
int 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_getIDGStatusCodeString\(\)](#)

11.7.4.85 device_registerCameraCallBk()

```
void device_registerCameraCallBk (
    pCMR_callback pCMRf )
```

To register the camera callback function to get the image data. (Pass NULL to disable the callback.)

11.7.4.86 device_registerCardStatusFrontSwitchCallBk()

```
void device_registerCardStatusFrontSwitchCallBk (
    pCSFS_callback pCSFSf )
```

To register the card status and front switch callback function to get status. (Pass NULL to disable the callback.)

11.7.4.87 device_registerFWCallBk()

```
void device_registerFWCallBk (
    pFW_callback pFWf )
```

To register the firmware update callback function to get the firmware update processing response. (Pass NULL to disable the callback.)

11.7.4.88 device_registerRKICallBk()

```
void device_registerRKICallBk (
    pRKI_callback pRKIf )
```

To register the RKI callback function to get the RKI status. (Pass NULL to disable the callback.)

11.7.4.89 device_rrcConnect()

```
int device_rrcConnect ( )
```

Use this function to allow a host to establish an RRC connection to a reader

Returns

RETURN_CODE: Values can be parsed with [device_getIDGStatusCodeString\(\)](#)

11.7.4.90 device_rrcDisconnect()

```
int device_rrcDisconnect ( )
```

Use this function to allow a host to terminate its existing RRC connection to a reader

Returns

RETURN_CODE: Values can be parsed with [device_getIDGStatusCodeString\(\)](#)

11.7.4.91 device_rrcDownloadApp()

```
int device_rrcDownloadApp (
    IN char * zipFileName,
    IN int zipFileNameLen,
    IN char * appName,
    IN int appNameLen )
```

Use this function to transfer a compressed application file from a host to the reader, extracts it, and performs signature verification on its contents.

Parameters

<i>zipFileName</i>	The zip file name
<i>zipFileNameLen</i>	Length of the zipFileName
<i>AppName</i>	Application Name in ASCII
<i>AppNameLen</i>	Length of Application Name between 2 to 128

Returns

RETURN_CODE: Values can be parsed with [device_getIDGStatusCodeString\(\)](#)

11.7.4.92 device_rrcInstallApp()

```
int device_rrcInstallApp (
    IN char * appName,
    IN int  appNameLen )
```

Use this function to install an application on a reader device

Parameters

<i>AppName</i>	Application Name in ASCII
<i>AppNameLen</i>	Length of Application Name between 2 to 128

Returns

RETURN_CODE: Values can be parsed with [device_getIDGStatusCodeString\(\)](#)

11.7.4.93 device_rrcRunApp()

```
int device_rrcRunApp (
    IN char * appName,
    IN int  appNameLen )
```

Use this function to run an application on a reader device

Parameters

<i>AppName</i>	Application Name in ASCII
<i>AppNameLen</i>	Length of Application Name between 2 to 128

Returns

RETURN_CODE: Values can be parsed with [device_getIDGStatusCodeString\(\)](#)

11.7.4.94 device_rrcUninstallApp()

```
int device_rrcUninstallApp (
    IN char * appName,
    IN int  appNameLen )
```

Use this function to uninstall an application on a reader device

Parameters

<i>AppName</i>	Application Name in ASCII
<i>AppNameLen</i>	Length of Application Name between 2 to 128

Returns

RETURN_CODE: Values can be parsed with [device_getIDGStatusCodeString\(\)](#)

11.7.4.95 device_SendDataCommandNEO()

```
int device_SendDataCommandNEO (
    IN int cmd,
    IN int subCmd,
    IN BYTE * data,
    IN int dataLen,
    OUT BYTE * response,
    IN_OUT int * respLen )
```

Send a Command to NEO device

Sends a command to the NEO device .

Parameters

<i>cmd</i>	command to execute.
<i>subCmd,sub</i>	command to execute.
<i>data</i>	buffer of NEO command data.
<i>dataLen,the</i>	length of the buffer data.
<i>response</i>	Response data
<i>respLen,the</i>	length of Response data

Returns

RETURN_CODE: Values can be parsed with [device_getIDGStatusCodeString\(\)](#)

11.7.4.96 device_setAudioVolume()

```
int device_setAudioVolume (
    IN BYTE volume )
```

Set Audio Volume This command sets the reader's audio volume.

Parameters

<i>Value</i>	0-20, where 0 is silent and 20 is full volume
--------------	---

Returns

RETURN_CODE: Values can be parsed with [device_getIDGStatusCodeString\(\)](#)

11.7.4.97 device_setBurstMode()

```
int device_setBurstMode (
    IN BYTE mode )
```

Send Burst Mode

Sets the burst mode for the device.

Parameters

<i>mode</i>	0 = OFF, 1 = Always On, 2 = Auto Exit
-------------	---------------------------------------

Returns

success or error code. Values can be parsed with [device_getIDGStatusCodeString](#)

See also

[ErrorCode](#)

11.7.4.98 device_setCameraParameters()

```
int device_setCameraParameters (
    IN BYTE isAutoFocus,
    IN BYTE focalLength )
```

Set Camera Parameters This command is used to set the camera parameters (e.g., auto/fixed focal length as focus).

Parameters

<i>isAutoFocus</i>	0: fixed focus, 1: auto focus
<i>focalLength</i>	focal length Value 0x00-0xFF, where 0x00 is the farthest, 0xFF is nearest. Not used for auto focus.

Returns

RETURN_CODE: Values can be parsed with [device_getIDGStatusCodeString\(\)](#)

11.7.4.99 device_setCancelTransactionMode()

```
int device_setCancelTransactionMode (
    int mode )
```

Set Cancel Transaction Mode

Set the cancel transaction mode to be with or without LCD message

Parameters

<i>mode</i>	0: With LCD message 1: Without LCD message
-------------	--

Returns

success or error code. 1: Success, 0: Failed

11.7.4.100 device_setConfigPath()

```
int device_setConfigPath (
    const char * path )
```

Set the path to the config xml file(s) if any

Parameters

<i>path</i>	The path to the config xml files (such as "NEO2_Devices.xml" which contains the information of NEO2 devices). Only need to specify the path to the folder which contains the config files. File names are not needed. The maximum length of path is 200 characters including the '\0' at the end.
-------------	---

Returns

RETURN_CODE: Values can be parsed with [device_getIDGStatusCodeString\(\)](#)

11.7.4.101 device_setCoreDumpLogFile()

```
int device_setCoreDumpLogFile (
    IN char * filename,
    IN int filenameLen )
```

Set the Core Dump Log File Name

Set the file name of the log file when the core dump occurs for the firmware

Parameters

<i>filename</i>	the file name including the path
<i>filenameLen</i>	the length of filename; the maximum length is 99 bytes not including the null character

Returns

RETURN_CODE: Values can be parsed with [device_getIDGStatusCodeString\(\)](#)

11.7.4.102 device_setCurrentDevice()

```
int device_setCurrentDevice (
    int deviceType )
```

Sets the current device to talk to

The connect status can be checked by [device_isConnected\(\)](#).

Note: the file "NEO2_Devices.xml" is required for NEO 2 readers

Parameters

<i>deviceType</i>	Device to connect to <pre>enum DEVICE_TYPE { IDT_DEVICE_UNKNOWN=0, IDT_DEVICE_AUGUSTA_HID, IDT_DEVICE_AUGUSTA_KB, IDT_DEVICE_AUGUSTA_S_HID, IDT_DEVICE_AUGUSTA_S_KB, IDT_DEVICE_AUGUSTA_S_TTK_HID, IDT_DEVICE_SPECTRUM_PRO, IDT_DEVICE_MINISMART_II, IDT_DEVICE_L80, IDT_DEVICE_L100, IDT_DEVICE_UNIPAY, IDT_DEVICE_UNIPAY_I_V, IDT_DEVICE_VP3300_AJ, IDT_DEVICE_KIOSK_III, IDT_DEVICE_KIOSK_III_S, IDT_DEVICE_PIP_READER, IDT_DEVICE_VENDI, IDT_DEVICE_VP3300_USB, IDT_DEVICE_UNIPAY_I_V_TTK, IDT_DEVICE_VP3300_BT, IDT_DEVICE_VP8800, IDT_DEVICE_SREDKEY2_HID, IDT_DEVICE_SREDKEY2_KB, IDT_DEVICE_NEO2, IDT_DEVICE_MINISMART_II_COM = IDT_DEVICE_NEO2+5, IDT_DEVICE_SPECTRUM_PRO_COM, IDT_DEVICE_KIOSK_III_COM, IDT_DEVICE_KIOSK_III_S_COM, IDT_DEVICE_PIP_READER_COM, IDT_DEVICE_VP3300_COM, IDT_DEVICE_NEO2_COM, IDT_DEVICE_MAX_DEVICES = IDT_DEVICE_NEO2_COM+5 };</pre>
-------------------	---

Returns

RETURN_CODE: 1: success, 0: failed

11.7.4.103 device_setMerchantRecord()

```
int device_setMerchantRecord (
    int index,
    int enabled,
    char * merchantID,
    char * merchantURL )
```

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

Returns

RETURN_CODE: Values can be parsed with [device_getIDGStatusCodeString\(\)](#)

11.7.4.104 device_setNEO2DevicesConfigs()

```
int device_setNEO2DevicesConfigs (
    IN const char * configs,
    IN int len )
```

Pass the content of the config xml file ("NEO2_Devices.xml") as a string to the SDK instead of reading the config xml file by the SDK It needs to be called before [device_init\(\)](#), otherwise the SDK will try to read the config xml file.

Parameters

<i>configs</i>	The content read from the config xml file ("NEO2_Devices.xml" which contains the information of NEO2 devices).
<i>len</i>	The length of the string configs. The maximum length is 5000 bytes.

11.7.4.105 device_setNEOAltDevice()

```
void device_setNEOAltDevice (
    int alt )
```

Set the alternative device type for the NEO 2 or 3 readers

Parameters

<i>alt</i>	The alternative device type of the NEO 2/3 readers.
------------	---

11.7.4.106 device_setNEOGen()

```
void device_setNEOGen (
    int gen )
```

Set the generation for the NEO 2 or 3 readers

Parameters

<i>gen</i>	The generation of the NEO 2/3 readers. 2: NEO 2, 3: NEO 3
------------	---

11.7.4.107 device_setPollMode()

```
int device_setPollMode (
    IN BYTE mode )
```

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>	0 = Auto Poll, 1 = Poll On Demand
-------------	-----------------------------------

Returns

RETURN_CODE: Values can be parsed with [device_getIDGStatusCodeString\(\)](#)

11.7.4.108 device_setRKI_URL()

```
void device_setRKI_URL (
    IN char * rkiURL,
    IN int rkiURLLen )
```

Set the URL for RKI

Parameters

<i>rkiURL</i>	The URL for RKI (less than 100 characters). Pass NULL to reset to default URL
<i>rkiURLLen</i>	The length of rkiURL. Pass 0 to reset to default URL

Returns**11.7.4.109 device_setSDKWaitTime()**

```
void device_setSDKWaitTime (
    int waitTime )
```

Set SDK Wait Time

Set the SDK wait time for transactions

Parameters

<i>waitTime</i>	The SDK wait time for transaction in seconds. The Maximum is 2147483 seconds.
-----------------	---

11.7.4.110 device_setThreadStackSize()

```
void device_setThreadStackSize (
    int threadSize )
```

Set Thread Stack Size

Set the stack size setting for newly created threads

11.7.4.111 device_setTransactionExponent()

```
void device_setTransactionExponent (
    int exponent )
```

Sets the transaction exponent to be used with device_startTransaction. Default value is 2

Parameters

<i>exponent, The</i>	exponent to use when calling device_startTransaction
----------------------	--

11.7.4.112 device_startListenNotifications()

```
int device_startListenNotifications ( )
```

Start Listen Notifications This function enables Card Status and Front Switch notifications.

Returns

RETURN_CODE: Values can be parsed with [device_getIDGStatusCodeString\(\)](#)

11.7.4.113 device_startQRCodeScan()

```
int device_startQRCodeScan (
    IN int _timeout )
```

Start QR Code Scanning

Enables QR Code scanning with the default window, waiting for the QR code.

Parameters

<i>timeout</i>	QR Code Scan Timeout Value. Between 30 and 65536 seconds.
----------------	---

Returns

RETURN_CODE: Values can be parsed with [device_getIDGStatusCodeString\(\)](#) Note: if auto poll mode is on, it will return command not allowed error

11.7.4.114 device_startQRCodeScanWithDisplayWindowInfo()

```
int device_startQRCodeScanWithDisplayWindowInfo (
    IN int _timeout,
    IN int x,
    IN int y,
    IN int width,
    IN int height )
```

Start QR Code Scanning with Display Window Info

Enables QR Code scanning, waiting for the QR code.

Parameters

<i>timeout</i>	QR Code Scan Timeout Value. Between 30 and 65536 seconds.
<i>x</i>	the x-coordinate of the display window.
<i>y</i>	the y-coordinate of the display window.
<i>width</i>	the width of the display window.
<i>height</i>	the height of the display window.

Returns

RETURN_CODE: Values can be parsed with [device_getIDGStatusCodeString\(\)](#) Note: if auto poll mode is on, it will return command not allowed error

11.7.4.115 device_startRKI()

```
int device_startRKI (
```

```
IN const char * caPath,  
IN int isProduction )
```

Start remote key injection.

Parameters

<i>caPath</i>	The path to ca-certificates.crt It should be NULL, because the file is not used anymore.
<i>isProduction</i>	1: The reader is a production unit, 0: The reader is not a production unit

Returns

success or error code.

See also

[ErrorCode](#)

11.7.4.116 device_startTakingPhoto()

```
int device_startTakingPhoto (  
    IN int _timeout )
```

Start Taking Photo

Enables the camera to take a photo.

Parameters

<i>timeout</i>	Photo taking Timeout Value. Between 30 and 65536 seconds.
----------------	---

Returns

RETURN_CODE: Values can be parsed with [device_getIDGStatusCodeString\(\)](#) Note: if auto poll mode is on, it will return command not allowed error

11.7.4.117 device_startTransaction()

```
int device_startTransaction (  
    IN double amount,  
    IN double amtOther,  
    IN int type,  
    IN const int _timeout,  
    IN BYTE * tags,  
    IN int tagsLen )
```

Start Transaction Request

Authorizes the transaction for an MSR/CTLS/ICC card

The tags will be returned in the callback routine.

Parameters

<i>amount</i>	Transaction amount value (tag value 9F02) • SEE IMPORTANT NOTE BELOW
<i>amtOther</i>	Other amount value, if any (tag value 9F03) • SEE IMPORTANT NOTE BELOW
<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. 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>tagsLen</i>	The length of tags data buffer.

>>>>>IMPORTANT: parameters for amount and amtOther MUST BE PASSED AS A DOUBLE VALUE WITH DECIMAL POINT. Example, do not pass 1, but instead pass 1.0 or 1.00. Otherwise, results will be unpredictable

Returns

RETURN_CODE: Values can be parsed with [device_getIDGStatusCodeString\(\)](#) Note: if auto poll is on, it will return the error IDG_P2_STATUS_CODE_COMMAND_NOT_ALLOWED

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 device_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

11.7.4.118 `device_stopAudio()`

```
int device_stopAudio ( )
```

Stop Audio This command stops the audio the reader is playing.

Returns

RETURN_CODE: Values can be parsed with [device_getIDGStatusCodeString\(\)](#)

11.7.4.119 `device_stopListenNotifications()`

```
int device_stopListenNotifications ( )
```

Stop Listen Notifications This function disables Card Status and Front Switch notifications.

Returns

RETURN_CODE: Values can be parsed with [device_getIDGStatusCodeString\(\)](#)

11.7.4.120 `device_stopQRCodeScan()`

```
int device_stopQRCodeScan ( )
```

Stop QR Code Scanning Cancels QR Code scanning request.

Returns

RETURN_CODE: Values can be parsed with [device_getIDGStatusCodeString\(\)](#)

11.7.4.121 device_stopTakingPhoto()

```
int device_stopTakingPhoto ( )
```

Stop Taking Photo Cancels Photo Taking request.

Returns

RETURN_CODE: Values can be parsed with [device_getIDGStatusCodeString\(\)](#)

11.7.4.122 device_toSDCard()

```
void device_toSDCard (
    int forSDCard )
```

To SD Card

Set the destination of the file or directory function

Parameters

<i>forSDCard</i>	0: for internal memory, 1: for SD card
------------------	--

11.7.4.123 device_transferFile()

```
int device_transferFile (
    IN char * fileName,
    IN int fileNameLen,
    IN BYTE * file,
    IN int fileLen )
```

Update Firmware with zip file Updates the firmware of NEO 2 devices.

Parameters

<i>firmwareZipFilename</i>	Firmware zip file name. <ul style="list-style-type: none"> For example "package_VP6300 FW v1.01.003.0432.T.zip"
<i>firmwareZipFilenameLen</i>	Firmware zip file name length

Returns

RETURN_CODE: Values can be parsed with [device_getIDGStatusCodeString\(\)](#)

Firmware update status is returned in the callback with the following values: sender = device type state = DEVICE_↵
E_FIRMWARE_UPDATE current block total blocks ResultCode:

- RETURN_CODE_DO_SUCCESS = Firmware Update Completed Successfully
- Any other return code represents an error condition Note: to call this function under Windows, the executable file "unzip.exe" is required 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>filenameLen</i>	File Name Length.
<i>file</i>	The data file.
<i>fileLen</i>	File Length.

Returns

RETURN_CODE: Values can be parsed with [device_getIDGStatusCodeString\(\)](#)

11.7.4.124 device_turnOffYellowLED()

```
int device_turnOffYellowLED ( )
```

Use this function to turn off the ViVOpay Vendi reader yellow LED. This LED is located below the three blue LEDs

Returns

RETURN_CODE: Values can be parsed with [device_getIDGStatusCodeString\(\)](#)

11.7.4.125 device_turnOnYellowLED()

```
int device_turnOnYellowLED ( )
```

Use this function to turn on the ViVOpay Vendi reader yellow LED. This LED is located below the three blue LEDs

Returns

RETURN_CODE: Values can be parsed with [device_getIDGStatusCodeString\(\)](#)

11.7.4.126 device_updateFirmware()

```
int device_updateFirmware (
    IN BYTE * firmwareData,
    IN int firmwareDataLen,
    IN char * firmwareName,
    IN int encryptionType,
    IN BYTE * keyBlob,
    IN int keyBlobLen )
```

Update Firmware Updates the firmware of NEO 2 devices.

Parameters

<i>firmwareData</i>	Signed binary data of a firmware file provided by IDTech
<i>firmwareDataLen</i>	Length of firmwareData
<i>firmwareName</i>	Firmware name. <ul style="list-style-type: none"> For example "VP5300_v1.00.023.0167.S_Test.fm"

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)
<i>keyBlobLen</i>	Length of keyBlob

Returns

RETURN_CODE: Values can be parsed with [device_getIDGStatusCodeString\(\)](#)

Firmware update status is returned in the callback with the following values: sender = device type state = DEVICE_FIRMWARE_UPDATE current block total blocks ResultCode:

- RETURN_CODE_DO_SUCCESS = Firmware Update Started Successfully
- Any other return code represents an error condition

11.7.4.127 emv_activateTransaction()

```
int emv_activateTransaction (
    IN int timeout,
    IN BYTE * tags,
    IN int tagsLen,
    IN int forceOnline )
```

Activate 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>	Tags to be included in the request. Passed as a TLV stream. Example, tag 9F02 with amount 0x000000000100 would be 0x9F0206000000000100
<i>tagsLen</i>	Length of tags
<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 = DFEE1A079F029F36959F37

Returns

RETURN_CODE: Values can be parsed with [device_getIDGStatusCodeString\(\)](#)

11.7.4.128 emv_allowFallback()

```
void emv_allowFallback (
    IN int allow )
```

Allow fallback for EMV transactions. Default is TRUE

Parameters

<i>allow</i>	TRUE = allow fallback, FALSE = don't allow fallback
--------------	---

11.7.4.129 emv_authenticateTransaction()

```
int emv_authenticateTransaction (
    IN BYTE * updatedTLV,
    IN int updatedTLVLen )
```

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 = DFEE1A079F029F36959F37

Parameters

<i>updatedTLVLen</i>	
----------------------	--

Returns

RETURN_CODE: Values can be parsed with [device_getIDGStatusCodeString\(\)](#)

11.7.4.130 emv_authenticateTransactionWithTimeout()

```
int emv_authenticateTransactionWithTimeout (
    IN int timeout,
    IN BYTE * updatedTLV,
    IN int updatedTLVLen )
```

Authenticate EMV Transaction Request with Timeout

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>timeout</i>	Timeout value in seconds.
<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 = DFEE1A079F029F36959F37
<i>updatedTLVLen</i>	

Returns

RETURN_CODE: Values can be parsed with [device_getIDGStatusCodeString\(\)](#)

11.7.4.131 `emv_cancelTransaction()`

```
int emv_cancelTransaction ( )
```

Cancel EMV Transaction

Cancels the currently executing EMV transaction.

Returns

RETURN_CODE: Values can be parsed with [device_getIDGStatusCodeString\(\)](#)

11.7.4.132 `emv_completeTransaction()`

```
int emv_completeTransaction (
    IN int commError,
    IN BYTE * authCode,
    IN int authCodeLen,
    IN BYTE * iad,
    IN int iadLen,
    IN BYTE * tlvScripts,
    IN int tlvScriptsLen,
    IN BYTE * tlv,
    IN int tlvLen )
```

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(1) if host was unreachable, or FALSE(0) 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>authCodeLen</i>	the length of authCode
<i>iad</i>	Issuer Authentication Data, if any. Example 0x11223344556677883030 (tag value 91).
<i>iadLen</i>	the length of iadLen
<i>tlvScripts</i>	71/72 scripts, if any
<i>tlvScriptsLen</i>	the length of tlvScriptsLen
<i>tlv</i>	Additional TLV data to return with transaction results (if any)
<i>tlvLen</i>	the length of tlv

Returns

RETURN_CODE: Values can be parsed with [device_getIDGStatusCodeString\(\)](#)

11.7.4.133 `emv_getAutoAuthenticateTransaction()`

```
int emv_getAutoAuthenticateTransaction ( )
```

Gets auto authenticate value for EMV transactions.

Returns

RETURN_CODE: TRUE = auto authenticate, FALSE = manually authenticate

11.7.4.134 `emv_getAutoCompleteTransaction()`

```
int emv_getAutoCompleteTransaction ( )
```

Gets auto complete value for EMV transactions.

Returns

RETURN_CODE: TRUE = auto complete, FALSE = manually complete

11.7.4.135 `emv_getEMVConfigurationCheckValue()`

```
int emv_getEMVConfigurationCheckValue (
    OUT BYTE * checkValue,
    IN_OUT int * checkValueLen )
```

Get EMV Kernel configuration check value info

Parameters

<i>checkValue</i>	Response returned of Kernel configuration check value info
<i>checkValueLen</i>	the length of checkValue

Returns

RETURN_CODE: Values can be parsed with [device_getIDGStatusCodeString\(\)](#)

11.7.4.136 emv_getEMVKernelCheckValue()

```
int emv_getEMVKernelCheckValue (
    OUT BYTE * checkValue,
    IN_OUT int * checkValueLen )
```

Get EMV Kernel check value info

Parameters

<i>checkValue</i>	Response returned of Kernel check value info
<i>checkValueLen</i>	the length of checkValue

Returns

RETURN_CODE: Values can be parsed with [device_getIDGStatusCodeString](#)

11.7.4.137 emv_getEMVKernelVersion()

```
int emv_getEMVKernelVersion (
    OUT char * version )
```

DEPRECATED : please use [emv_getEMVKernelVersion_Len\(OUT char* version, IN_OUT int *versionLen\)](#)

Polls device for EMV Kernel Version

Parameters

<i>version</i>	Response returned of Kernel Version; needs to have at least 128 bytes of memory.
----------------	--

Returns

RETURN_CODE: Values can be parsed with [device_getIDGStatusCodeString\(\)](#)

11.7.4.138 emv_getEMVKernelVersion_Len()

```
int emv_getEMVKernelVersion_Len (
    OUT char * version,
    IN_OUT int * versionLen )
```

Polls device for EMV Kernel Version

Parameters

<i>version</i>	Response returned of Kernel Version
<i>versionLen</i>	Length of version

Returns

RETURN_CODE: Values can be parsed with [device_getIDGStatusCodeString\(\)](#)

11.7.4.139 emv_registerCallBk()

```
void emv_registerCallBk (
    pEMV_callback pEMVf )
```

To register the emv callback function to get the EMV processing response. (Pass NULL to disable the callback.)

11.7.4.140 emv_removeAllApplicationData()

```
int emv_removeAllApplicationData ( )
```

Remove All Application Data

Removes all the Application Data

Returns

RETURN_CODE: Values can be parsed with [device_getIDGStatusCodeString\(\)](#)

11.7.4.141 emv_removeAllCAPK()

```
int emv_removeAllCAPK ( )
```

Remove All Certificate Authority Public Key

Removes all the CAPK

Returns

RETURN_CODE: Values can be parsed with [device_getIDGStatusCodeString\(\)](#)

11.7.4.142 emv_removeAllCRL()

```
int emv_removeAllCRL ( )
```

Remove All Certificate Revocation List Entries

Removes all CRLentry entries

Returns

RETURN_CODE: Values can be parsed with [device_getIDGStatusCodeString\(\)](#)

11.7.4.143 emv_removeApplicationData()

```
int emv_removeApplicationData (
    IN BYTE * AID,
    IN int AIDLen )
```

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>AIDLen</i>	the length of AID data buffer

Returns

RETURN_CODE: Values can be parsed with [device_getIDGStatusCodeString\(\)](#)

11.7.4.144 emv_removeCAPK()

```
int emv_removeCAPK (
    IN BYTE * capk,
    IN int capkLen )
```

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>capkLen</i>	the length of capk data buffer

Returns

RETURN_CODE: Values can be parsed with [device_getIDGStatusCodeString\(\)](#)

11.7.4.145 emv_removeCRL()

```
int emv_removeCRL (
    IN BYTE * list,
    IN int lsLen )
```

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>lsLen</i>	the length of list data buffer

Returns

RETURN_CODE: Values can be parsed with [device_getIDGStatusCodeString\(\)](#)

11.7.4.146 emv_retrieveAIDList()

```
int emv_retrieveAIDList (
    OUT BYTE * AIDList,
    IN_OUT int * AIDListLen )
```

Retrieve AID list

Returns all the AID names installed on the terminal for CTLS. .

Parameters

<i>AIDList</i>	array of AID name byte arrays
<i>AIDListLen</i>	the length of AIDList array buffer

Returns

RETURN_CODE: Values can be parsed with [device_getIDGStatusCodeString\(\)](#)

11.7.4.147 emv_retrieveApplicationData()

```
int emv_retrieveApplicationData (
    IN BYTE * AID,
    IN int AIDLen,
    OUT BYTE * tlv,
    IN_OUT int * tlvLen )
```

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>AIDLen</i>	the length of AID data buffer.
<i>tlv</i>	The TLV elements of the requested AID
<i>tlvLen</i>	the length of tlv data buffer.

Returns

RETURN_CODE: Values can be parsed with [device_getIDGStatusCodeString\(\)](#)

11.7.4.148 emv_retrieveCAPK()

```
int emv_retrieveCAPK (
    IN BYTE * capk,
    IN int capkLen,
    OUT BYTE * key,
    IN_OUT int * keyLen )
```

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>capkLen</i>	the length of capk data buffer
<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>keyLen</i>	the length of key data buffer

Returns

RETURN_CODE: Values can be parsed with [device_getIDGStatusCodeString\(\)](#)

11.7.4.149 emv_retrieveCAPKList()

```
int emv_retrieveCAPKList (
    OUT BYTE * keys,
    IN_OUT int * keysLen )
```

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>keysLen</i>	the length of keys data buffer

Returns

RETURN_CODE: Values can be parsed with [device_getIDGStatusCodeString\(\)](#)

11.7.4.150 emv_retrieveCRL()

```
int emv_retrieveCRL (
    OUT BYTE * list,
    IN_OUT int * lssLen )
```

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>lssLen</i>	the length of list data buffer

Returns

RETURN_CODE: Values can be parsed with [device_getIDGStatusCodeString\(\)](#)

11.7.4.151 emv_retrieveTerminalData()

```
int emv_retrieveTerminalData (
    OUT BYTE * tlv,
    IN_OUT int * tlvLen )
```

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>tlvLen</i>	the length of tlv data buffer

Returns

RETURN_CODE: Values can be parsed with [device_getIDGStatusCodeString\(\)](#)

11.7.4.152 emv_retrieveTransactionResult()

```
int emv_retrieveTransactionResult (
    IN BYTE * tags,
    IN int tagsLen,
    OUT IDTTransactionData * cardData )
```

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>tagsLen</i>	Length of tag list
<i>cardData</i>	All requested tags returned as unencrypted, encrypted and masked TLV data in IDTTransactionData object

Returns

RETURN_CODE: Values can be parsed with [device_getIDGStatusCodeString\(\)](#)

11.7.4.153 emv_setApplicationData()

```
int emv_setApplicationData (
    IN BYTE * name,
    IN int nameLen,
    IN BYTE * tlv,
    IN int tlvLen )
```

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>nameLen</i>	the length of name data buffer of Application name,
<i>tlv</i>	Application data in TLV format
<i>tlvLen</i>	the length of tlv data buffer

Returns

RETURN_CODE: Values can be parsed with [device_getIDGStatusCodeString\(\)](#)

11.7.4.154 emv_setApplicationDataTLV()

```
int emv_setApplicationDataTLV (
    IN BYTE * tlv,
    IN int tlvLen )
```

Set Application Data by TLV

Sets the Application Data as specified by the TLV data

Parameters

<i>tlv</i>	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) Example valid TLV, for Group #2, AID a0000000035010: "dfee2d01029f0607a0000000051010ffe10101ffe50110ffe30114ffe20106"
<i>tlvLen</i>	the length of tlv data buffer

Returns

RETURN_CODE: Values can be parsed with [device_getIDGStatusCodeString\(\)](#)

11.7.4.155 emv_setAutoAuthenticateTransaction()

```
void emv_setAutoAuthenticateTransaction (
```

```
IN int authenticate )
```

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

Parameters

<i>authenticate</i>	TRUE = auto authenticate, FALSE = manually authenticate
---------------------	---

11.7.4.156 `emv_setAutoCompleteTransaction()`

```
void emv_setAutoCompleteTransaction (
    IN int complete )
```

Enables complete for EMV transactions. If a `emv_authenticateTransaction` results in code 0x0004 (go online), then `emv_completeTransaction` can automatically execute if parameter is set to TRUE

Parameters

<i>complete</i>	TRUE = auto complete, FALSE = manually complete
-----------------	---

11.7.4.157 `emv_setCAPK()`

```
int emv_setCAPK (
    IN BYTE * capk,
    IN int capkLen )
```

Set Certificate Authority Public Key

Sets the CAPK as specified by the CAKey structure

Parameters

<i>capk</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]</p> <p>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>capkLen</i>	the length of capk data buffer

Returns

RETURN_CODE: Values can be parsed with [device_getIDGStatusCodeString\(\)](#)

11.7.4.158 emv_setCRL()

```
int emv_setCRL (
    IN BYTE * list,
    IN int lsLen )
```

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>lsLen</i>	the length of list data buffer

Returns

RETURN_CODE: Values can be parsed with [device_getIDGStatusCodeString\(\)](#)

11.7.4.159 emv_setTerminalData()

```
int emv_setTerminalData (
    IN BYTE * tlv,
    IN int tlvLen )
```

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 file
<i>tlvLen</i>	the length of tlv data buffer

Return values

<i>RETURN_CODE</i>	Values can be parsed with device_getIDGStatusCodeString()
--------------------	---

11.7.4.160 emv_setTerminalMajorConfiguration()

```
int emv_setTerminalMajorConfiguration (
```

```
IN int configuration )
```

Sets the terminal major configuration in ICS .

Parameters

<i>configuration</i>	A configuration value, range 1-23 <ul style="list-style-type: none"> • 1 = 1C • 2 = 2C • 3 = 3C • 4 = 4C • 5 = 5C ... • 23 = 23C
----------------------	--

Returns

RETURN_CODE: Values can be parsed with [device_getIDGStatusCodeString\(\)](#)

11.7.4.161 emv_setTransactionParameters()

```
void emv_setTransactionParameters (
    IN double amount,
    IN double amtOther,
    IN int type,
    IN int timeout,
    IN BYTE * tags,
    IN int tagsLen )
```

Set EMV Transaction Parameters

Set the parameters to be used on EMV transactions for an ICC card when Auto Poll is on

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 (Maximum Length = 500 bytes). Passed as a string. Example, tag 9F02 with amount 0x000000000100 would be "9F0206000000000100" 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 = DFEE1A079F029F36959F37
<i>tagsLen</i>	the length of tags

11.7.4.162 emv_startTransaction()

```
int emv_startTransaction (
    IN double amount,
    IN double amtOther,
    IN int exponent,
    IN int type,
    IN int timeout,
    IN BYTE * tags,
    IN int tagsLen,
    IN int forceOnline )
```

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) - SEE IMPORTANT NOTE BELOW
<i>amtOther</i>	Other amount value, if any (tag value 9F03) - SEE IMPORTANT NOTE BELOW
<i>exponent</i>	Number of characters after decimal 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 stream. 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>tagsLen</i>	Length of tags
<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 = DFEE1A079F029F36959F37

Returns

RETURN_CODE: Values can be parsed with device_getIDGStatusCodeString >>>>>IMPORTANT: parameters for amount and amtOther MUST BE PASSED AS A DOUBLE VALUE WITH DECIMAL POINT. Example, do not pass 1, but instead pass 1.0 or 1.00. Otherwise, results will be unpredictable

11.7.4.163 executeTransaction()

```
int executeTransaction (
    WorldPayData * data,
    pWP_callback wpCallback,
    int requestOnly )
```

executeTransaction Requests 3 sets of public keys: encrypting Keys, signing/validating keys, signing/validating 3rd party apps

Parameters

<i>data</i>	WorldPay data object
<i>wpCallback</i>	WorldPay callback

Parameters

<i>requestOnly</i>	<ul style="list-style-type: none"> • 0 = send transaction and return response, • 1 = send transaction and return both request and response, • 2 = do not send transaction, instead return request
--------------------	--

Returns

RETURN_CODE: Values can be parsed with [device_getIDGStatusCodeString\(\)](#)

11.7.4.164 executeTransaction_WorldNet()

```
int executeTransaction_WorldNet (
    WorldNetData * data,
    pWN_callback wnCallback,
    int requestOnly )
```

executeTransaction_WorldNet Requests 3 sets of public keys: encrypting Keys, signing/validating keys, signing/validating 3rd party apps

Parameters

<i>data</i>	WorldNet data object
<i>wnCallback</i>	WorldNet callback
<i>requestOnly</i>	<ul style="list-style-type: none"> • 0 = send transaction and return response, • 1 = send transaction and return both request and response, • 2 = do not send transaction, instead return request

Returns

RETURN_CODE: Values can be parsed with [device_getIDGStatusCodeString\(\)](#)

11.7.4.165 felica_authentication()

```
int felica_authentication (
    IN BYTE * key,
    IN int keyLen )
```

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

Parameters

<i>key</i>	16-byte key used for MAC generation of Read or Write with MAC
<i>keyLen</i>	length of key, must be 16 bytes

Returns

RETURN_CODE: Values can be parsed with [device_getIDGStatusCodeString\(\)](#)

11.7.4.166 felica_cancelCodeEntry()

```
int felica_cancelCodeEntry ( )
```

FeliCa Cancel Code Entry

Cancels FeliCa Get Code request

Returns

RETURN_CODE: Values can be parsed with [device_getIDGStatusCodeString\(\)](#)

11.7.4.167 felica_getCode()

```
int felica_getCode ( )
```

FeliCa Get Code

Start the FeliCa get code process Since the firmware timeout is 180 seconds, the SDK timeout of the command is set to 181 seconds.

Returns

RETURN_CODE: Values can be parsed with [device_getIDGStatusCodeString\(\)](#)

11.7.4.168 felica_poll()

```
int felica_poll (
    IN BYTE * systemCode,
    IN int systemCodeLen,
    OUT BYTE * respData,
    OUT int * respDataLen )
```

FeliCa Poll for Card

Polls for a Felica Card

Parameters

<i>systemCode</i>	System Code.
<i>systemCodeLen</i>	Length of systemCode. Must be 2 bytes
<i>respData</i>	response data will be stored in respData. Poll response as explained in FeliCA Lite-S User's Manual
<i>respDataLen</i>	Length of systemCode.

Returns

RETURN_CODE: Values can be parsed with [device_getIDGStatusCodeString\(\)](#)

11.7.4.169 felica_read()

```
int felica_read (
    IN BYTE * serviceCodeList,
    IN int serviceCodeListLen,
    IN int blockCnt,
    IN BYTE * blockList,
    IN int blockListLen,
    OUT BYTE * blockData,
    OUT int * blockDataLen )
```

FeliCa Read

Reads up to 4 blocks.

Parameters

<i>serviceCodeList</i>	Service Code List. Each service code in Service Code List = 2 bytes of data
<i>serviceCodeListLen</i>	Length of serviceCodeList
<i>blockCnt</i>	Number of blocks in blockList. Maximum 4 block requests
<i>blockList</i>	Block to read. Each block in blockList = 2 or 3 bytes of data.
<i>blockListLen</i>	Length of blockList.
<i>blockData</i>	Blocks read will be stored in blockData. Each block 16 bytes.
<i>blockDataLen</i>	Length of blockData.

Returns

RETURN_CODE: Values can be parsed with [device_getIDGStatusCodeString\(\)](#)

11.7.4.170 felica_readWithMac()

```
int felica_readWithMac (
    IN int blockCnt,
    IN BYTE * blockList,
    IN int blockListLen,
    OUT BYTE * blockData,
    OUT int * blockDataLen )
```

FeliCa Read with MAC Generation

Reads up to 3 blocks with MAC Generation. FeliCa Authentication must be performed first

Parameters

<i>blockCnt</i>	Number of blocks in blockList. Maximum 3 block requests
<i>blockList</i>	Block to read. Each block in blockList = 2 or 3 bytes of data.
<i>blockListLen</i>	Length of blockList.
<i>blockData</i>	Blocks read will be stored in blockData. Each block is 16 bytes.
<i>blockDataLen</i>	Length of blockData.

Returns

RETURN_CODE: Values can be parsed with [device_getIDGStatusCodeString\(\)](#)

11.7.4.171 felica_requestService()

```
int felica_requestService (
    IN BYTE * nodeCode,
    IN int nodeCodeLen,
    OUT BYTE * respData,
    OUT int * respDataLen )
```

FeliCa Request Service

Request Service for a Felica Card

Parameters

<i>nodeCode</i>	Node Code List. Each node 2 bytes
<i>nodeCodeLen</i>	Length of nodeCode.
<i>respData</i>	response data will be stored in respData. Response as explained in FeliCA Lite-S User's Manual
<i>respDataLen</i>	Length of respData.

Returns

RETURN_CODE: Values can be parsed with [device_getIDGStatusCodeString\(\)](#)

11.7.4.172 felica_SendCommand()

```
int felica_SendCommand (
    IN BYTE * command,
    IN int commandLen,
    OUT BYTE * respData,
    OUT int * respDataLen )
```

FeliCa Send Command

Send a Felica Command

Parameters

<i>command</i>	Command data from settlement center to be sent to felica card
<i>commandLen</i>	Length of command data
<i>respData</i>	Response data from felica card.
<i>respDataLen</i>	Length of respData.

Returns

RETURN_CODE: Values can be parsed with [device_getIDGStatusCodeString\(\)](#)

11.7.4.173 felica_write()

```
int felica_write (
    IN BYTE * serviceCodeList,
    IN int serviceCodeListLen,
    IN int blockCnt,
    IN BYTE * blockList,
    IN int blockListLen,
    IN BYTE * blockData,
    IN int blockDataLen,
    OUT BYTE * statusFlag,
    OUT int * statusFlagLen )
```

FeliCa Write

Writes a block

Parameters

<i>serviceCodeList</i>	Service Code List. Each service code in Service Code List = 2 bytes of data
<i>serviceCodeListLen</i>	Length of serviceCodeList
<i>blockCnt</i>	Number of blocks in blockList. Currently only support 1 block.
<i>blockList</i>	Block list. Each block in blockList = 2 or 3 bytes of data.
<i>blockListLen</i>	Length of blockData.
<i>blockData</i>	Block to write.
<i>blockDataLen</i>	Length of blockData. Must be 16 bytes.
<i>respData</i>	If successful, the Status Flag (2 bytes) is stored in respData.resData. Status flag response as explained in FeliCA Lite-S User's Manual, Section 4.5

Returns

RETURN_CODE: Values can be parsed with [device_getIDGStatusCodeString\(\)](#)

11.7.4.174 felica_writeWithMac()

```
int felica_writeWithMac (
    IN BYTE blockNum,
    IN BYTE * blockData,
    IN int blockDataLen )
```

FeliCa Write with MAC Generation

Writes a block with MAC Generation. FeliCa Authentication must be performed first

Parameters

<i>blockNum</i>	Number of block
<i>blockData</i>	Block to write.
<i>blockDataLen</i>	Length of blockData. Must be 16 bytes.

Returns

RETURN_CODE: Values can be parsed with [device_getIDGStatusCodeString\(\)](#)

11.7.4.175 forwardTransaction()

```
int forwardTransaction (
    IN pWP_callback wpCallback,
    IN char * forwardID,
    IN int forwardIDLen,
    IN char * password,
    IN int passwordLen,
    IN int bypassProcessing )
```

forwardTransaction Send the saved data to WorldPay and complete the transaction.

Parameters

<i>wpCallback</i>	WPCallback is the callback to send the results. Should be the same as executeTransaction callback.
<i>forwardID</i>	= ID, which could be either unique ID or Memo.
<i>forwardIDLen</i>	The length of forwardID.
<i>password</i>	= password. If null/blank, no password.
<i>passwordLen</i>	The length of passwordLen.
<i>bypassProcess</i>	= true means read the file from disk, but don't send to WorldPay, then delete the transaction as if you did send to WorldPay. The purpose of this is to allow them to delete transactions from the storage without sending to WorldPay.

Returns

RETURN_CODE: Values can be parsed with [device_getIDGStatusCodeString\(\)](#)

11.7.4.176 forwardTransaction_WorldNet()

```
int forwardTransaction_WorldNet (
    IN char * apiKey,
    IN int apiKeyLen,
    IN pWN_callback wnCallback,
    IN char * forwardID,
    IN int forwardIDLen,
    IN char * password,
    IN int passwordLen,
    IN int bypassProcessing )
```

forwardTransaction_WorldNet Send the saved data to WorldNet and complete the transaction.

Parameters

<i>wnCallback</i>	WNCallback is the callback to send the results. Should be the same as executeTransaction_WorldNet callback.
<i>forwardID</i>	= ID, which could be either unique ID or Memo.
<i>forwardIDLen</i>	The length of forwardID.
<i>password</i>	= password. If null/blank, no password.
<i>passwordLen</i>	The length of passwordLen.
<i>bypassProcess</i>	= true means read the file from disk, but don't send to WorldPay, then delete the transaction as if you did send to WorldPay. The purpose of this is to allow them to delete transactions from the storage without sending to WorldPay.

Returns

RETURN_CODE: Values can be parsed with [device_getIDGStatusCodeString\(\)](#)

11.7.4.177 `icc_exchangeAPDU()`

```
int icc_exchangeAPDU (
    IN BYTE * c_APDU,
    IN int cLen,
    OUT BYTE * reData,
    IN_OUT int * reLen )
```

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>cLen</i>	APDU data packet length
<i>reData</i>	Unencrypted APDU response
<i>reLen</i>	Unencrypted APDU response data length

Returns

RETURN_CODE: Values can be parsed with [device_getIDGStatusCodeString](#)

11.7.4.178 `icc_getICCRReaderStatus()`

```
int icc_getICCRReaderStatus (
    OUT BYTE * status )
```

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
---------------	--

Returns

RETURN_CODE: Values can be parsed with [device_getIDGStatusCodeString](#)

11.7.4.179 `icc_powerOffICC()`

```
int icc_powerOffICC ( )
```

Power Off ICC

Powers down the ICC

Returns

RETURN_CODE: Values can be parsed with [device_getIDGStatusCodeString\(\)](#)

If Success, empty If Failure, ASCII encoded data of error string

11.7.4.180 icc_powerOnICC()

```
int icc_powerOnICC (
    OUT BYTE * ATR,
    IN_OUT int * inLen )
```

Power On ICC

Power up the currently selected microprocessor card in the ICC reader

Parameters

<i>ATR,the</i>	ATR data response when succeeded power on ICC,
<i>inLen,the</i>	length of ATR data

Returns

RETURN_CODE: Values can be parsed with [device_getIDGStatusCodeString\(\)](#)

11.7.4.181 lcd_addButton()

```
int lcd_addButton (
    IN char * screenName,
    IN int screenNameLen,
    IN char * buttonName,
    IN int buttonNameLen,
    IN BYTE type,
    IN BYTE alignment,
    IN int xCord,
    IN int yCord,
    IN char * label,
    IN int labelLen,
    OUT IDTLCDItem * returnItem )
```

Add Button

Adds a button to a selected screen. Must execute `lcd_createScreen` first to establish a screen to draw on. The SDK timeout of the command is set to 6 seconds.

Parameters

<i>screenName</i>	Screen name that will be the target of add button
<i>screenNameLen</i>	Length of screenName
<i>buttonName</i>	Button name that will be the target of add button
<i>buttonNameLen</i>	Length of buttonName

Parameters

<i>type</i>	Button Type <ul style="list-style-type: none"> • Large = 0x01 • Medium = 0x02 • Invisible = 0x03 (70px by 60 px)
<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 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 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>labelLen</i>	Length of label
<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

Returns

RETURN_CODE: Values can be parsed with [device_getIDGStatusCodeString\(\)](#)

11.7.4.182 lcd_addEthernet()

```
int lcd_addEthernet (
    IN char * screenName,
    IN int  screenNameLen,
    IN char * objectName,
    IN int  objectNameLen,
    IN BYTE alignment,
    IN int  xCord,
    IN int  yCord,
    OUT IDTLCDItem * returnItem )
```

Add Ethernet Settings

Adds an Ethernet settings to a selected screen. Must execute lcd_createScreen first to establish a screen to draw on. The SDK timeout of the command is set to 6 seconds.

Parameters

<i>screenName</i>	Screen name that will be the target of add widget
<i>screenNameLen</i>	Length of screenName
<i>objectName</i>	Object name that will be the target of add widget
<i>objectNameLen</i>	Length of objectName

Parameters

<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

Returns

RETURN_CODE: Values can be parsed with [device_getIDGStatusCodeString\(\)](#)

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

11.7.4.183 lcd_addExtVideo()

```
int lcd_addExtVideo (
    IN char * screenName,
    IN int  screenNameLen,
    IN char * objectName,
    IN int  objectNameLen,
    IN BYTE alignment,
    IN int  xCord,
    IN int  yCord,
    IN BYTE loop,
    IN BYTE numVideos,
    IN char * filenames,
    IN int  filenamesLen,
    OUT IDTLCDItem * returnItem )
```

Add Extended Video

Adds a extended video to a selected screen. Must execute `lcd_createScreen` first to establish a screen to draw on. The SDK timeout of the command is set to 6 seconds.

Parameters

<i>screenName</i>	Screen name that will be the target of add video
<i>screenNameLen</i>	Length of screenName
<i>objectName</i>	Object name that will be the target of add video
<i>objectNameLen</i>	Length of objectName
<i>alignment</i>	Position for Video <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 Video, range 0-271
<i>yCord</i>	y-coordinate for Video, range 0-479
<i>loop</i>	loop the videos when it's done. 0: Do not Loop 1: Loop
<i>numVideos</i>	number of video files, range 1-4
<i>filenames</i>	Filenames of the videos separated by '\0'. Must be available in the sd card.
<i>filenamesLen</i>	Length of filenames excluding the last NULL character. Should be less than 64.
<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 video

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
video	1

11.7.4.184 lcd_addImage()

```
int lcd_addImage (
    IN char * screenName,
    IN int screenNameLen,
    IN char * objectName,
    IN int objectNameLen,
    IN BYTE alignment,
    IN int xCord,
    IN int yCord,
```

```

IN char * filename,
IN int  filenameLen,
OUT IDTLCDItem * returnItem )

```

Add Image

Adds a image to a selected screen. Must execute `lcd_createScreen` first to establish a screen to draw on. The SDK timeout of the command is set to 6 seconds.

Parameters

<i>screenName</i>	Screen name that will be the target of add image
<i>screenNameLen</i>	Length of screenName
<i>objectName</i>	Object name that will be the target of add image
<i>objectNameLen</i>	Length of objectName
<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 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 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>filenameLen</i>	Length of filename
<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

Returns

RETURN_CODE: Values can be parsed with [device_getIDGStatusCodeString\(\)](#)

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

11.7.4.185 lcd_addLED()

```

int lcd_addLED (
    IN char * screenName,
    IN int  screenNameLen,

```

```

IN char * objectName,
IN int  objectNameLen,
IN BYTE alignment,
IN int  xCord,
IN int  yCord,
OUT IDTLCDItem * returnItem,
IN BYTE * LED,
IN int  LEDLen )

```

Add LED

Adds a LED widget to a selected screen. Must execute `lcd_createScreen` first to establish a screen to draw on. The SDK timeout of the command is set to 6 seconds.

Parameters

<i>screenName</i>	Screen name that will be the target of add LED
<i>screenNameLen</i>	Length of screenName
<i>objectName</i>	Object name that will be the target of add LED
<i>objectNameLen</i>	Length of objectName
<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 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 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>LEDLen</i>	Length of LED

Returns

RETURN_CODE: Values can be parsed with [device_getIDGStatusCodeString\(\)](#)

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

Item	Maximum can be created for each screen
Led widget	1
image	20

11.7.4.186 lcd_addText()

```
int lcd_addText (
    IN char * screenName,
    IN int  screenNameLen,
    IN char * objectName,
    IN int  objectNameLen,
    IN BYTE alignment,
    IN int  xCord,
    IN int  yCord,
    IN int  width,
    IN int  height,
    IN BYTE fontID,
    IN BYTE * color,
    IN int  colorLen,
    IN char * label,
    IN int  labelLen,
    OUT IDTLCDItem * returnItem )
```

Add text

Adds a text component to a selected screen. Must execute lcd_createScreen first to establish a screen to draw on. The SDK timeout of the command is set to 6 seconds.

Parameters

<i>screenName</i>	Screen name that will be the target of add text
<i>screenNameLen</i>	Length of screenName
<i>objectName</i>	Object name that will be the target of add text
<i>objectNameLen</i>	Length of objectName
<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

Parameters

<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>colorLen</i>	Length of color
<i>label</i>	Label to show on the text
<i>labelLen</i>	Length of label
<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

Returns

RETURN_CODE: Values can be parsed with [device_getIDGStatusCodeString\(\)](#)

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
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

Font ID	Typography Name	Font	Size
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

11.7.4.187 lcd_addVideo()

```
int lcd_addVideo (
    IN char * screenName,
    IN int screenNameLen,
    IN char * objectName,
    IN int objectNameLen,
    IN BYTE alignment,
    IN int xCord,
    IN int yCord,
    IN char * filename,
    IN int filenameLen,
    OUT IDTLCDItem * returnItem )
```

Add Video

Adds a video to a selected screen. Must execute lcd_createScreen first to establish a screen to draw on. The SDK timeout of the command is set to 6 seconds.

Parameters

<i>screenName</i>	Screen name that will be the target of add video
<i>screenNameLen</i>	Length of screenName

Parameters

<i>objectName</i>	Object name that will be the target of add video
<i>objectNameLen</i>	Length of objectName
<i>alignment</i>	Position for Video <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 Video, range 0-271
<i>yCord</i>	y-coordinate for Video, range 0-479
<i>filename</i>	Filename of the video. Must be available in the sd card.
<i>filenameLen</i>	Length of filename
<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 video

Returns

RETURN_CODE: Values can be parsed with [device_getIDGStatusCodeString\(\)](#)

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
video	1

11.7.4.188 lcd_clearDisplay()

```
int lcd_clearDisplay (
    IN BYTE control )
```

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>control</i>	for L80 and L100 only. 0:First Line 1:Second Line 2:Third Line 3:Fourth Line 0xFF: All Screen
----------------	---

Returns

RETURN_CODE: Values can be parsed with [device_getIDGStatusCodeString\(\)](#)

11.7.4.189 lcd_clearScreenInfo()

```
int lcd_clearScreenInfo ( )
```

Clear Screen Info

Clear all current screen information in RAM and flash. And then show 'power-on screen'

Returns

RETURN_CODE: Values can be parsed with [device_getIDGStatusCodeString\(\)](#)

11.7.4.190 lcd_cloneScreen()

```
int lcd_cloneScreen (
    IN char * screenName,
    IN int  screenNameLen,
    IN char * cloneName,
    IN int  cloneNameLen,
    OUT int * cloneID )
```

Clone Screen

Clones an existing screen.

Parameters

<i>screenName</i>	Screen name to clone.
<i>screenNameLen</i>	Length of screenName.
<i>cloneName</i>	Name of the cloned screen.
<i>cloneNameLen</i>	Length of cloneName.
<i>cloneID</i>	Screen ID of the cloned screen

Returns

RETURN_CODE: Values can be parsed with [device_getIDGStatusCodeString](#)

11.7.4.191 lcd_createScreen()

```
int lcd_createScreen (
    IN char * screenName,
    IN int  screenNameLen,
    OUT int * ScreenID )
```

Create Screen

Creates a new screen.

Parameters

<i>screenName</i>	Screen name to use.
<i>screenNameLen</i>	Length of screenName.
<i>screenID</i>	Screen ID that was created.

Returns

RETURN_CODE: Values can be parsed with [device_getIDGStatusCodeString\(\)](#)

11.7.4.192 lcd_destroyScreen()

```
int lcd_destroyScreen (
    IN char * screenName,
    IN int screenNameLen )
```

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>screenNameLen</i>	Length of screenName.

Returns

RETURN_CODE: Values can be parsed with [device_getIDGStatusCodeString\(\)](#)

11.7.4.193 lcd_displayMessage()

```
int lcd_displayMessage (
    int lineNumber,
    char * message,
    int messageLen )
```

Display Message on Line

Displays a message on a display line.

@param lineNumber Line number to display message on (1-4)

@param message Message to display

@param messageLen length of message

Returns

RETURN_CODE: Values can be parsed with [device_getResponseCodeString](#)

11.7.4.194 lcd_getActiveScreen()

```
int lcd_getActiveScreen (
    OUT char * screenName,
    IN_OUT int * screenNameLen )
```

Get Active Screen

Returns the active screen ID.

Parameters

<i>screenName</i>	Screen name this is active.
<i>screenNameLen</i>	Length of screenName.

Returns

RETURN_CODE: Values can be parsed with [device_getIDGStatusCodeString\(\)](#)

11.7.4.195 lcd_getAllObjects()

```
int lcd_getAllObjects (
    IN char * screenName,
    IN int screenNameLen,
    IN_OUT int * objectNumbers,
    OUT IDTObjectInfo * objectInfo )
```

Get All Objects on Screen

Get all created objects' name on certain screen

Parameters

<i>screenName</i>	Screen name to get all objects
<i>screenNameLen</i>	Length of screenName
<i>objectNumbers</i>	Number of created objects
<i>returnObjects</i>	Array of all created objects each element includes -objectID of a created object -objectName of a created object -objectNameLen of objectName

Returns

RETURN_CODE: Values can be parsed with [device_getIDGStatusCodeString\(\)](#)

11.7.4.196 lcd_getAllScreens()

```
int lcd_getAllScreens (
    IN_OUT int * screenNumbers,
    OUT IDTScreenInfo * screenInfo )
```

Get All Screens

Get all created screens' name

Parameters

<i>screenNumbers</i>	Number of created screens
<i>screenInfo</i>	Array of all created screens each element includes -screenID of a created screen -screenName of a created screen -screenNameLen of screenName

Returns

RETURN_CODE: Values can be parsed with [device_getIDGStatusCodeString\(\)](#)

11.7.4.197 lcd_getButtonEvent()

```
int lcd_getButtonEvent (
    OUT int * screenID,
    OUT int * objectID,
    OUT char * screenName,
    IN_OUT int * screenNameLen,
    OUT char * objectName,
    IN_OUT int * objectNameLen,
    OUT int * isLongPress )
```

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>screenNameLen</i>	Length of screenName
<i>objectName</i>	Button Name of the last clicked button
<i>objectNameLen</i>	Length of objectName
<i>isLongPress</i>	1 = Long Press, 0 = Short Press

Returns

RETURN_CODE: Values can be parsed with [device_getIDGStatusCodeString\(\)](#)

11.7.4.198 lcd_linkUIWithTransactionMessageId()

```
int lcd_linkUIWithTransactionMessageId (
    IN BYTE MessageId,
    IN char * screenName,
    IN int screenNameLen )
```

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 32 bytes including the NULL character)
<i>screenNameLen</i>	Length of screenName

Returns

RETURN_CODE: Values can be parsed with [device_getIDGStatusCodeString\(\)](#)

11.7.4.199 lcd_loadScreenInfo()

```
int lcd_loadScreenInfo ( )
```

Load Screen Info

Load all current screen information from flash to RAM

Returns

RETURN_CODE: Values can be parsed with [device_getIDGStatusCodeString\(\)](#)

11.7.4.200 lcd_queryObjectbyID()

```
int lcd_queryObjectbyID (
    IN int objectID,
    OUT int * objectNumbers,
    OUT IDTScreenInfo * screenInfo )
```

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>objectNumbers</i>	Number of the checked object
<i>screenInfo</i>	screen names containing the item

Returns

RETURN_CODE: Values can be parsed with [device_getIDGStatusCodeString\(\)](#)

11.7.4.201 lcd_queryObjectbyName()

```
int lcd_queryObjectbyName (
    IN char * objectName,
    IN int objectNameLen,
    IN_OUT int * objectNumbers,
    OUT IDTScreenInfo * screenInfo )
```

Queery 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>objectNameLen</i>	Length of objectName
<i>objectNumbers</i>	Number of the checked object
<i>screenInfo</i>	Array of all the screen names that contain the object

Returns

RETURN_CODE: Values can be parsed with [device_getIDGStatusCodeString\(\)](#)

11.7.4.202 lcd_queryScreenbyID()

```
int lcd_queryScreenbyID (
    IN int screenID,
    OUT int * result,
    OUT int * screenName,
    IN_OUT int * screenNameLen )
```

Queery 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>screenNameLen</i>	Length of screenName
<i>ip</i>	Optional: IP address to execute command on (for IP connected devices)

Returns

RETURN_CODE: Values can be parsed with [device_getIDGStatusCodeString\(\)](#)

11.7.4.203 lcd_queryScreenbyName()

```
int lcd_queryScreenbyName (
    IN char * screenName,
    IN int screenNameLen,
    OUT int * result )
```

Queery Screen by Name

Check if the given screen exists or not

Parameters

<i>screenName</i>	Name of the checked screen
<i>screenNameLen</i>	Length of screenName
<i>result</i>	the checking result

Returns

RETURN_CODE: Values can be parsed with [device_getIDGStatusCodeString\(\)](#)

11.7.4.204 lcd_registerCallBk()

```
void lcd_registerCallBk (
    pLCD_callBack pLCDf )
```

To register the lcd callback function to get the LCDItem. (Pass NULL to disable the callback.)

11.7.4.205 lcd_removeItem()

```
int lcd_removeItem (
    IN char * screenName,
    IN int  screenNameLen,
    IN char * objectName,
    IN int  objectNameLen )
```

Removed Item

Removes a component.

Parameters

<i>screenName</i>	Screen name where to remove the target from.
<i>screenNameLen</i>	Length of screenName.
<i>objectName</i>	Identifier of the component
<i>objectNameLen</i>	Length of objectName.

Returns

RETURN_CODE: Values can be parsed with [device_getIDGStatusCodeString\(\)](#)

11.7.4.206 lcd_setBacklight()

```
int lcd_setBacklight (
    IN BYTE isBacklightOn,
    IN BYTE backlightVal )
```

Set Backlight

Set backlight percentage. If the percent > 100, it will be rejected. If 0 < percent < 10, backlight percent will be set to 10.

Parameters

<i>isBacklightOn</i>	0: backlight off 1: backlight on
<i>backlightVal</i>	Backlight percent value to be sat

Returns

RETURN_CODE: Values can be parsed with [device_getIDGStatusCodeString\(\)](#)

11.7.4.207 lcd_showScreen()

```
int lcd_showScreen (
    IN char * screenName,
    IN int  screenNameLen )
```

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>screenNameLen</i>	Length of <code>screenName</code> .

Returns

RETURN_CODE: Values can be parsed with [device_getIDGStatusCodeString\(\)](#)

11.7.4.208 lcd_storeScreenInfo()

```
int lcd_storeScreenInfo ( )
```

Store Screen Info

Store all current screen information from RAM to flash

Returns

RETURN_CODE: Values can be parsed with [device_getIDGStatusCodeString\(\)](#)

11.7.4.209 lcd_updateColor()

```
int lcd_updateColor (
    IN char * screenName,
    IN int  screenNameLen,
    IN char * objectName,
    IN int  objectNameLen,
    IN BYTE * color,
    IN int  colorLen )
```

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>screenNameLen</i>	Length of <code>screenName</code> .

Parameters

<i>objectName</i>	Identifier of the component
<i>objectNameLen</i>	Length of objectName.
<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>colorLen</i>	Length of color.

Returns

RETURN_CODE: Values can be parsed with [device_getIDGStatusCodeString\(\)](#)

11.7.4.210 lcd_updateLabel()

```
int lcd_updateLabel (
    IN char * screenName,
    IN int screenNameLen,
    IN char * objectName,
    IN int objectNameLen,
    IN char * label,
    IN int labelLen )
```

Update Label

Updates the component label.

Parameters

<i>screenName</i>	Screen name that will be the target of update label
<i>screenNameLen</i>	Length of screenName.
<i>objectName</i>	Identifier of the component
<i>objectNameLen</i>	Length of objectName.
<i>label</i>	Label to show on the component
<i>labelLen</i>	Length of label.

Returns

RETURN_CODE: Values can be parsed with [device_getIDGStatusCodeString\(\)](#)

11.7.4.211 lcd_updatePosition()

```
int lcd_updatePosition (
    IN char * screenName,
    IN int  screenNameLen,
    IN char * objectName,
    IN int  objectNameLen,
    IN BYTE alignment,
    IN int  new_xCord,
    IN int  new_yCord )
```

Update Position

Updates the component position.

Parameters

<i>screenName</i>	Screen Name that will be the target of update position
<i>screenNameLen</i>	Length of screenName.
<i>objectName</i>	Identifier of the component
<i>objectNameLen</i>	Length of objectName.
<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 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>new_xCord</i>	x-coordinate for Text, range 0-271
<i>new_yCord</i>	y-coordinate for Text, range 0-479

Returns

RETURN_CODE: Values can be parsed with [device_getIDGStatusCodeString\(\)](#)

11.7.4.212 loyalty_cancelTransaction()

```
int loyalty_cancelTransaction ( )
```

Cancel Loyalty Transaction Only for VP6800

Cancels the currently executing transaction.

Returns

RETURN_CODE: Values can be parsed with [device_getIDGStatusCodeString\(\)](#)

11.7.4.213 loyalty_cancelTransactionSilent()

```
int loyalty_cancelTransactionSilent (
    int enable )
```

Cancel Loyalty Transaction Silent Only for VP6800

Cancel transaction with or without showing the LCD message

Parameters

<i>enable</i>	0: With LCD message 1: Without LCD message
---------------	--

Returns

success or error code. Values can be parsed with device_getIDGStatusCodeString

11.7.4.214 loyalty_registerCallBk()

```
void loyalty_registerCallBk (
    pEMV_callback pEMVf )
```

To register the loyalty callback function to get the EMV processing response. (Pass NULL to disable the callback.)
Only for VP6800

11.7.4.215 loyalty_startTransaction()

```
int loyalty_startTransaction (
    IN double amount,
    IN double amtOther,
    IN int type,
    IN const int _timeout,
    IN BYTE * tags,
    IN int tagsLen,
    IN const int cardType,
    IN const int iccReadType )
```

Start Loyalty Transaction Request Only for VP6800

Authorizes the transaction for an MSR/ICC card

The tags will be returned in the callback routine.

Parameters

<i>amount</i>	Transaction amount value (tag value 9F02) • SEE IMPORTANT NOTE BELOW
<i>amtOther</i>	Other amount value, if any (tag value 9F03) • SEE IMPORTANT NOTE BELOW
<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. 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>tagsLen</i>	The length of tags data buffer.
----------------	---------------------------------

>>>>>IMPORTANT: parameters for amount and amtOther MUST BE PASSED AS A DOUBLE VALUE WITH DECIMAL POINT. Example, do not pass 1, but instead pass 1.0 or 1.00. Otherwise, results will be unpredictable

Parameters

<i>cardType</i>	Set available card to accept. 0 = MSR only. 1 = MSR and ICC.
<i>iccReadType</i>	In case of ICC reading, this is how to behave. 0 = Same as device_startTransaction 1 = When reading ICC, read DF4F(JIS2EquivalentData) in ReadRecord. If the user swipes an IC card, the reader will ask for using ICC 3 = When reading ICC, read DF4F(JIS2EquivalentData) in ReadRecord. If the user swipes an IC card, the reader will not ask for using ICC and output MSR data directly

Returns

RETURN_CODE: Values can be parsed with [device_getIDGStatusCodeString\(\)](#) Note: if auto poll is on, it will return the error IDG_P2_STATUS_CODE_COMMAND_NOT_ALLOWED

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 device_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) 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

11.7.4.216 `msr_cancelMSRSwipe()`

```
int msr_cancelMSRSwipe ( )
```

Disable MSR Swipe Cancels MSR swipe request.

Returns

RETURN_CODE: Values can be parsed with [device_getIDGStatusCodeString\(\)](#)

11.7.4.217 `msr_registerCallBk()`

```
void msr_registerCallBk (
    pMSR_callback pMSRf )
```

To register the msr callback function to get the MSR card data. (Pass NULL to disable the callback.)

11.7.4.218 `msr_registerCallBkp()`

```
void msr_registerCallBkp (
    pMSR_callbackp pMSRf )
```

To register the msr callback function to get the MSR card data pointer. (Pass NULL to disable the callback.)

11.7.4.219 `msr_startMSRSwipe()`

```
int msr_startMSRSwipe (
    IN int _timeout )
```

Start MSR Swipe

Enables MSR, waiting for swipe to occur. Allows track selection. Returns IDTMSRData instance to `swipeMSRData()`

Parameters

<i>timeout</i>	Swipe Timeout Value
----------------	---------------------

Returns

RETURN_CODE: Values can be parsed with [device_getIDGStatusCodeString\(\)](#) Note: if auto poll mode is on, it will return command not allowed error

11.7.4.220 parseMSRData()

```
void parseMSRData (
    IN BYTE * resData,
    IN int resLen,
    IN_OUT IDTMSRData * cardData )
```

Parser the MSR data from the buffer into IDTMSRData structure

Parameters

<i>resData</i>	MSR card data buffer
<i>resLen</i>	the length of resData
<i>cardData</i>	the parser result with IDTMSTData structure

11.7.4.221 pin_cancelPINEntry()

```
int pin_cancelPINEntry ( )
```

Cancel PIN Entry

Cancels PIN entry request

Returns

RETURN_CODE: Values can be parsed with [device_getIDGStatusCodeString\(\)](#)

11.7.4.222 pin_capturePin()

```
int pin_capturePin (
    IN int timeout,
    IN int type,
    IN char * PAN,
    IN int PANLen,
    IN int minPIN,
    IN int maxPIN,
    IN char * message,
    IN int messageLen )
```

Capture PIN

Parameters

<i>timeout</i>	Timeout, in seconds. Value of 0 will use system timeout, if any
----------------	---

Parameters

<i>type</i>	PAN and Key Type <ul style="list-style-type: none">• 00h = MKSK to encrypt PIN, Internal PAN (from MSR)• 01h = DUKPT to encrypt PIN, Internal PAN (from MSR)• 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)
<i>PANLen</i>	Length of PAN
<i>minPIN</i>	Minimum PIN Length
<i>maxPIN</i>	Maximum PIN Length
<i>message</i>	LCD Message
<i>messageLen</i>	Length of message

Returns

RETURN_CODE: Values can be parsed with [device_getIDGStatusCodeString\(\)](#)

11.7.4.223 pin_capturePinExt()

```
int pin_capturePinExt (
    IN int type,
    IN char * PAN,
    IN int PANLen,
    IN int minPIN,
    IN int maxPIN,
    IN char * message,
    IN int messageLen,
    IN char * verify,
    IN int verifyLen )
```

- Capture PIN Ext

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>message</i>	LCD Message Up to 2 lines of text, each line 1-16, separated by 0x00
<i>messageLen</i>	Length of 1st scenario LCD message, valid in 00h~21h (0~33).If no LCD message input, length is 00h, and display default msg "PLEASE ENTER PIN"
<i>verify</i>	LCD Message Up to 2 lines of text, each line 1-16, separated by 0x00
<i>verifyLen</i>	Length of 2nd Scenario LCD message.valid in 00h~21h (0~33).This field is present only when PAN and Key Type has Verify PIN.If no LCD message input, length is 00h, and display default msg " ENTER PIN AGAIN"

Returns

RETURN_CODE: Values can be parsed with [device_getIDGStatusCodeString\(\)](#)

11.7.4.224 pin_getPanEntry()

```
int pin_getPanEntry (
    IN int csc,
    IN int expDate,
    IN int ADR,
    IN int ZIP,
    IN int mod10CK,
    IN int timeout,
    IN int encPANOnly )
```

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>csc</i>	Request CSS
<i>expDate</i>	Request Expiration Date
<i>ADR</i>	Request Address
<i>ZIP</i>	Request Zip
<i>mod10CK</i>	Validate entered PAN passes MOD-10 checking before accepting
<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>encPANOnly</i>	Output only encrypted PAN

Returns

RETURN_CODE: Values can be parsed with [device_getIDGStatusCodeString\(\)](#)

11.7.4.225 pin_inputFromPrompt()

```
int pin_inputFromPrompt (
    BYTE mask,
    BYTE preClearText,
    BYTE postClearText,
    int minLen,
    int maxLen,
    char * lang,
    BYTE promptID,
    char * defaultResponse,
    int defaultResponseLen,
    int timeout )
```

Get PIN Input from Prompt Results returned to PIN Callback. If successful function key capture, data is returned as IDTPINData.keyString.

```
@param mask 0 = no masking, 1 = Display "*" for numeric key according to Pre-Cleartext and Post-Cleartext
@param preClearText Range 0-6 Characters to mask at start of text if masking is on;
@param postClearText Range 0-6 Characters to mask at end of text if masking is on;
@param minLen Minimum number of digits required to input
@param maxLen Maximum number of digits allowed to be input
@param lang Valid values;
"EN" is English display message
"JP" is Japanese display message
"ES" is Spanish display message
"FR" is French display message
"ZH" is Chinese display message
"PT" is Portuguese display message
@param promptID Valid values:
0x00: Enter Phone Number
0x01: Enter IP Address
0x02: Enter Subnet Mask
0x03: Enter Default Gateway
0x04: Enter Odometer Reading/Mileage
0x05: Enter Employee ID
0x06: Enter Password for Default Factory Setting
0x07: Enter Email Address (Full keyboard)
@param defaultResponse Default String on input field
@param defaultResponseLen Length of defaultResponse
@param timeout Timeout, in seconds
```

Returns

RETURN_CODE: Values can be parsed with [device_getIDGStatusCodeString\(\)](#)

11.7.4.226 pin_promptForNumericKey()

```
int pin_promptForNumericKey (
    IN int timeout,
    IN int maskInput,
    IN int minLen,
    IN int maxLen,
    IN char * message,
    IN int messageLen,
    BYTE * signature,
    IN int signatureLen )
```

Capture Numeric Input**Parameters**

<i>timeout</i>	Timeout, in seconds. Value of 0 will use system timeout, if any
<i>maskInput</i>	<ul style="list-style-type: none"> • 0 = Display numeric for numeric key on LCD • 1 = Display * for numeric key on LCD
<i>minLen</i>	Minimum input Length
<i>maxLen</i>	Maximum input Length
<i>message</i>	Plaintext Display Message. - 16 chars max
<i>messageLen</i>	Length of 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>signatureLen</i>	Length of signature

Returns

RETURN_CODE: Values can be parsed with [device_getIDGStatusCodeString\(\)](#)

11.7.4.227 pin_promptForNumericKeyWithSwipe()

```
int pin_promptForNumericKeyWithSwipe (
    IN int timeout,
    IN BYTE function,
    IN int minLen,
    IN int maxLen,
    IN char * line1,
```

```

IN int line1Len,
IN char * line2,
IN int line2Len,
BYTE * signature,
IN int signatureLen )

```

Capture Numeric Input

Parameters

<i>timeout</i>	Timeout, in seconds. Value of 0 will use system timeout, if any
<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>line1Len</i>	Length of line1
<i>line2</i>	Line 2 of LCD Message - 16 chars max
<i>line2Len</i>	Length of line2
<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>signatureLen</i>	Length of signature

Returns

RETURN_CODE: Values can be parsed with [device_getIDGStatusCodeString\(\)](#)

11.7.4.228 pin_registerCallBk()

```

void pin_registerCallBk (
    pPIN_callback pPINf )

```

To register the pin callback function to get the PINPad data. (Pass NULL to disable the callback.)

11.7.4.229 pin_setKeyValues()

```

int pin_setKeyValues (
    int mode )

```

Set Key Values

Set return key values on or off

Parameters

<i>mode</i>	On: 1, Off: 0
-------------	---------------

Returns

RETURN_CODE: Values can be parsed with [device_getIDGStatusCodeString\(\)](#)

11.7.4.230 registerHotplugCallBk()

```
void registerHotplugCallBk (
    pMessageHotplug pMsgHotplug )
```

To register the USB HID hot-plug callback function which implemented in the application to monitor the hotplug message from the SDK.

11.7.4.231 registerLogCallBk()

```
void registerLogCallBk (
    pSendDataLog pFSend,
    pReadDataLog pFRead )
```

To register the log callback function which implemented in the application to monitor sending/reading data between application and reader.

11.7.4.232 rs232_device_init()

```
int rs232_device_init (
    int deviceType,
    int port_number,
    int brate )
```

Initial the device by RS232

It will try to connect to the device with provided deviceType, port_number, and brate.

Parameters

<i>deviceType</i>	Device to connect to
<i>port_number</i>	Port number of the device

Port nr. | Linux | Windows

| 0 | ttyS0 | COM1 |

| 1 | ttyS1 | COM2 |

| 2 | ttyS2 | COM3 |

| 3 | ttyS3 | COM4 |

| 4 | ttyS4 | COM5 |

5	ttyS5	COM6
6	ttyS6	COM7
7	ttyS7	COM8
8	ttyS8	COM9
9	ttyS9	COM10
10	ttyS10	COM11
11	ttyS11	COM12
12	ttyS12	COM13
13	ttyS13	COM14
14	ttyS14	COM15
15	ttyS15	COM16
16	ttyUSB0	n.a.
17	ttyUSB1	n.a.
18	ttyUSB2	n.a.
19	ttyUSB3	n.a.
20	ttyUSB4	n.a.
21	ttyUSB5	n.a.
22	ttyAMA0	n.a.
23	ttyAMA1	n.a.
24	ttyACM0	n.a.
25	ttyACM1	n.a.
26	rfcomm0	n.a.
27	rfcomm1	n.a.
28	ircomm0	n.a.
29	ircomm1	n.a.
30	cuau0	n.a.
31	cuau1	n.a.
32	cuau2	n.a.
33	cuau3	n.a.
34	cuaU0	n.a.
35	cuaU1	n.a.
36	cuaU2	n.a.
37	cuaU3	n.a.
38	ttyIDG	n.a.

Parameters

<i>brate</i>	Bitrate of the device
--------------	-----------------------

Returns

RETURN_CODE: Values can be parsed with [device_getIDGStatusCodeString\(\)](#)

11.7.4.233 SDK_Version()

```
char* SDK_Version ( )
```

To Get SDK version

Returns

return the SDK version string

11.7.4.234 set_open_com_port_timeout()

```
void set_open_com_port_timeout (
    int timeout )
```

Set the timeout value for opening the com port

Parameters

<i>timeout</i>	should be set to greater than 0 in seconds, otherwise there will be no timeout for opening the com port
----------------	---

11.7.4.235 setAbsoluteLibraryPath()

```
int setAbsoluteLibraryPath (
    const char * absoluteLibraryPath )
```

Set the path to use when searching for ID TECH's libraries. If this is not set, the libraries will be searched for with the system's default procedures.

Parameters

<i>absoluteLibraryPath</i>	The absolute path to ID TECH's libraries.
----------------------------	---

Returns

RETURN_CODE: Values can be parsed with [device_getIDGStatusCodeString\(\)](#)

11.8 Source_C/libIDT_PipReader.h File Reference

PipReader API.

```
#include "IDTDef.h"
```

Macros

- #define [IN](#)
- #define [OUT](#)
- #define [IN_OUT](#)

Typedefs

- typedef void(* [pMessageHotplug](#)) (int, int)
- typedef void(* [pSendDataLog](#)) (BYTE *, int)
- typedef void(* [pReadDataLog](#)) (BYTE *, int)
- typedef void(* [pEMV_callBack](#)) (int, int, BYTE *, int, IDTTransactionData *, EMV_Callback *, int)
- typedef void(* [pMSR_callBack](#)) (int, IDTMSRData)
- typedef void(* [pMSR_callBackp](#)) (int, IDTMSRData *)
- typedef void(* [pPIN_callBack](#)) (int, IDTPINData *)
- typedef void(* [pCMR_callBack](#)) (int, IDTCMRData *)
- typedef void(* [pCSFS_callBack](#)) (BYTE status)
- typedef void(* [ftpComm_callBack](#)) (int, int, int)
- typedef void(* [httpComm_callBack](#)) (BYTE *, int)
- typedef void(* [v4Comm_callBack](#)) (BYTE, BYTE, BYTE *, int)

Functions

- void [registerHotplugCallBk](#) ([pMessageHotplug](#) pMsgHotplug)
- void [registerLogCallBk](#) ([pSendDataLog](#) pFSend, [pReadDataLog](#) pFRead)
- void [emv_registerCallBk](#) ([pEMV_callBack](#) pEMVf)
- void [ctls_registerCallBk](#) ([pMSR_callBack](#) pCTLSf)
- void [ctls_registerCallBkp](#) ([pMSR_callBackp](#) pCTLSf)
- void [pin_registerCallBk](#) ([pPIN_callBack](#) pPINf)
- void [device_registerCameraCallBk](#) ([pCMR_callBack](#) pCMRf)
- void [device_registerCardStatusFrontSwitchCallBk](#) ([pCSFS_callBack](#) pCSFSf)
- char * [SDK_Version](#) ()
- int [setAbsoluteLibraryPath](#) (const char *absoluteLibraryPath)
- int [device_init](#) ()
- int [rs232_device_init](#) (int deviceType, int port_number, int brate)
- int [device_setCurrentDevice](#) (int deviceType)
- int [device_close](#) ()
- void [device_getIDGStatusCodeString](#) ([IN](#) int returnCode, [OUT](#) char *despcrition)
- int [device_isConnected](#) ()
- int [device_isAttached](#) (int deviceType)
- int [device_getFirmwareVersion](#) ([OUT](#) char *firmwareVersion)
- int [device_getFirmwareVersion_Len](#) ([OUT](#) char *firmwareVersion, [IN_OUT](#) int *firmwareVersionLen)
- int [device_pingDevice](#) ()
- int [device_controlUserInterface](#) ([IN](#) BYTE *values)
- int [device_getCurrentDeviceType](#) ()
- int [device_SendDataCommandNEO](#) ([IN](#) int cmd, [IN](#) int subCmd, [IN](#) BYTE *data, [IN](#) int dataLen, [OUT](#) BYTE *response, [IN_OUT](#) int *respLen)
- int [device_enablePassThrough](#) (int enablePassThrough)
- int [device_setBurstMode](#) ([IN](#) BYTE mode)
- int [device_setPollMode](#) ([IN](#) BYTE mode)
- int [device_setMerchantRecord](#) (int index, int enabled, char *merchantID, char *merchantURL)
- int [device_getTransactionResults](#) (IDTMSRData *cardData)
- int [device_getMerchantRecord](#) ([IN](#) int index, [OUT](#) BYTE *record)
- int [device_getMerchantRecord_Len](#) ([IN](#) int index, [OUT](#) BYTE *record, [IN_OUT](#) int *recordLen)

- int [device_getSDKWaitTime](#) ()
- void [device_setSDKWaitTime](#) (int waitTime)
- int [config_getSerialNumber](#) (OUT char *sNumber)
- int [config_getSerialNumber_Len](#) (OUT char *sNumber, IN_OUT int *sNumberLen)
- int [ctls_startTransaction](#) (IN double amount, IN double amtOther, IN int type, IN const int _timeout, IN BYTE *tags, IN int tagsLen)
- int [ctls_activateTransaction](#) (IN const int _timeout, IN BYTE *tags, IN int tagsLen)
- int [ctls_cancelTransaction](#) ()
- int [ctls_retrieveApplicationData](#) (IN BYTE *AID, IN int AIDLen, OUT BYTE *tlv, IN_OUT int *tlvLen)
- int [ctls_setApplicationData](#) (IN BYTE *tlv, IN int tlvLen)
- int [ctls_removeApplicationData](#) (IN BYTE *AID, IN int AIDLen)
- int [ctls_removeAllApplicationData](#) ()
- int [ctls_retrieveAIDList](#) (OUT BYTE *AIDList, IN_OUT int *AIDListLen)
- int [ctls_retrieveTerminalData](#) (OUT BYTE *tlv, IN_OUT int *tlvLen)
- int [ctls_setTerminalData](#) (IN BYTE *tlv, IN int tlvLen)
- int [ctls_retrieveCAPK](#) (IN BYTE *capk, IN int capkLen, OUT BYTE *key, IN_OUT int *keyLen)
- int [ctls_setCAPK](#) (IN BYTE *capk, IN int capkLen)
- int [ctls_removeCAPK](#) (IN BYTE *capk, IN int capkLen)
- int [ctls_removeAllCAPK](#) ()
- int [ctls_retrieveCAPKList](#) (OUT BYTE *keys, IN_OUT int *keysLen)
- int [ctls_setConfigurationGroup](#) (IN BYTE *tlv, IN int tlvLen)
- int [ctls_getConfigurationGroup](#) (IN int group, OUT BYTE *tlv, OUT int *tlvLen)
- int [ctls_getAllConfigurationGroups](#) (OUT BYTE *tlv, IN_OUT int *tlvLen)
- int [ctls_removeConfigurationGroup](#) (int group)
- void [parseMSRData](#) (IN BYTE *resData, IN int resLen, IN_OUT IDTMSRData *cardData)

11.8.1 Detailed Description

PipReader API.

PipReader Global API methods.

11.8.2 Macro Definition Documentation

11.8.2.1 IN

```
#define IN
```

INPUT parameter.

11.8.2.2 IN_OUT

```
#define IN_OUT
```

INPUT / OUTPUT PARAMETER.

11.8.2.3 OUT

```
#define OUT
```

OUTPUT parameter.

11.8.3 Typedef Documentation

11.8.3.1 ftpComm_callBack

```
typedef void(* ftpComm_callBack) (int, int, int)
```

Define the comm callback function to get FTP file transfer status

It should be passed as a parameter in a FTP request, Signature (int, int, int) = response code, current block, total blocks
RESPONSE CODES: 100 = FILE DOWNLOAD STARTED 101 = FILE BLOCK XX OF XX RECEIVED 102 = FILE DOWNLOAD COMPLETED 103 = FILE DOWNLOAD TERMINATED PREMATURELY

11.8.3.2 httpComm_callBack

```
typedef void(* httpComm_callBack) (BYTE *, int)
```

Define the comm callback function to get the async url data

It should be registered using the comm_registerHTTPCallback

11.8.3.3 pCMR_callBack

```
typedef void(* pCMR_callBack) (int, IDTCMRData *)
```

Define the camera callback function to get the image data

It should be registered using the device_registerCameraCallBk,

11.8.3.4 pCSFS_callBack

```
typedef void(* pCSFS_callBack) (BYTE status)
```

Define the card status and front switch callback function to get card and front switch status

It should be registered using the device_registerCardStatusFrontSwitchCallBk,

11.8.3.5 pEMV_callBack

```
typedef void(* pEMV_callBack) (int, int, BYTE *, int, IDTTransactionData *, EMV_Callback *, int)
```

Define the EMV callback function to get the transaction message/data/result.

It should be registered using the emv_registerCallBk,

11.8.3.6 pMessageHotplug

```
typedef void(* pMessageHotplug) (int, int)
```

Define the USB hot-plug callback function to monitor the info when plug in/out the reader.

It should be registered using the registerHotplugCallBk, The first integer parameter is device type, and the second integer parameter is either 0: Device Plugged Out or 1: Device Plugged In

11.8.3.7 pMSR_callBack

```
typedef void(* pMSR_callBack) (int, IDTMSRData)
```

Define the MSR callback function to get the MSR card data

It should be registered using the msr_registerCallBk, this callback function is for backward compatibility

11.8.3.8 pMSR_callBackp

```
typedef void(* pMSR_callBackp) (int, IDTMSRData *)
```

Define the MSR callback function to get pointer to the MSR card data

It should be registered using the `msr_registerCallBk`, this callback function is recommended instead of `pMSR_callBack`

11.8.3.9 pPIN_callBack

```
typedef void(* pPIN_callBack) (int, IDTPINData *)
```

Define the PINPad callback function to get the input PIN Pad data

It should be registered using the `pin_registerCallBk`,

11.8.3.10 pReadDataLog

```
typedef void(* pReadDataLog) (BYTE *, int)
```

Define the read response callback function to monitor the reading response from the reader.

It should be registered using the `registerLogCallBk`,

11.8.3.11 pSendDataLog

```
typedef void(* pSendDataLog) (BYTE *, int)
```

Define the send command callback function to monitor the sending command into the reader.

It should be registered using the `registerLogCallBk`,

11.8.3.12 v4Comm_callBack

```
typedef void(* v4Comm_callBack) (BYTE, BYTE, BYTE *, int)
```

Define the comm callback function to receive the V4 Protocol packets received by the device from an external source (IP/USB/RS-232) It should be registered using the `comm_registerV4Callback`, Data callback will contain command, sub-command, and data from V4 packet

11.8.4 Function Documentation

11.8.4.1 config_getSerialNumber()

```
int config_getSerialNumber (
    OUT char * sNumber )
```

DEPRECATED : please use `config_getSerialNumber_Len(OUT char* sNumber, IN_OUT int *sNumberLen)`

Polls device for Serial Number

Parameters

<code>sNumber</code>	Returns Serial Number; needs to have at least 64 bytes of memory
----------------------	--

Returns

RETURN_CODE: Values can be parsed with device_getIDGStatusCodeString

11.8.4.2 config_getSerialNumber_Len()

```
int config_getSerialNumber_Len (
    OUT char * sNumber,
    IN_OUT int * sNumberLen )
```

Polls device for Serial Number

Parameters

<i>sNumber</i>	Returns Serial Number
<i>sNumberLen</i>	Length of Serial Number

Returns

RETURN_CODE: Values can be parsed with device_getIDGStatusCodeString

11.8.4.3 ctls_activateTransaction()

```
int ctls_activateTransaction (
    IN const int _timeout,
    IN BYTE * tags,
    IN int tagsLen )
```

Start CTLS Transaction Request

Authorizes the CTLS 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 stream. Example, tag 9F0C with amount 0x000000000100 would be 0x9F0C06000000000100
<i>tagsLen</i>	The length of tags data buffer.

>>>>>IMPORTANT: parameters for amount and amtOther MUST BE PASSED AS A DOUBLE VALUE WITH DECIMAL POINT. Example, do not pass 1, but instead pass 1.0 or 1.00. Otherwise, results will be unpredictable

Returns

RETURN_CODE: Values can be parsed with [device_getIDGStatusCodeString\(\)](#) Note: if auto poll is on, it will return the error IDG_P2_STATUS_CODE_COMMAND_NOT_ALLOWED

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

11.8.4.4 `ctls_cancelTransaction()`

```
int ctls_cancelTransaction ( )
```

Cancel EMV Transaction

Cancels the currently executing EMV transaction.

Returns

RETURN_CODE: Values can be parsed with [device_getIDGStatusCodeString\(\)](#)

11.8.4.5 ctls_getAllConfigurationGroups()

```
int ctls_getAllConfigurationGroups (
    OUT BYTE * tlv,
    IN_OUT int * tlvLen )
```

Retrieve All Configuration Groups

Returns all the Configuration Groups installed on the terminal for CTLS

Parameters

<i>tlv</i>	The TLV elements data
<i>tlvLen</i>	the length of tlv data buffer.

Returns

RETURN_CODE: Values can be parsed with [device_getIDGStatusCodeString\(\)](#)

11.8.4.6 ctls_getConfigurationGroup()

```
int ctls_getConfigurationGroup (
    IN int group,
    OUT BYTE * tlv,
    OUT int * tlvLen )
```

Get Configuration Group

Retrieves the Configuration for the specified Group.

Parameters

<i>group</i>	Configuration Group
<i>tlv</i>	return data
<i>tlvLen</i>	the length of tlv data buffer

Returns

RETURN_CODE: Values can be parsed with [device_getIDGStatusCodeString\(\)](#)

11.8.4.7 ctls_registerCallBk()

```
void ctls_registerCallBk (
    pMSR_callBack pCTLSf )
```

To register the ctls callback function to get the MSR card data. (Pass NULL to disable the callback.)

11.8.4.8 ctls_registerCallBkp()

```
void ctls_registerCallBkp (
    pMSR_callBackp pCTLSf )
```

To register the ctls callback function to get the MSR card data pointer. (Pass NULL to disable the callback.)

11.8.4.9 ctls_removeAllApplicationData()

```
int ctls_removeAllApplicationData ( )
```

Remove All Application Data

Removes all the Application Data

Returns

RETURN_CODE: Values can be parsed with [device_getIDGStatusCodeString\(\)](#)

11.8.4.10 ctls_removeAllCAPK()

```
int ctls_removeAllCAPK ( )
```

Remove All Certificate Authority Public Key

Removes all the CAPK

Returns

RETURN_CODE: Values can be parsed with [device_getIDGStatusCodeString\(\)](#)

11.8.4.11 ctls_removeApplicationData()

```
int ctls_removeApplicationData (
    IN BYTE * AID,
    IN int AIDLen )
```

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>AIDLen</i>	the length of AID data buffer

Returns

RETURN_CODE: Values can be parsed with [device_getIDGStatusCodeString\(\)](#)

11.8.4.12 ctls_removeCAPK()

```
int ctls_removeCAPK (
    IN BYTE * capk,
    IN int capkLen )
```

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>capkLen</i>	the length of capk data buffer

Returns

RETURN_CODE: Values can be parsed with [device_getIDGStatusCodeString\(\)](#)

11.8.4.13 ctls_removeConfigurationGroup()

```
int ctls_removeConfigurationGroup (  
    int group )
```

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>	Values can be parsed with device_getIDGStatusCodeString()
--------------------	---

11.8.4.14 ctls_retrieveAIDList()

```
int ctls_retrieveAIDList (  
    OUT BYTE * AIDList,  
    IN_OUT int * AIDListLen )
```

Retrieve AID list

Returns all the AID names installed on the terminal for CTLS. .

Parameters

<i>AIDList</i>	array of AID name byte arrays
<i>AIDListLen</i>	the length of AIDList array buffer

Returns

RETURN_CODE: Values can be parsed with [device_getIDGStatusCodeString\(\)](#)

11.8.4.15 ctls_retrieveApplicationData()

```
int ctls_retrieveApplicationData (  
    IN BYTE * AID,
```

```

    IN int AIDLen,
    OUT BYTE * tlv,
    IN_OUT int * tlvLen )

```

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>AIDLen</i>	the length of AID data buffer.
<i>tlv</i>	The TLV elements of the requested AID
<i>tlvLen</i>	the length of tlv data buffer.

Returns

RETURN_CODE: Values can be parsed with [device_getIDGStatusCodeString\(\)](#)

11.8.4.16 ctls_retrieveCAPK()

```

int ctls_retrieveCAPK (
    IN BYTE * capk,
    IN int capkLen,
    OUT BYTE * key,
    IN_OUT int * keyLen )

```

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>capkLen</i>	the length of capk data buffer
<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>keyLen</i>	<p>the length of key data buffer</p> <ul style="list-style-type: none"> •

Returns

RETURN_CODE: Values can be parsed with [device_getIDGStatusCodeString\(\)](#)

11.8.4.17 ctls_retrieveCAPKList()

```
int ctls_retrieveCAPKList (
    OUT BYTE * keys,
    IN_OUT int * keysLen )
```

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>keysLen</i>	the length of keys data buffer

Returns

RETURN_CODE: Values can be parsed with [device_getIDGStatusCodeString\(\)](#)

11.8.4.18 ctls_retrieveTerminalData()

```
int ctls_retrieveTerminalData (
    OUT BYTE * tlv,
    IN_OUT int * tlvLen )
```

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>tlvLen</i>	the length of tlv data buffer

Returns

RETURN_CODE: Values can be parsed with [device_getIDGStatusCodeString\(\)](#)

11.8.4.19 ctls_setApplicationData()

```
int ctls_setApplicationData (
    IN BYTE * tlv,
    IN int tlvLen )
```

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>tlvLen</i>	the length of tlv data buffer
---------------	-------------------------------

Returns

RETURN_CODE: Values can be parsed with [device_getIDGStatusCodeString\(\)](#)

11.8.4.20 `ctls_setCAPK()`

```
int ctls_setCAPK (
    IN BYTE * capk,
    IN int capkLen )
```

Set Certificate Authority Public Key

Sets the CAPK as specified by the CAKey structure

Parameters

<i>capk</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>capkLen</i>	the length of capk data buffer

Returns

RETURN_CODE: Values can be parsed with [device_getIDGStatusCodeString\(\)](#)

11.8.4.21 ctls_setConfigurationGroup()

```
int ctls_setConfigurationGroup (
    IN BYTE * tlv,
    IN int tlvLen )
```

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>tlvLen</i>	the length of tlv data buffer

Returns

RETURN_CODE: Values can be parsed with [device_getIDGStatusCodeString\(\)](#)

11.8.4.22 ctls_setTerminalData()

```
int ctls_setTerminalData (
    IN BYTE * tlv,
    IN int tlvLen )
```

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 [ctls_getConfigurationGroup\(int group\)](#), and deleted with [ctls_removeConfigurationGroup\(int group\)](#). You cannot delete group 0.

Parameters

<i>tlv</i>	TerminalData configuration file
<i>tlvLen</i>	the length of tlv data buffer

Return values

<i>RETURN_CODE</i>	Values can be parsed with device_getIDGStatusCodeString()
--------------------	---

11.8.4.23 ctls_startTransaction()

```
int ctls_startTransaction (
    IN double amount,
    IN double amtOther,
    IN int type,
    IN const int _timeout,
    IN BYTE * tags,
    IN int tagsLen )
```

Start CTLS Transaction Request

Authorizes the CTLS transaction for an ICC card

The tags will be returned in the callback routine.

Parameters

<i>amount</i>	Transaction amount value (tag value 9F02) - SEE IMPORTANT NOTE BELOW
<i>amtOther</i>	Other amount value, if any (tag value 9F03) - SEE IMPORTANT NOTE BELOW
<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 stream. Example, tag 9F0C with amount 0x000000000100 would be 0x9F0C06000000000100 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>tagsLen</i>	The length of tags data buffer.

>>>>>IMPORTANT: parameters for amount and amtOther MUST BE PASSED AS A DOUBLE VALUE WITH DECIMAL POINT. Example, do not pass 1, but instead pass 1.0 or 1.00. Otherwise, results will be unpredictable

Returns

RETURN_CODE: Values can be parsed with [device_getIDGStatusCodeString\(\)](#) Note: if auto poll is on, it will return the error IDG_P2_STATUS_CODE_COMMAND_NOT_ALLOWED

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

11.8.4.24 device_close()

```
int device_close ( )
```

Close the device

Returns

RETURN_CODE: 0: success, 0x0A: failed

11.8.4.25 device_controlUserInterface()

```
int device_controlUserInterface (
    IN BYTE * values )
```

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) • 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
---------------	--

Returns

RETURN_CODE: Values can be parsed with [device_getIDGStatusCodeString\(\)](#)

11.8.4.26 device_enablePassThrough()

```
int device_enablePassThrough (
    int enablePassThrough )
```

Enable Pass Through - NEO

Enables Pass Through Mode for direct communication with L1 interface (power on icc, send apdu, etc).

Parameters

<i>enablePassThrough</i>	1 = pass through ON, 0 = pass through OFF
--------------------------	---

Returns

RETURN_CODE: Values can be parsed with [device_getIDGStatusCodeString\(\)](#)

11.8.4.27 device_getCurrentDeviceType()

```
int device_getCurrentDeviceType ( )
```

Get current active device type

Returns

: return the device type defined as DEVICE_TYPE in the IDTDef.h

11.8.4.28 device_getFirmwareVersion()

```
int device_getFirmwareVersion (
    OUT char * firmwareVersion )
```

DEPRECATED : please use [device_getFirmwareVersion_Len\(\)](#)(OUT char* firmwareVersion, IN_OUT int *firmwareVersionLen)

Polls device for Firmware Version

Parameters

<i>firmwareVersion</i>	Response returned of Firmware Version; needs to have at least 128 bytes of memory
------------------------	---

Returns

RETURN_CODE: Values can be parsed with [device_getResponseCodeString\(\)](#)

11.8.4.29 device_getFirmwareVersion_Len()

```
int device_getFirmwareVersion_Len (
    OUT char * firmwareVersion,
    IN_OUT int * firmwareVersionLen )
```

Polls device for Firmware Version

Parameters

<i>firmwareVersion</i>	Response returned of Firmware Version
<i>firmwareVersionLen</i>	Length of Firmware Version

Returns

RETURN_CODE: Values can be parsed with [device_getResponseCodeString\(\)](#)

11.8.4.30 device_getIDGStatusCodeString()

```
void device_getIDGStatusCodeString (
    IN int  returnCode,
    OUT char * despcrition )
```

Review the return code description.

Parameters

<i>returnCode</i>	the response result.
<i>description</i>	

Return values

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

- 0: "no error, beginning task";
- 1: "no response from reader";
- 2: "invalid response data";
- 01: " Incorrect Header Tag";
- 02: " Unknown Command";
- 03: " Unknown Sub-Command";
- 04: " CRC Error in Frame";
- 05: " Incorrect Parameter";
- 06: " Parameter Not Supported";
- 07: " Mal-formatted Data";
- 08: " Timeout";
- 0A: " Failed / NACK";
- 0B: " Command not Allowed";
- 0C: " Sub-Command not Allowed";
- 0D: " Buffer Overflow (Data Length too large for reader buffer)";

- 0E: " User Interface Event";
- 10: " Need clear firmware(apply in boot loader only)";
- 11: " Communication type not supported, VT-1, burst, etc. Need encrypted firmware (apply in boot loader only)";
- 12: " Secure interface is not functional or is in an intermediate state.";
- 13: " Data field is not mod 8";
- 14: " Pad 0x80 not found where expected";
- 15: " Specified key type is invalid";
- 16: " Could not retrieve key from the SAM (InitSecureComm)";
- 17: " Hash code problem";
- 18: " Could not store the key into the SAM (InstallKey)";
- 19: " Frame is too large";
- 1A: " Unit powered up in authentication state but POS must resend the InitSecureComm command";
- 1B: " The EEPROM may not be initialized because SecCommInterface does not make sense";
- 1C: " Problem encoding APDU Module-Specific Status Codes ";
- 20: " Unsupported Index (ILM) SAM Transceiver error - problem communicating with the SAM (Key Mgr)";
- 21: " Unexpected Sequence Counter in multiple frames for single bitmap (ILM)Length error in data returned from the SAM (Key Mgr) - 22: " Improper bit map (ILM)"; - 23: " Request Online Authorization"; - 24: " ViV \leftrightarrow OCard3 raw data read successful"; - 25: " Message index not available (ILM) ViVOcomm activate transaction card type (ViVOcomm)"; - 26: " Version Information Mismatch (ILM)"; - 27: " Not sending commands in correct index message index (ILM)"; - 28: " Time out or next expected message not received (ILM)"; - 29: " ILM languages not available for viewing (ILM)"; - 2A: " Other language not supported (ILM)"; - 41: " from 41 to 4F, Module-specific errors for Key Manager";
- 50: " Auto-Switch OK";
- 51: " Auto-Switch failed";
- 70: " Antenna Error 80h Use another card";
- 81: " Insert or swipe card";
- 90: " Data encryption Key does not exist";
- 91: " Data encryption Key KSN exhausted";

11.8.4.31 device_getMerchantRecord()

```
int device_getMerchantRecord (
    IN int index,
    OUT BYTE * record )
```

DEPRECATED : please use device_getMerchantRecord_Len(IN int index, OUT BYTE * record, IN_OUT int *recordLen)

Get Merchant Record

Gets the merchant record for the device.

Parameters

<i>index</i>	Merchant Record index, valid values 1-6
<i>record;</i>	needs to have at least 99 bytes of memory response data from reader. Merchant Record Index: 1 byte enabled: 1 byte Merchant ID: 32 bytes Length of Merchant URL: 1 byte Merchant URL: 64 bytes

Returns

success or error code. Values can be parsed with [device_getIDGStatusCodeString\(\)](#)

See also

[ErrorCode](#)

11.8.4.32 device_getMerchantRecord_Len()

```
int device_getMerchantRecord_Len (
    IN int index,
    OUT BYTE * record,
    IN_OUT int * recordLen )
```

Get Merchant Record

Gets the merchant record for the device.

Parameters

<i>index</i>	Merchant Record index, valid values 1-6
<i>record</i>	response data from reader. Merchant Record Index: 1 byte enabled: 1 byte Merchant ID: 32 bytes Length of Merchant URL: 1 byte Merchant URL: 64 bytes
<i>recordLen</i>	Length of record

Returns

success or error code. Values can be parsed with [device_getIDGStatusCodeString\(\)](#)

See also

[ErrorCode](#)

11.8.4.33 device_getSDKWaitTime()

```
int device_getSDKWaitTime ( )
```

Get SDK Wait Time

Get the SDK wait time for transactions

Returns

SDK wait time in seconds

11.8.4.34 device_getTransactionResults()

```
int device_getTransactionResults (
    IDTMSRData * cardData )
```

Get Transaction Results Gets the transaction results when the reader is functioning in "Auto Poll" mode

Parameters

<i>cardData</i>	The transaction results
-----------------	-------------------------

Returns

success or error code. Values can be parsed with [device_getResponseCodeString](#)

See also

[ErrorCode](#)

11.8.4.35 device_init()

```
int device_init ( )
```

Initial the device by USB

It will detect the device and trying connect.

The connect status can be checked by [device_isConnected\(\)](#).

Note: after the function returns success, the function [device_setCurrentDevice\(\)](#) has to be called to set the device type.

Returns

RETURN_CODE: Values can be parsed with [device_getIDGStatusCodeString\(\)](#)

11.8.4.36 device_isAttached()

```
int device_isAttached (
    int deviceType )
```

Check if the device is attached to the USB port The function [device_init\(\)](#) must be called before this function.

Parameters

<i>deviceType,the</i>	device type of the USB device
-----------------------	-------------------------------

Returns

1 if the device is attached, or 0 if the device is not attached

11.8.4.37 device_isConnected()

```
int device_isConnected ( )
```

Check the device connected status

Returns

DEVICE_DISCONNECT=0, or DEVICE_CONNECTED = 1

11.8.4.38 device_pingDevice()

```
int device_pingDevice ( )
```

Ping Device

Pings the reader. If connected, returns success. Otherwise, returns timeout.

Returns

RETURN_CODE: Values can be parsed with [device_getIDGStatusCodeString\(\)](#)

11.8.4.39 device_registerCameraCallBk()

```
void device_registerCameraCallBk (
    pCMR_callback pCMRf )
```

To register the camera callback function to get the image data. (Pass NULL to disable the callback.)

11.8.4.40 device_registerCardStatusFrontSwitchCallBk()

```
void device_registerCardStatusFrontSwitchCallBk (
    pCSFS_callback pCSFSf )
```

To register the card status and front switch callback function to get status. (Pass NULL to disable the callback.)

11.8.4.41 device_SendDataCommandNEO()

```
int device_SendDataCommandNEO (
    IN int cmd,
    IN int subCmd,
    IN BYTE * data,
    IN int dataLen,
    OUT BYTE * response,
    IN_OUT int * respLen )
```

Send a Command to NEO device

Sends a command to the NEO device .

Parameters

<i>cmd</i>	command to execute.
<i>subCmd,sub</i>	command to execute.
<i>data</i>	buffer of NEO command data.

Parameters

<i>dataLen,the</i>	length of the buffer data.
<i>response</i>	Response data
<i>respLen,the</i>	length of Response data

Returns

RETURN_CODE: Values can be parsed with [device_getIDGStatusCodeString\(\)](#)

11.8.4.42 device_setBurstMode()

```
int device_setBurstMode (
    IN BYTE mode )
```

Send Burst Mode - NEO

Sets the burst mode for the device.

Parameters

<i>mode</i>	0 = OFF, 1 = Always On, 2 = Auto Exit
-------------	---------------------------------------

Returns

success or error code. Values can be parsed with [device_getIDGStatusCodeString](#)

See also

[ErrorCode](#)

11.8.4.43 device_setCurrentDevice()

```
int device_setCurrentDevice (
    int deviceType )
```

Sets the current device to talk to

The connect status can be checked by [device_isConnected\(\)](#).

Parameters

<i>deviceType</i>	<p>Device to connect to</p> <pre>enum DEVICE_TYPE { IDT_DEVICE_UNKNOWN=0, IDT_DEVICE_AUGUSTA_HID, IDT_DEVICE_AUGUSTA_KB, IDT_DEVICE_AUGUSTA_S_HID, IDT_DEVICE_AUGUSTA_S_KB, IDT_DEVICE_AUGUSTA_S_TTK_HID, IDT_DEVICE_SPECTRUM_PRO, IDT_DEVICE_MINISMART_II, IDT_DEVICE_L80, IDT_DEVICE_L100, IDT_DEVICE_UNIPAY, IDT_DEVICE_UNIPAY_I_V, IDT_DEVICE_VP3300_AJ, IDT_DEVICE_KIOSK_III, IDT_DEVICE_KIOSK_III_S, IDT_DEVICE_PIP_READER, IDT_DEVICE_VENDI, IDT_DEVICE_VP3300_USB, IDT_DEVICE_UNIPAY_I_V_TTK, IDT_DEVICE_VP3300_BT, IDT_DEVICE_VP8800, IDT_DEVICE_SREDKEY2_HID, IDT_DEVICE_SREDKEY2_KB, IDT_DEVICE_NEO2, IDT_DEVICE_MINISMART_II_COM = IDT_DEVICE_NEO2+5, IDT_DEVICE_SPECTRUM_PRO_COM, IDT_DEVICE_KIOSK_III_COM, IDT_DEVICE_KIOSK_III_S_COM, IDT_DEVICE_PIP_READER_COM, IDT_DEVICE_VP3300_COM, IDT_DEVICE_NEO2_COM, IDT_DEVICE_MAX_DEVICES = IDT_DEVICE_NEO2_COM+5 };</pre>
-------------------	---

Returns

RETURN_CODE: 1: success, 0: failed

11.8.4.44 device_setMerchantRecord()

```
int device_setMerchantRecord (
    int index,
    int enabled,
    char * merchantID,
    char * merchantURL )
```

Set Merchant Record - NEO 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

Returns

RETURN_CODE: Values can be parsed with [device_getIDGStatusCodeString\(\)](#)

11.8.4.45 device_setPollMode()

```
int device_setPollMode (
    IN BYTE mode )
```

Set Poll Mode - NEO

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
-------------	-----------------------------------

Returns

RETURN_CODE: Values can be parsed with [device_getIDGStatusCodeString\(\)](#)

11.8.4.46 device_setSDKWaitTime()

```
void device_setSDKWaitTime (
    int waitTime )
```

Set SDK Wait Time

Set the SDK wait time for transactions

Parameters

<i>waitTime</i>	The SDK wait time for transaction in seconds. The Maximum is 2147483 seconds.
-----------------	---

11.8.4.47 emv_registerCallBk()

```
void emv_registerCallBk (
    pEMV_callback pEMVf )
```

To register the emv callback function to get the EMV processing response. (Pass NULL to disable the callback.)

11.8.4.48 parseMSRData()

```
void parseMSRData (
    IN BYTE * resData,
    IN int resLen,
    IN_OUT IDTMSRData * cardData )
```

Parser the MSR data from the buffer into IDTMSTData structure

Parameters

<i>resData</i>	MSR card data buffer
<i>resLen</i>	the length of resData
<i>cardData</i>	the parser result with IDTMSTData structure

11.8.4.49 pin_registerCallBk()

```
void pin_registerCallBk (
    pPIN_callback pPINf )
```

To register the pin callback function to get the PINPad data. (Pass NULL to disable the callback.)

11.8.4.50 registerHotplugCallBk()

```
void registerHotplugCallBk (
    pMessageHotplug pMsgHotplug )
```

To register the USB HID hot-plug callback function which implemented in the application to monitor the hotplug message from the SDK.

11.8.4.51 registerLogCallBk()

```
void registerLogCallBk (
    pSendDataLog pFSend,
    pReadDataLog pFRead )
```

To register the log callback function which implemented in the application to monitor sending/reading data between application and reader.

11.8.4.52 rs232_device_init()

```
int rs232_device_init (
    int deviceType,
    int port_number,
    int brate )
```

Initial the device by RS232

It will try to connect to the device with provided deviceType, port_number, and brate.

Parameters

<i>deviceType</i>	Device to connect to
<i>port_number</i>	Port number of the device

Port nr. | Linux | Windows

```
| 0 | ttyS0 | COM1 | | 1 | ttyS1 | COM2 | | 2 | ttyS2 | COM3 | | 3 | ttyS3 | COM4 | | 4 | ttyS4 | COM5 | | 5 | ttyS5 |
COM6 | | 6 | ttyS6 | COM7 | | 7 | ttyS7 | COM8 | | 8 | ttyS8 | COM9 | | 9 | ttyS9 | COM10 | | 10 | ttyS10 | COM11
| | 11 | ttyS11 | COM12 | | 12 | ttyS12 | COM13 | | 13 | ttyS13 | COM14 | | 14 | ttyS14 | COM15 | | 15 | ttyS15 |
COM16 | | 16 | ttyUSB0 | n.a. | | 17 | ttyUSB1 | n.a. | | 18 | ttyUSB2 | n.a. | | 19 | ttyUSB3 | n.a. | | 20 | ttyUSB4 |
n.a. | | 21 | ttyUSB5 | n.a. | | 22 | ttyAMA0 | n.a. | | 23 | ttyAMA1 | n.a. | | 24 | ttyACM0 | n.a. | | 25 | ttyACM1 | n.a.
| | 26 | rfcomm0 | n.a. | | 27 | rfcomm1 | n.a. | | 28 | ircomm0 | n.a. | | 29 | ircomm1 | n.a. | | 30 | cuau0 | n.a. | | 31
| cuau1 | n.a. | | 32 | cuau2 | n.a. | | 33 | cuau3 | n.a. | | 34 | cuaU0 | n.a. | | 35 | cuaU1 | n.a. | | 36 | cuaU2 | n.a. |
| 37 | cuaU3 | n.a. |
```

Parameters

<i>brate</i>	Bitrate of the device
--------------	-----------------------

Returns

RETURN_CODE: Values can be parsed with [device_getIDGStatusCodeString\(\)](#)

11.8.4.53 SDK_Version()

```
char* SDK_Version ( )
```

To Get SDK version

Returns

return the SDK version string

11.8.4.54 setAbsoluteLibraryPath()

```
int setAbsoluteLibraryPath (
    const char * absoluteLibraryPath )
```

Set the path to use when searching for ID TECH's libraries. If this is not set, the libraries will be searched for with the system's default procedures.

Parameters

<i>absoluteLibraryPath</i>	The absolute path to ID TECH's libraries.
----------------------------	---

Returns

RETURN_CODE: Values can be parsed with [device_getIDGStatusCodeString\(\)](#)

11.9 Source_C/libIDT_SpectrumPro.h File Reference

SpectrumPro API.

```
#include "IDTDef.h"
```

Macros

- `#define IN`
- `#define OUT`
- `#define IN_OUT`

Typedefs

- `typedef void(* pMessageHotplug) (int, int)`
- `typedef void(* pSendDataLog) (BYTE *, int)`
- `typedef void(* pReadDataLog) (BYTE *, int)`
- `typedef void(* pEMV_callBack) (int, int, BYTE *, int, IDTTransactionData *, EMV_Callback *, int)`
- `typedef void(* pMSR_callBack) (int, IDTMSRData)`
- `typedef void(* pMSR_callBackp) (int, IDTMSRData *)`
- `typedef void(* pPIN_callBack) (int, IDTPINData *)`
- `typedef void(* pCMR_callBack) (int, IDTCMRData *)`
- `typedef void(* pCSFS_callBack) (BYTE status)`
- `typedef void(* ftpComm_callBack) (int, int, int)`
- `typedef void(* httpComm_callBack) (BYTE *, int)`
- `typedef void(* v4Comm_callBack) (BYTE, BYTE, BYTE *, int)`

Functions

- `void registerHotplugCallBk (pMessageHotplug pMsgHotplug)`
- `void registerLogCallBk (pSendDataLog pFSend, pReadDataLog pFRead)`
- `void emv_registerCallBk (pEMV_callBack pEMVf)`
- `void msr_registerCallBk (pMSR_callBack pMSRf)`
- `void msr_registerCallBkp (pMSR_callBackp pMSRf)`
- `void pin_registerCallBk (pPIN_callBack pPINf)`
- `void device_registerCameraCallBk (pCMR_callBack pCMRf)`
- `void device_registerCardStatusFrontSwitchCallBk (pCSFS_callBack pCSFSf)`
- `char * SDK_Version ()`
- `int setAbsoluteLibraryPath (const char *absoluteLibraryPath)`
- `int device_init ()`
- `int rs232_device_init (int deviceType, int port_number, int brate)`
- `int device_setCurrentDevice (int deviceType)`
- `int device_close ()`

- void `device_getResponseCodeString` (IN int returnCode, OUT char *despcriton)
- int `device_isConnected` ()
- int `device_isAttached` (int deviceType)
- int `device_getFirmwareVersion` (OUT char *firmwareVersion)
- int `device_getFirmwareVersion_Len` (OUT char *firmwareVersion, IN_OUT int *firmwareVersionLen)
- int `device_getCurrentDeviceType` ()
- int `device_SendDataCommand` (IN BYTE *cmd, IN int cmdLen, IN BYTE *data, IN int dataLen, OUT BYTE *response, IN_OUT int *respLen)
- int `device_rebootDevice` ()
- int `device_updateFirmware` (IN BYTE *firmwareData, IN int firmwareDataLen, IN char *firmwareName, IN int encryptionType, IN BYTE *keyBlob, IN int keyBlobLen)
- int `config_getModelNumber` (OUT char *sNumber)
- int `config_getModelNumber_Len` (OUT char *sNumber, IN_OUT int *sNumberLen)
- int `config_getSerialNumber` (OUT char *sNumber)
- int `config_getSerialNumber_Len` (OUT char *sNumber, IN_OUT int *sNumberLen)
- int `device_pollCardReader` (OUT BYTE *status)
- int `device_pollCardReader_Len` (OUT BYTE *status, IN_OUT int *statusLen)
- int `device_getSpectrumProKSN` (IN int type, OUT BYTE *KSN)
- int `device_getSpectrumProKSN_Len` (IN int type, OUT BYTE *KSN, IN_OUT int *KSNLen)
- int `device_getSDKWaitTime` ()
- void `device_setSDKWaitTime` (int waitTime)
- int `device_getThreadStackSize` ()
- void `device_setThreadStackSize` (int threadSize)
- int `icc_powerOnICC` (OUT BYTE *ATR, IN_OUT int *inLen)
- int `icc_powerOffICC` ()
- int `icc_getICCReaderStatus` (OUT BYTE *status)
- int `emv_getEMVKernelVersion` (OUT char *version)
- int `emv_getEMVKernelVersion_Len` (OUT char *version, IN_OUT int *versionLen)
- int `emv_getEMVKernelCheckValue` (OUT BYTE *checkValue, IN_OUT int *checkValueLen)
- void `emv_setAutoAuthenticateTransaction` (IN int authenticate)
- void `emv_setAutoCompleteTransaction` (IN int complete)
- int `emv_getAutoAuthenticateTransaction` ()
- int `emv_getAutoCompleteTransaction` ()
- int `emv_getEMVConfigurationCheckValue` (OUT BYTE *checkValue, IN_OUT int *checkValueLen)
- void `emv_allowFallback` (IN int allow)
- int `emv_startTransaction` (IN double amount, IN double amtOther, IN int exponent, IN int type, IN int timeout, IN BYTE *tags, IN int tagsLen, IN int forceOnline)
- int `emv_activateTransaction` (IN int timeout, IN BYTE *tags, IN int tagsLen, IN int forceOnline)
- int `emv_authenticateTransaction` (IN BYTE *updatedTLV, IN int updatedTLVLen)
- int `emv_authenticateTransactionWithTimeout` (IN int timeout, IN BYTE *updatedTLV, IN int updatedTLVLen)
- int `emv_completeTransaction` (IN int commError, IN BYTE *authCode, IN int authCodeLen, IN BYTE *iad, IN int iadLen, IN BYTE *tlvScripts, IN int tlvScriptsLen, IN BYTE *tlv, IN int tlvLen)
- int `emv_cancelTransaction` ()
- int `emv_retrieveTransactionResult` (IN BYTE *tags, IN int tagsLen, IDTTransactionData *cardData)
- int `emv_callbackResponseLCD` (IN int type, byte selection)
- int `emv_callbackResponseMSR` (IN BYTE *MSR, IN_OUT int MSRLen)
- int `emv_retrieveApplicationData` (IN BYTE *AID, IN int AIDLen, OUT BYTE *tlv, IN_OUT int *tlvLen)
- int `emv_setApplicationData` (IN BYTE *name, IN int nameLen, IN BYTE *tlv, IN int tlvLen)
- int `emv_removeApplicationData` (IN BYTE *AID, IN int AIDLen)
- int `emv_removeAllApplicationData` ()
- int `emv_retrieveAIDList` (OUT BYTE *AIDList, IN_OUT int *AIDListLen)
- int `emv_retrieveTerminalData` (OUT BYTE *tlv, IN_OUT int *tlvLen)
- int `emv_setTerminalData` (IN BYTE *tlv, IN int tlvLen)
- int `emv_removeTerminalData` ()
- int `emv_retrieveCAPK` (IN BYTE *capk, IN int capkLen, OUT BYTE *key, IN_OUT int *keyLen)

- int [emv_setCAPK](#) (IN BYTE *capk, IN int capkLen)
- int [emv_removeCAPK](#) (IN BYTE *capk, IN int capkLen)
- int [emv_removeAllCAPK](#) ()
- int [emv_retrieveCAPKList](#) (OUT BYTE *keys, IN_OUT int *keysLen)
- int [emv_retrieveTerminalID](#) (OUT char *terminalID)
- int [emv_retrieveTerminalID_Len](#) (OUT char *terminalID, IN_OUT int *terminalIDLen)
- int [emv_setTerminalID](#) (IN char *terminalID)
- int [emv_retrieveCRL](#) (OUT BYTE *list, IN_OUT int *lssLen)
- int [emv_setCRL](#) (IN BYTE *list, IN int lssLen)
- int [emv_removeCRL](#) (IN BYTE *list, IN int lssLen)
- int [emv_removeAllCRL](#) ()
- int [msr_clearMSRData](#) ()
- int [msr_getMSRData](#) (OUT BYTE *reData, IN_OUT int *reLen)
- int [msr_cancelMSRSwipe](#) ()
- int [msr_startMSRSwipe](#) (IN int _timeout)
- void [parseMSRData](#) (IN BYTE *resData, IN int resLen, IN_OUT IDTMSRData *cardData)
- int [pin_getPIN](#) (IN int mode, IN int PANSource, IN char *iccPAN, IN int IN iccPANLen, int startTimeout, IN int entryTimeout, IN char *language, IN int languageLen)
- int [pin_cancelPINEntry](#) ()
- void [parsePINBlockData](#) (IN BYTE *resData, IN int resLen, IN_OUT IDTPINData *cardData)
- void [parsePINData](#) (IN BYTE *resData, IN int resLen, IN_OUT IDTPINData *cardData)

11.9.1 Detailed Description

SpectrumPro API.

Spectrum Pro Global API methods.

11.9.2 Macro Definition Documentation

11.9.2.1 IN

```
#define IN
```

INPUT parameter.

11.9.2.2 IN_OUT

```
#define IN_OUT
```

INPUT / OUTPUT PARAMETER.

11.9.2.3 OUT

```
#define OUT
```

OUTPUT parameter.

11.9.3 Typedef Documentation

11.9.3.1 ftpComm_callBack

```
typedef void(* ftpComm_callBack) (int, int, int)
```

Define the comm callback function to get FTP file transfer status

It should be passed as a parameter in a FTP request, Signature (int, int, int) = response code, current block, total blocks
 RESPONSE CODES: 100 = FILE DOWNLOAD STARTED 101 = FILE BLOCK XX OF XX RECEIVED 102 = FILE DOWNLOAD COMPLETED 103 = FILE DOWNLOAD TERMINATED PREMATURELY

11.9.3.2 httpComm_callBack

```
typedef void(* httpComm_callBack) (BYTE *, int)
```

Define the comm callback function to get the async url data

It should be registered using the comm_registerHTTPCallback

11.9.3.3 pCMR_callBack

```
typedef void(* pCMR_callBack) (int, IDTCMRData *)
```

Define the camera callback function to get the image data

It should be registered using the device_registerCameraCallBk,

11.9.3.4 pCSFS_callBack

```
typedef void(* pCSFS_callBack) (BYTE status)
```

Define the card status and front switch callback function to get card and front switch status

It should be registered using the device_registerCardStatusFrontSwitchCallBk,

11.9.3.5 pEMV_callBack

```
typedef void(* pEMV_callBack) (int, int, BYTE *, int, IDTTransactionData *, EMV_Callback *, int)
```

Define the EMV callback function to get the transaction message/data/result.

It should be registered using the emv_registerCallBk,

11.9.3.6 pMessageHotplug

```
typedef void(* pMessageHotplug) (int, int)
```

Define the USB hot-plug callback function to monitor the info when plug in/out the reader.

It should be registered using the registerHotplugCallBk, The first integer parameter is device type, and the second integer parameter is either 0: Device Plugged Out or 1: Device Plugged In

11.9.3.7 pMSR_callBack

```
typedef void(* pMSR_callBack) (int, IDTMSRData)
```

Define the MSR callback function to get the MSR card data

It should be registered using the msr_registerCallBk, this callback function is for backward compatibility

11.9.3.8 pMSR_callBackp

```
typedef void(* pMSR_callBackp) (int, IDTMSRData *)
```


Define the MSR callback function to get pointer to the MSR card data

It should be registered using the `msr_registerCallBk`, this callback function is recommended instead of `pMSR_callBack`

11.9.3.9 pPIN_callBack

```
typedef void(* pPIN_callBack) (int, IDTPINData *)
```

Define the PINPad callback function to get the input PIN Pad data

It should be registered using the `pin_registerCallBk`,

11.9.3.10 pReadDataLog

```
typedef void(* pReadDataLog) (BYTE *, int)
```

Define the read response callback function to monitor the reading response from the reader.

It should be registered using the `registerLogCallBk`,

11.9.3.11 pSendDataLog

```
typedef void(* pSendDataLog) (BYTE *, int)
```

Define the send command callback function to monitor the sending command into the reader.

It should be registered using the `registerLogCallBk`,

11.9.3.12 v4Comm_callBack

```
typedef void(* v4Comm_callBack) (BYTE, BYTE, BYTE *, int)
```

Define the comm callback function to receive the V4 Protocol packets received by the device from an external source (IP/USB/RS-232) It should be registered using the `comm_registerV4Callback`, Data callback will contain command, sub-command, and data from V4 packet

11.9.4 Function Documentation

11.9.4.1 config_getModelNumber()

```
int config_getModelNumber (
    OUT char * sNumber )
```

DEPRECATED : please use `config_getModelNumber_Len(OUT char* sNumber, IN_OUT int *sNumberLen)`

Polls device for Model Number

Parameters

<code>sNumber</code>	Returns Model Number; needs to have at least 64 bytes of memory
----------------------	---

Returns

RETURN_CODE: Values can be parsed with `device_getResponseCodeString`

11.9.4.2 config_getModelNumber_Len()

```
int config_getModelNumber_Len (
    OUT char * sNumber,
    IN_OUT int * sNumberLen )
```

Polls device for Model Number

Parameters

<i>sNumber</i>	Returns Model Number
<i>sNumber</i>	length of Model Number

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString

11.9.4.3 config_getSerialNumber()

```
int config_getSerialNumber (
    OUT char * sNumber )
```

DEPRECATED : please use [config_getSerialNumber_Len\(OUT char* sNumber, IN_OUT int *sNumberLen\)](#)

Polls device for Serial Number

Parameters

<i>sNumber</i>	Returns Serial Number; needs to have at least 64 bytes of memory
----------------	--

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString

11.9.4.4 config_getSerialNumber_Len()

```
int config_getSerialNumber_Len (
    OUT char * sNumber,
    IN_OUT int * sNumberLen )
```

Polls device for Serial Number

Parameters

<i>sNumber</i>	Returns Serial Number
<i>sNumberLen</i>	Length of Serial Number

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString

11.9.4.5 device_close()

```
int device_close ( )
```

Close the device

Returns

RETURN_CODE: 0: success, 0x0A: failed

11.9.4.6 device_getCurrentDeviceType()

```
int device_getCurrentDeviceType ( )
```

Get current active device type

Returns

: return the device type defined as DEVICE_TYPE in the IDTDef.h

11.9.4.7 device_getFirmwareVersion()

```
int device_getFirmwareVersion (
    OUT char * firmwareVersion )
```

DEPRECATED : please use [device_getFirmwareVersion_Len](#)(OUT char* firmwareVersion, IN_OUT int *firmwareVersionLen)

Polls device for Firmware Version

Parameters

<i>firmwareVersion</i>	Response returned of Firmware Version; needs to have at least 128 bytes of memory
------------------------	---

Returns

RETURN_CODE: Values can be parsed with [device_getResponseCodeString](#)()

11.9.4.8 device_getFirmwareVersion_Len()

```
int device_getFirmwareVersion_Len (
    OUT char * firmwareVersion,
    IN_OUT int * firmwareVersionLen )
```

Polls device for Firmware Version

Parameters

<i>firmwareVersion</i>	Response returned of Firmware Version
<i>firmwareVersionLen</i>	Length of Firmware Version

Returns

RETURN_CODE: Values can be parsed with [device_getResponseCodeString\(\)](#)

11.9.4.9 device_getResponseCodeString()

```
void device_getResponseCodeString (
    IN int returnCode,
    OUT char * despcrition )
```

Review the return code description.

Parameters

<i>returnCode</i>	the response result.
<i>description</i>	

Return values

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

- 0: "no error, beginning task";
- 1: "no response from reader";
- 2: "invalid response data";
- 3: "time out for task or CMD";
- 4: "wrong parameter";
- 5: "SDK is doing MSR or ICC task";
- 6: "SDK is doing PINPad task";
- 7: "SDK is doing CTLS task";
- 8: "SDK is doing EMV task";
- 9: "SDK is doing Other task";
- 10: "err response or data";
- 11: "no reader attached";
- 12: "mono audio is enabled";
- 13: "did connection";
- 14: "audio volume is too low";
- 15: "task or CMD be canceled";
- 16: "UF wrong string format";
- 17: "UF file not found";
- 18: "UF wrong file format";

- 19: "Attempt to contact online host failed";
- 20: "Attempt to perform RKL failed";
- 22: "Buffer size is not enough";
- 0X300: "Key Type(TDES) of Session Key is not same as the related Master Key.";
- 0X400: "Related Key was not loaded.";
- 0X500: "Key Same.";
- 0X501: "Key is all zero";
- 0X502: "TR-31 format error";
- 0X702: "PAN is Error Key.";
- 0X705: "No Internal MSR PAN (or Internal MSR PAN is erased timeout)";
- 0X0C01: "Incorrect Frame Tag";
- 0X0C02: "Incorrect Frame Type";
- 0X0C03: "Unknown Frame Type";
- 0X0C04: "Unknown Command";
- 0X0C05: "Unknown Sub-Command";
- 0X0C06: "CRC Error";
- 0X0C07: "Failed";
- 0X0C08: "Timeout";
- 0X0C0A: "Incorrect Parameter";
- 0X0C0B: "Command Not Supported";
- 0X0C0C: "Sub-Command Not Supported";
- 0X0C0D: "Parameter Not Supported / Status Abort Command";
- 0X0C0F: "Sub-Command Not Allowed";
- 0X0D01: "Incorrect Header Tag";
- 0X0D02: "Unknown Command";
- 0X0D03: "Unknown Sub-Command";
- 0X0D04: "CRC Error in Frame";
- 0X0D05: "Incorrect Parameter";
- 0X0D06: "Parameter Not Supported";
- 0X0D07: "Mal-formatted Data";
- 0X0D08: "Timeout";
- 0X0D0A: "Failed / NACK";
- 0X0D0B: "Command not Allowed";
- 0X0D0C: "Sub-Command not Allowed";
- 0X0D0D: "Buffer Overflow (Data Length too large for reader buffer)";
- 0X0D0E: "User Interface Event";

- 0X0D11: "Communication type not supported, VT-1, burst, etc.";
- 0X0D12: "Secure interface is not functional or is in an intermediate state.";
- 0X0D13: "Data field is not mod 8";
- 0X0D14: "Pad - 0X80 not found where expected";
- 0X0D15: "Specified key type is invalid";
- 0X0D1: "Could not retrieve key from the SAM(InitSecureComm)";
- 0X0D17: "Hash code problem";
- 0X0D18: "Could not store the key into the SAM(InstallKey)";
- 0X0D19: "Frame is too large";
- 0X0D1A: "Unit powered up in authentication state but POS must resend the InitSecureComm command";
- 0X0D1B: "The EEPROM may not be initialized because SecCommInterface does not make sense";
- 0X0D1C: "Problem encoding APDU";
- 0X0D20: "Unsupported Index(ILM) SAM Transceiver error - problem communicating with the SAM(Key Mgr)";
- 0X0D2: "Unexpected Sequence Counter in multiple frames for single bitmap(ILM) Length error in data returned from the SAM(Key Mgr)";
- 0X0D22: "Improper bit map(ILM)";
- 0X0D23: "Request Online Authorization";
- 0X0D24: "ViVOCard3 raw data read successful";
- 0X0D25: "Message index not available(ILM) ViVOcomm activate transaction card type(ViVOcomm)";
- 0X0D26: "Version Information Mismatch(ILM)";
- 0X0D27: "Not sending commands in correct index message index(ILM)";
- 0X0D28: "Time out or next expected message not received(ILM)";
- 0X0D29: "ILM languages not available for viewing(ILM)";
- 0X0D2A: "Other language not supported(ILM)";
- 0X0D41: "Unknown Error from SAM";
- 0X0D42: "Invalid data detected by SAM";
- 0X0D43: "Incomplete data detected by SAM";
- 0X0D44: "Reserved";
- 0X0D45: "Invalid key hash algorithm";
- 0X0D46: "Invalid key encryption algorithm";
- 0X0D47: "Invalid modulus length";
- 0X0D48: "Invalid exponent";
- 0X0D49: "Key already exists";
- 0X0D4A: "No space for new RID";
- 0X0D4B: "Key not found";
- 0X0D4C: "Crypto not responding";

- 0X0D4D: "Crypto communication error";
- 0X0D4E: "Module-specific error for Key Manager";
- 0X0D4F: "All key slots are full (maximum number of keys has been installed)";
- 0X0D50: "Auto-Switch OK";
- 0X0D51: "Auto-Switch failed";
- 0X0D90: "Account DUKPT Key not exist";
- 0X0D91: "Account DUKPT Key KSN exhausted";
- 0X0D00: "This Key had been loaded.";
- 0X0E00: "Base Time was loaded.";
- 0X0F00: "Encryption Or Decryption Failed.";
- 0X1000: "Battery Low Warning (It is High Priority Response while Battery is Low.)";
- 0X1800: "Send 'Cancel Command' after send 'Get Encrypted PIN' & 'Get Numeric' & 'Get Amount'; - 0↔
- X1900: "Press 'Cancel' key after send 'Get Encrypted PIN' & 'Get Numeric' & 'Get Amount';
- 0X30FF: "Security Chip is not connect";
- 0X3000: "Security Chip is deactivation & Device is In Removal Legally State.";
- 0X3101: "Security Chip is activation & Device is In Removal Legally State.";
- 0X5500: "No Admin DUKPT Key.";
- 0X5501: "Admin DUKPT Key STOP.";
- 0X5502: "Admin DUKPT Key KSN is Error.";
- 0X5503: "Get Authentication Code1 Failed.";
- 0X5504: "Validate Authentication Code Error.";
- 0X5505: "Encrypt or Decrypt data failed.";
- 0X5506: "Not Support the New Key Type.";
- 0X5507: "New Key Index is Error.";
- 0X5508: "Step Error.";
- 0X5509: "KSN Error";
- 0X550A: "MAC Error.";
- 0X550B: "Key Usage Error.";
- 0X550C: "Mode Of Use Error.";
- 0X550F: "Other Error.";
- 0X6000: "Save or Config Failed / Or Read Config Error.";
- 0X6200: "No Serial Number.";
- 0X6900: "Invalid Command - Protocol is right, but task ID is invalid.";
- 0X6A01: "Unsupported Command - Protocol and task ID are right, but command is invalid - In this State";
- 0X6A00: "Unsupported Command - Protocol and task ID are right, but command is invalid.";
- 0X6B00: "Unknown parameter in command - Protocol task ID and command are right, but parameter is invalid.";

- 0X6C00: "Unknown parameter in command - Protocol task ID and command are right, but length is out of the requirement.";
- 0X7200: "Device is suspend (MKSK suspend or press password suspend).";
- 0X7300: "PIN DUKPT is STOP (21 bit 1).";
- 0X7400: "Device is Busy.";
- 0XE100: "Can not enter sleep mode";
- 0XE200: "File has existed";
- 0XE300: "File has not existed";
- 0XE313: "IO line low -- Card error after session start";
- 0XE400: "Open File Error";
- 0XE500: "SmartCard Error";
- 0XE600: "Get MSR Card data is error";
- 0XE700: "Command time out";
- 0XE800: "File read or write is error";
- 0XE900: "Active 1850 error!";
- 0XEA00: "Load bootloader error";
- 0XEF00: "Protocol Error- STX or ETX or check error.";
- 0XEB00: "Picture is not exist";
- 0X2C02: "No Microprocessor ICC seated";
- 0X2C06: "no card seated to request ATR";
- 0X2D01: "Card Not Supported,";
- 0X2D03: "Card Not Supported, wants CRC";
- 0X690D: "Command not supported on reader without ICC support";
- 0X8100: "ICC error time out on power-up";
- 0X8200: "invalid TS character received - Wrong operation step";
- 0X8300: "Decode MSR Error";
- 0X8400: "TriMagII no Response";
- 0X8500: "No Swipe MSR Card";
- 0X8510: "No Financial Card";
- 0X8600: "Unsupported F, D, or combination of F and D";
- 0X8700: "protocol not supported EMV TD1 out of range";
- 0X8800: "power not at proper level";
- 0X8900: "ATR length too long";
- 0X8B01: "EMV invalid TA1 byte value";
- 0X8B02: "EMV TB1 required";
- 0X8B03: "EMV Unsupported TB1 only 00 allowed";

- 0X8B04: "EMV Card Error, invalid BWI or CWI";
- 0X8B06: "EMV TB2 not allowed in ATR";
- 0X8B07: "EMV TC2 out of range";
- 0X8B08: "EMV TC2 out of range";
- 0X8B09: "per EMV96 TA3 must be > - 0XF";
- 0X8B10: "ICC error on power-up";
- 0X8B11: "EMV T=1 then TB3 required";
- 0X8B12: "Card Error, invalid BWI or CWI";
- 0X8B13: "Card Error, invalid BWI or CWI";
- 0X8B17: "EMV TC1/TB3 conflict-";
- 0X8B20: "EMV TD2 out of range must be T=1";
- 0X8C00: "TCK error";
- 0XA304: "connector has no voltage setting";
- 0XA305: "ICC error on power-up invalid (SBLK(IFSD) exchange";
- 0XE301: "ICC error after session start";
- 0XFF00: "Request to go online";
- 0XFF01: "EMV: Accept the offline transaction";
- 0XFF02: "EMV: Decline the offline transaction";
- 0XFF03: "EMV: Accept the online transaction";
- 0XFF04: "EMV: Decline the online transaction";
- 0XFF05: "EMV: Application may fallback to magstripe technology";
- 0XFF06: "EMV: ICC detected tah the conditions of use are not satisfied";
- 0XFF07: "EMV: ICC didn't accept transaction";
- 0XFF08: "EMV: Transaction was cancelled";
- 0XFF09: "EMV: Application was not selected by kernel or ICC format error or ICC missing data error";
- 0XFF0A: "EMV: Transaction is terminated";
- 0XFF0B: "EMV: Other EMV Error";
- 0XFFFF: "NO RESPONSE";
- 0XF002: "ICC communication timeout";
- 0XF003: "ICC communication Error";
- 0XF00F: "ICC Card Seated and Highest Priority, disable MSR work request";
- 0XF200: "AID List / Application Data is not exist";
- 0XF201: "Terminal Data is not exist";
- 0XF202: "TLV format is error";
- 0XF203: "AID List is full";
- 0XF204: "Any CA Key is not exist";

- 0XF205: "CA Key RID is not exist";
- 0XF206: "CA Key Index it not exist";
- 0XF207: "CA Key is full";
- 0XF208: "CA Key Hash Value is Error";
- 0XF209: "Transaction format error";
- 0XF20A: "The command will not be processing";
- 0XF20B: "CRL is not exist";
- 0XF20C: "CRL number exceed max number";
- 0XF20D: "Amount,Other Amount,Trasaction Type are missing";
- 0XF20E: "The Identification of algorithm is mistake";
- 0XF20F: "No Financial Card";
- 0XF210: "In Encrypt Result state, TLV total Length is greater than Max Length";
- 0X1001: "INVALID ARG";
- 0X1002: "FILE_OPEN_FAILED";
- 0X1003: "FILE OPERATION_FAILED";
- 0X2001: "MEMORY_NOT_ENOUGH";
- 0X3002: "SMARTCARD_FAIL";
- 0X3003: "SMARTCARD_INIT_FAILED";
- 0X3004: "FALLBACK_SITUATION";
- 0X3005: "SMARTCARD_ABSENT";
- 0X3006: "SMARTCARD_TIMEOUT";
- 0X3012: "EMV_RESULT_CODE_MSR_CARD_ERROR_FALLBACK";
- 0X5001: "EMV_PARSING_TAGS_FAILED";
- 0X5002: "EMV_DUPLICATE_CARD_DATA_ELEMENT";
- 0X5003: "EMV_DATA_FORMAT_INCORRECT";
- 0X5004: "EMV_NO_TERM_APP";
- 0X5005: "EMV_NO_MATCHING_APP";
- 0X5006: "EMV_MISSING_MANDATORY_OBJECT";
- 0X5007: "EMV_APP_SELECTION_RETRY";
- 0X5008: "EMV_GET_AMOUNT_ERROR";
- 0X5009: "EMV_CARD_REJECTED";
- 0X5010: "EMV_AIP_NOT_RECEIVED";
- 0X5011: "EMV_AFL_NOT_RECEIVED";
- 0X5012: "EMV_AFL_LEN_OUT_OF_RANGE";
- 0X5013: "EMV_SFI_OUT_OF_RANGE";
- 0X5014: "EMV_AFL_INCORRECT";

- 0X5015: "EMV_EXP_DATE_INCORRECT";
- 0X5016: "EMV_EFF_DATE_INCORRECT";
- 0X5017: "EMV_ISS_COD_TBL_OUT_OF_RANGE";
- 0X5018: "EMV_CRYPTOGRAM_TYPE_INCORRECT";
- 0X5019: "EMV_PSE_NOT_SUPPORTED_BY_CARD";
- 0X5020: "EMV_USER_SELECTED_LANGUAGE";
- 0X5021: "EMV_SERVICE_NOT_ALLOWED";
- 0X5022: "EMV_NO_TAG_FOUND";
- 0X5023: "EMV_CARD_BLOCKED";
- 0X5024: "EMV_LEN_INCORRECT";
- 0X5025: "CARD_COM_ERROR";
- 0X5026: "EMV_TSC_NOT_INCREASED";
- 0X5027: "EMV_HASH_INCORRECT";
- 0X5028: "EMV_NO_ARC";
- 0X5029: "EMV_INVALID_ARC";
- 0X5030: "EMV_NO_ONLINE_COMM";
- 0X5031: "TRAN_TYPE_INCORRECT";
- 0X5032: "EMV_APP_NO_SUPPORT";
- 0X5033: "EMV_APP_NOT_SELECT";
- 0X5034: "EMV_LANG_NOT_SELECT";
- 0X5035: "EMV_NO_TERM_DATA";
- 0X5039: "EMV_PIN_ENTRY_TIMEOUT";
- 0X6001: "CVM_TYPE_UNKNOWN";
- 0X6002: "CVM_AIP_NOT_SUPPORTED";
- 0X6003: "CVM_TAG_8E_MISSING";
- 0X6004: "CVM_TAG_8E_FORMAT_ERROR";
- 0X6005: "CVM_CODE_IS_NOT_SUPPORTED";
- 0X6006: "CVM_COND_CODE_IS_NOT_SUPPORTED";
- 0X6007: "NO_MORE_CVM";
- 0X6008: "PIN_BYPASSED_BEFORE";
- 0X7001: "PK_BUFFER_SIZE_TOO_BIG";
- 0X7002: "PK_FILE_WRITE_ERROR";
- 0X7003: "PK_HASH_ERROR";
- 0X8001: "NO_CARD_HOLDER_CONFIRMATION";
- 0X8002: "GET_ONLINE_PIN";
- 0XD000: "Data not exist";

- 0XD001: "Data access error";
- 0XD100: "RID not exist";
- 0XD101: "RID existed";
- 0XD102: "Index not exist";
- 0XD200: "Maximum exceeded";
- 0XD201: "Hash error";
- 0XD205: "System Busy";
- 0X0E01: "Unable to go online";
- 0X0E02: "Technical Issue";
- 0X0E03: "Declined";
- 0X0E04: "Issuer Referral transaction";
- 0X0F01: "Decline the online transaction";
- 0X0F02: "Request to go online";
- 0X0F03: "Transaction is terminated";
- 0X0F05: "Application was not selected by kernel or ICC format error or ICC missing data error";
- 0X0F07: "ICC didn't accept transaction";
- 0X0F0A: "Application may fallback to magstripe technology";
- 0X0F0C: "Transaction was cancelled";
- 0X0F0D: "Timeout";
- 0X0F0F: "Other EMV Error";
- 0X0F10: "Accept the offline transaction";
- 0X0F11: "Decline the offline transaction";
- 0X0F21: "ICC detected that the conditions of use are not satisfied";
- 0X0F22: "No app were found on card matching terminal configuration";
- 0X0F23: "Terminal file does not exist";
- 0X0F24: "CAPK file does not exist";
- 0X0F25: "CRL Entry does not exist";
- 0X0FFE: "code when blocking is disabled";
- 0X0FFF: "code when command is not applicable on the selected device";
- 0XF005: "ICC Encrypted C-APDU Data Structure Length Error Or Format Error.";
- 0XBBE0: "CM100 Success";
- 0XBBE1: "CM100 Parameter Error";
- 0XBBE2: "CM100 Low Output Buffer";
- 0XBBE3: "CM100 Card Not Found";
- 0XBBE4: "CM100 Collision Card Exists";
- 0XBBE5: "CM100 Too Many Cards Exist";

- 0XBBE6: "CM100 Saved Data Does Not Exist";
- 0XBBE8: "CM100 No Data Available";
- 0XBBE9: "CM100 Invalid CID Returned";
- 0XBBEA: "CM100 Invalid Card Exists";
- 0BBEC: "CM100 Command Unsupported";
- 0BBED: "CM100 Error In Command Process";
- 0BBEE: "CM100 Invalid Command";
- 0X9031: "Unknown command";
- 0X9032: "Wrong parameter (such as the length of the command is incorrect)";
- 0X9038: "Wait (the command couldnt be finished in BWT)";
- 0X9039: "Busy (a previously command has not been finished)";
- 0X903A: "Number of retries over limit";
- 0X9040: "Invalid Manufacturing system data";
- 0X9041: "Not authenticated";
- 0X9042: "Invalid Master DUKPT Key";
- 0X9043: "Invalid MAC Key";
- 0X9044: "Reserved for future use";
- 0X9045: "Reserved for future use";
- 0X9046: "Invalid DATA DUKPT Key";
- 0X9047: "Invalid PIN Pairing DUKPT Key";
- 0X9048: "Invalid DATA Pairing DUKPT Key";
- 0X9049: "No nonce generated";
- 0X9949: "No GUID available. Perform getVersion first.";
- 0X9950: "MAC Calculation unsuccessful. Check BDK value.";
- 0X904A: "Not ready";
- 0X904B: "Not MAC data";
- 0X9050: "Invalid Certificate";
- 0X9051: "Duplicate key detected";
- 0X9052: "AT checks failed";
- 0X9053: "TR34 checks failed";
- 0X9054: "TR31 checks failed";
- 0X9055: "MAC checks failed";
- 0X9056: "Firmware download failed";
- 0X9060: "Log is full";
- 0X9061: "Removal sensor unengaged";
- 0X9062: "Any hardware problems";

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

11.9.4.10 device_getSDKWaitTime()

```
int device_getSDKWaitTime ( )
```

Get SDK Wait Time

Get the SDK wait time for transactions

Returns

SDK wait time in seconds

11.9.4.11 device_getSpectrumProKSN()

```
int device_getSpectrumProKSN (
    IN int type,
    OUT BYTE * KSN )
```

DEPRECATED : please use device_getSpectrumProKSN_Len(IN int type, OUT BYTE * KSN, IN_OUT int *KSN_Len)

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) • 2: Data Encryption Key (DEK) • 5: MAC Key (MAK) • 10: RKL Key Encryption Key (REK) • 20: HSM DUKPT Key
<i>KSN</i>	Key Serial Number; needs to have at least 10 bytes of memory

Returns

RETURN_CODE: Values can be parsed with [device_getResponseCodeString\(\)](#)

11.9.4.12 device_getSpectrumProKSN_Len()

```
int device_getSpectrumProKSN_Len (
    IN int type,
```

```
OUT BYTE * KSN,  
IN_OUT int * KSNLen )
```

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)• 2: Data Encryption Key (DEK)• 5: MAC Key (MAK)• 10: RKL Key Encryption Key (REK)• 20: HSM DUKPT Key
<i>KSN</i>	Key Serial Number
<i>KSNLen</i>	Length of KSN

Returns

RETURN_CODE: Values can be parsed with [device_getResponseCodeString\(\)](#)

11.9.4.13 device_getThreadStackSize()

```
int device_getThreadStackSize ( )
```

Get Thread Stack Size

Get the stack size setting for newly created threads

Returns

Thread Stack Size

11.9.4.14 device_init()

```
int device_init ( )
```

Initial the device by USB

It will detect the device and trying connect.

The connect status can be checked by [device_isConnected\(\)](#).

Note: after the function returns success, the function [device_setCurrentDevice\(\)](#) has to be called to set the device type.

Returns

RETURN_CODE: Values can be parsed with [device_getResponseCodeString\(\)](#)

11.9.4.15 device_isAttached()

```
int device_isAttached (
    int deviceType )
```

Check if the device is attached to the USB port The function [device_init\(\)](#) must be called before this function.

Parameters

<i>deviceType, the</i>	device type of the USB device
------------------------	-------------------------------

Returns

1 if the device is attached, or 0 if the device is not attached

11.9.4.16 device_isConnected()

```
int device_isConnected ( )
```

Check the device connected status

Returns

DEVICE_DISCONNECT=0, or DEVICE_CONNECTED = 1

11.9.4.17 device_pollCardReader()

```
int device_pollCardReader (
    OUT BYTE * status )
```

DEPRECATED : please use [device_pollCardReader_Len\(OUT BYTE * status, IN_OUT int *statusLen\)](#)

Poll Card Reader

Provides information about the state of the Card Reader

Parameters

<i>status</i>	<p>Six bytes indicating card reader information Byte 0:</p> <ul style="list-style-type: none"> • 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) • Bit 4: RFU • Bit 5: RFU • Bit 6: DATA_DUKPT Key Valid • Bit 7: Key is initialized (MFK and RSA Key pairs)
---------------	--

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)
- 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)

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

Returns

RETURN_CODE: Values can be parsed with [device_getResponseCodeString\(\)](#)

11.9.4.18 device_pollCardReader_Len()

```
int device_pollCardReader_Len (
    OUT BYTE * status,
    IN_OUT int * statusLen )
```

Poll Card Reader

Provides information about the state of the Card Reader

Parameters

<i>status</i>	<p>Six bytes indicating card reader information Byte 0:</p> <ul style="list-style-type: none"> • 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) • Bit 4: RFU • Bit 5: RFU • Bit 6: DATA_DUKPT Key Valid • Bit 7: Key is initialized (MFK and RSA Key pairs)
---------------	--

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)
- 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)

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>statusLen</i>	Length of status
------------------	------------------

Returns

RETURN_CODE: Values can be parsed with [device_getResponseCodeString\(\)](#)

11.9.4.19 device_rebootDevice()

```
int 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\(\)](#)

11.9.4.20 device_registerCameraCallBk()

```
void device_registerCameraCallBk (
    pCMR_callback pCMRf )
```

To register the camera callback function to get the image data. (Pass NULL to disable the callback.)

11.9.4.21 device_registerCardStatusFrontSwitchCallBk()

```
void device_registerCardStatusFrontSwitchCallBk (
    pCSFS_callback pCSFSf )
```

To register the card status and front switch callback function to get status. (Pass NULL to disable the callback.)

11.9.4.22 device_SendDataCommand()

```
int device_SendDataCommand (
    IN BYTE * cmd,
    IN int cmdLen,
    IN BYTE * data,
    IN int dataLen,
    OUT BYTE * response,
    IN_OUT int * respLen )
```

Send a Command to device

Sends a command to the device .

Parameters

<i>cmd</i>	buffer of command to execute.
<i>cmdLen,the</i>	length of the buffer cmd.
<i>data</i>	buffer of IDG command data.
<i>dataLen,the</i>	length of the buffer data.
<i>response</i>	Response data
<i>respLen,the</i>	length of Response data

Returns

RETURN_CODE: Values can be parsed with [device_getResponseCodeString\(\)](#)

11.9.4.23 device_setCurrentDevice()

```
int device_setCurrentDevice (
    int deviceType )
```

Sets the current device to talk to

The connect status can be checked by [device_isConnected\(\)](#).

Parameters

<i>deviceType</i>	Device to connect to
	<pre>enum DEVICE_TYPE { IDT_DEVICE_UNKNOWN=0, IDT_DEVICE_AUGUSTA_HID, IDT_DEVICE_AUGUSTA_KB, IDT_DEVICE_AUGUSTA_S_HID, IDT_DEVICE_AUGUSTA_S_KB, IDT_DEVICE_AUGUSTA_S_TTK_HID, IDT_DEVICE_SPECTRUM_PRO, IDT_DEVICE_MINISMART_II, IDT_DEVICE_L80, IDT_DEVICE_L100, IDT_DEVICE_UNIPAY, IDT_DEVICE_UNIPAY_I_V, IDT_DEVICE_VP3300_AJ, IDT_DEVICE_KIOSK_III, IDT_DEVICE_KIOSK_III_S, IDT_DEVICE_PIP_READER, IDT_DEVICE_VENDI, IDT_DEVICE_VP3300_USB, IDT_DEVICE_UNIPAY_I_V_TTK, IDT_DEVICE_VP3300_BT, IDT_DEVICE_VP8800, IDT_DEVICE_SREDKEY2_HID, IDT_DEVICE_SREDKEY2_KB, IDT_DEVICE_NEO2, IDT_DEVICE_MINISMART_II_COM = IDT_DEVICE_NEO2+5, IDT_DEVICE_SPECTRUM_PRO_COM, IDT_DEVICE_KIOSK_III_COM, IDT_DEVICE_KIOSK_III_S_COM, IDT_DEVICE_PIP_READER_COM, IDT_DEVICE_VP3300_COM, IDT_DEVICE_NEO2_COM, IDT_DEVICE_MAX_DEVICES = IDT_DEVICE_NEO2_COM+5 };</pre>

Returns

RETURN_CODE: 1: success, 0: failed

11.9.4.24 device_setSDKWaitTime()

```
void device_setSDKWaitTime (
    int waitTime )
```

Set SDK Wait Time

Set the SDK wait time for transactions

Parameters

<i>waitTime</i>	The SDK wait time for transaction in seconds. The Maximum is 2147483 seconds.
-----------------	---

11.9.4.25 device_setThreadStackSize()

```
void device_setThreadStackSize (
    int threadSize )
```

Set Thread Stack Size

Set the stack size setting for newly created threads

11.9.4.26 device_updateFirmware()

```
int device_updateFirmware (
    IN BYTE * firmwareData,
    IN int firmwareDataLen,
    IN char * firmwareName,
    IN int encryptionType,
    IN BYTE * keyBlob,
    IN int keyBlobLen )
```

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>firmwareDataLen</i>	Length of firmwareData
<i>firmwareName</i>	Firmware name. Must be one of the following two strings (with appropriate version information) <ul style="list-style-type: none"> • "SP K21 APP Vx.xx.xxx" • "SP MAX APP Vx.xx.xxx"
<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)
<i>keyBlobLen</i>	Length of keyBlob

Returns

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

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

11.9.4.27 emv_activateTransaction()

```
int emv_activateTransaction (
    IN int timeout,
    IN BYTE * tags,
    IN int tagsLen,
    IN int forceOnline )
```

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>	Tags to be included in the request. Passed as a TLV stream. Example, tag 9F0C with amount 0x000000000100 would be 0x9F0C06000000000100
<i>tagsLen</i>	Length of tags
<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

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString >>>>IMPORTANT: parameters for amount and amtOther MUST BE PASSED AS A DOUBLE VALUE WITH DECIMAL POINT. Example, do not pass 1, but instead pass 1.0 or 1.00. Otherwise, results will be unpredictable

11.9.4.28 emv_allowFallback()

```
void emv_allowFallback (
    IN int allow )
```

Allow fallback for EMV transactions. Default is TRUE

Parameters

<i>allow</i>	TRUE = allow fallback, FALSE = don't allow fallback
--------------	---

11.9.4.29 emv_authenticateTransaction()

```
int emv_authenticateTransaction (
    IN BYTE * updatedTLV,
    IN int updatedTLVLen )
```

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 = DFEE1A079F029F36959F37
<i>updatedTLVLen</i>	

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString

11.9.4.30 emv_authenticateTransactionWithTimeout()

```
int emv_authenticateTransactionWithTimeout (
    IN int timeout,
    IN BYTE * updatedTLV,
    IN int updatedTLVLen )
```

Authenticate EMV Transaction Request with Timeout

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>timeout</i>	Timeout value in seconds.
<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 = DFEE1A079F029F36959F37
<i>updatedTLVLen</i>	

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString

11.9.4.31 `emv_callbackResponseLCD()`

```
int emv_callbackResponseLCD (
    IN int type,
    byte selection )
```

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

Returns

RETURN_CODE: Values can be parsed with [device_getResponseCodeString\(\)](#)

11.9.4.32 `emv_callbackResponseMSR()`

```
int emv_callbackResponseMSR (
    IN BYTE * MSR,
    IN_OUT int MSRLen )
```

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>MSRLen</i>	the length of Swiped track data

Returns

RETURN_CODE: Values can be parsed with [device_getResponseCodeString\(\)](#)

11.9.4.33 `emv_cancelTransaction()`

```
int emv_cancelTransaction ( )
```

Cancel EMV Transaction

Cancels the currently executing EMV transaction.

Returns

RETURN_CODE: Values can be parsed with [device_getResponseCodeString\(\)](#)

11.9.4.34 emv_completeTransaction()

```
int emv_completeTransaction (
    IN int commError,
    IN BYTE * authCode,
    IN int authCodeLen,
    IN BYTE * iad,
    IN int iadLen,
    IN BYTE * tlvScripts,
    IN int tlvScriptsLen,
    IN BYTE * tlv,
    IN int tlvLen )
```

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(1) if host was unreachable, or FALSE(0) 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>authCodeLen</i>	the length of authCode
<i>iad</i>	Issuer Authentication Data, if any. Example 0x11223344556677883030 (tag value 91).
<i>iadLen</i>	the length of iadLen
<i>tlvScripts</i>	71/72 scripts, if any
<i>tlvScriptsLen</i>	the length of tlvScriptsLen
<i>tlv</i>	Additional TLV data to return with transaction results (if any)
<i>tlvLen</i>	the length of tlv

Returns

RETURN_CODE: Values can be parsed with [device_getResponseCodeString](#)

11.9.4.35 emv_getAutoAuthenticateTransaction()

```
int emv_getAutoAuthenticateTransaction ( )
```

Gets auto authenticate value for EMV transactions.

Returns

RETURN_CODE: TRUE = auto authenticate, FALSE = manually authenticate

11.9.4.36 emv_getAutoCompleteTransaction()

```
int emv_getAutoCompleteTransaction ( )
```

Gets auto complete value for EMV transactions.

Returns

RETURN_CODE: TRUE = auto complete, FALSE = manually complete

11.9.4.37 emv_getEMVConfigurationCheckValue()

```
int emv_getEMVConfigurationCheckValue (
    OUT BYTE * checkValue,
    IN_OUT int * checkValueLen )
```

Get EMV Kernel configuration check value info

Parameters

<i>checkValue</i>	Response returned of Kernel configuration check value info
<i>checkValueLen</i>	the length of checkValue

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString

11.9.4.38 emv_getEMVKernelCheckValue()

```
int emv_getEMVKernelCheckValue (
    OUT BYTE * checkValue,
    IN_OUT int * checkValueLen )
```

Get EMV Kernel check value info

Parameters

<i>checkValue</i>	Response returned of Kernel check value info
<i>checkValueLen</i>	the length of checkValue

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString

11.9.4.39 emv_getEMVKernelVersion()

```
int emv_getEMVKernelVersion (
    OUT char * version )
```

DEPRECATED : please use [emv_getEMVKernelVersion_Len\(OUT char* version, IN_OUT int *versionLen\)](#)

Polls device for EMV Kernel Version

Parameters

<i>version</i>	Response returned of Kernel Version; needs to have at least 128 bytes of memory.
----------------	--

Returns

RETURN_CODE: Values can be parsed with [device_getResponseCodeString\(\)](#)

11.9.4.40 emv_getEMVKernelVersion_Len()

```
int emv_getEMVKernelVersion_Len (
    OUT char * version,
    IN_OUT int * versionLen )
```

Polls device for EMV Kernel Version

Parameters

<i>version</i>	Response returned of Kernel Version
<i>versionLen</i>	Length of version

Returns

RETURN_CODE: Values can be parsed with [device_getResponseCodeString\(\)](#)

11.9.4.41 emv_registerCallBk()

```
void emv_registerCallBk (
    pEMV_callback pEMVf )
```

To register the emv callback function to get the EMV processing response. (Pass NULL to disable the callback.)

11.9.4.42 emv_removeAllApplicationData()

```
int emv_removeAllApplicationData ( )
```

Remove All Application Data

Removes all the Application Data

Returns

RETURN_CODE: Values can be parsed with [device_getResponseCodeString\(\)](#)

11.9.4.43 emv_removeAllCAPK()

```
int emv_removeAllCAPK ( )
```

Remove All Certificate Authority Public Key

Removes all the CAPK

Returns

RETURN_CODE: Values can be parsed with [device_getResponseCodeString\(\)](#)

11.9.4.44 emv_removeAllCRL()

```
int emv_removeAllCRL ( )
```

Remove All Certificate Revocation List Entries

Removes all CRLentry entries

Returns

RETURN_CODE: Values can be parsed with [device_getResponseCodeString\(\)](#)

11.9.4.45 emv_removeApplicationData()

```
int emv_removeApplicationData (
    IN BYTE * AID,
    IN int AIDLen )
```

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>AIDLen</i>	the length of AID data buffer

Returns

RETURN_CODE: Values can be parsed with [device_getResponseCodeString\(\)](#)

11.9.4.46 emv_removeCAPK()

```
int emv_removeCAPK (
    IN BYTE * capk,
    IN int capkLen )
```

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>capkLen</i>	the length of capk data buffer

Returns

RETURN_CODE: Values can be parsed with [device_getResponseCodeString\(\)](#)

11.9.4.47 emv_removeCRL()

```
int emv_removeCRL (
    IN BYTE * list,
    IN int lsLen )
```

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>lsLen</i>	the length of list data buffer

Returns

RETURN_CODE: Values can be parsed with [device_getResponseCodeString\(\)](#)

11.9.4.48 emv_removeTerminalData()

```
int emv_removeTerminalData ( )
```

Remove Terminal Data

Removes the Terminal Data

Returns

RETURN_CODE: Values can be parsed with [device_getResponseCodeString\(\)](#)

11.9.4.49 emv_retrieveAIDList()

```
int emv_retrieveAIDList (
    OUT BYTE * AIDList,
    IN_OUT int * AIDListLen )
```

Retrieve AID list

Returns all the AID names installed on the terminal.

Parameters

<i>AIDList</i>	array of AID name byte arrays
<i>AIDListLen</i>	the length of AIDList array buffer

Returns

RETURN_CODE: Values can be parsed with [device_getResponseCodeString\(\)](#)

11.9.4.50 emv_retrieveApplicationData()

```
int emv_retrieveApplicationData (
    IN BYTE * AID,
    IN int AIDLen,
    OUT BYTE * tlv,
    IN_OUT int * tlvLen )
```

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>AIDLen</i>	the length of AID data buffer.
<i>tlv</i>	The TLV elements of the requested AID
<i>tlvLen</i>	the length of tlv data buffer.

Returns

RETURN_CODE: Values can be parsed with [device_getResponseCodeString\(\)](#)

11.9.4.51 emv_retrieveCAPK()

```
int emv_retrieveCAPK (
    IN BYTE * capk,
    IN int capkLen,
    OUT BYTE * key,
    IN_OUT int * keyLen )
```

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>capkLen</i>	the length of capk data buffer

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>keyLen</i>	<p>the length of key data buffer</p> <ul style="list-style-type: none"> •

Returns

RETURN_CODE: Values can be parsed with [device_getResponseCodeString\(\)](#)

11.9.4.52 emv_retrieveCAPKList()

```
int emv_retrieveCAPKList (
    OUT BYTE * keys,
    IN_OUT int * keyLen )
```

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>keyLen</i>	the length of keys data buffer

Returns

RETURN_CODE: Values can be parsed with [device_getResponseCodeString\(\)](#)

11.9.4.53 emv_retrieveCRL()

```
int emv_retrieveCRL (
    OUT BYTE * list,
    IN_OUT int * listLen )
```

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>lssLen</i>	the length of list data buffer

Returns

RETURN_CODE: Values can be parsed with [device_getResponseCodeString\(\)](#)

11.9.4.54 emv_retrieveTerminalData()

```
int emv_retrieveTerminalData (
    OUT BYTE * tlv,
    IN_OUT int * tlvLen )
```

Retrieve Terminal Data

Retrieves the Terminal Data.

Parameters

<i>tlv</i>	Response returned as a TLV
<i>tlvLen</i>	the length of tlv data buffer

Returns

RETURN_CODE: Values can be parsed with [device_getResponseCodeString\(\)](#)

11.9.4.55 emv_retrieveTerminalID()

```
int emv_retrieveTerminalID (
    OUT char * terminalID )
```

DEPRECATED : please use [emv_retrieveTerminalID_Len\(OUT char* terminalID, IN_OUT int *terminalIDLen\)](#)

Gets the terminal ID as printable characters .

Parameters

<i>terminalID</i>	Terminal ID string; needs to have at least 30 bytes of memory
-------------------	---

Returns

RETURN_CODE: Values can be parsed with [device_getResponseCodeString\(\)](#)

11.9.4.56 `emv_retrieveTerminalID_Len()`

```
int emv_retrieveTerminalID_Len (
    OUT char * terminalID,
    IN_OUT int * terminalIDLen )
```

Gets the terminal ID as printable characters .

Parameters

<i>terminalID</i>	Terminal ID string
<i>terminalIDLen</i>	Length of terminalID

Returns

RETURN_CODE: Values can be parsed with [device_getResponseCodeString\(\)](#)

11.9.4.57 `emv_retrieveTransactionResult()`

```
int emv_retrieveTransactionResult (
    IN BYTE * tags,
    IN int tagsLen,
    IDTTransactionData * cardData )
```

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>tagsLen</i>	Length of tag list
<i>cardData</i>	All requested tags returned as unencrypted, encrypted and masked TLV data in IDTTransactionData object

Returns

RETURN_CODE: Values can be parsed with [device_getResponseCodeString\(\)](#)

11.9.4.58 `emv_setApplicationData()`

```
int emv_setApplicationData (
    IN BYTE * name,
    IN int nameLen,
    IN BYTE * tlv,
    IN int tlvLen )
```

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>nameLen</i>	the length of name data buffer of Application name,
<i>tlv</i>	Application data in TLV format
<i>tlvLen</i>	the length of tlv data buffer

Returns

RETURN_CODE: Values can be parsed with [device_getResponseCodeString\(\)](#)

11.9.4.59 `emv_setAutoAuthenticateTransaction()`

```
void emv_setAutoAuthenticateTransaction (
    IN int authenticate )
```

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

Parameters

<i>authenticate</i>	TRUE = auto authenticate, FALSE = manually authenticate
---------------------	---

11.9.4.60 `emv_setAutoCompleteTransaction()`

```
void emv_setAutoCompleteTransaction (
    IN int complete )
```

Enables complete for EMV transactions. If a `emv_authenticateTransaction` results in code 0x0004 (go online), then `emv_completeTransaction` can automatically execute if parameter is set to TRUE

Parameters

<i>complete</i>	TRUE = auto complete, FALSE = manually complete
-----------------	---

11.9.4.61 `emv_setCAPK()`

```
int emv_setCAPK (
    IN BYTE * capk,
    IN int capkLen )
```

Set Certificate Authority Public Key

Sets the CAPK as specified by the CAKey structure

Parameters

<i>capk</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>capkLen</i>	the length of capk data buffer

Returns

RETURN_CODE: Values can be parsed with [device_getResponseCodeString\(\)](#)

11.9.4.62 **emv_setCRL()**

```
int emv_setCRL (
    IN BYTE * list,
    IN int lsLen )
```

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>lsLen</i>	the length of list data buffer

Returns

RETURN_CODE: Values can be parsed with [device_getResponseCodeString\(\)](#)

11.9.4.63 **emv_setTerminalData()**

```
int emv_setTerminalData (
    IN BYTE * tlv,
    IN int tlvLen )
```

Set Terminal Data

Sets the Terminal Data as specified by the TerminalData structure passed as a parameter

Parameters

<i>tlv</i>	TerminalData configuration file
<i>tlvLen</i>	the length of tlv data buffer

Return values

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

11.9.4.64 `emv_setTerminalID()`

```
int emv_setTerminalID (
    IN char * terminalID )
```

Sets the terminal ID as printable characters .

Parameters

<i>terminalID</i>	Terminal ID to set
-------------------	--------------------

Returns

RETURN_CODE: Values can be parsed with [device_getResponseCodeString\(\)](#)

11.9.4.65 `emv_startTransaction()`

```
int emv_startTransaction (
    IN double amount,
    IN double amtOther,
    IN int exponent,
    IN int type,
    IN int timeout,
    IN BYTE * tags,
    IN int tagsLen,
    IN int forceOnline )
```

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) - SEE IMPORTANT NOTE BELOW
<i>amtOther</i>	Other amount value, if any (tag value 9F03) - SEE IMPORTANT NOTE BELOW
<i>exponent</i>	Number of characters after decimal point
<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 a TLV stream. Example, tag 9F0C with amount 0x000000000100 would be 0x9F0C06000000000100 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>tagsLen</i>	Length of tags
<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 = DFEE1A079F029F36959F37

Returns

RETURN_CODE: Values can be parsed with `device_getResponseCodeString` >>>>IMPORTANT: parameters for amount and amtOther MUST BE PASSED AS A DOUBLE VALUE WITH DECIMAL POINT. Example, do not pass 1, but instead pass 1.0 or 1.00. Otherwise, results will be unpredictable

11.9.4.66 `icc_getICCRaderStatus()`

```
int icc_getICCRaderStatus (
    OUT BYTE * status )
```

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
---------------	--

Returns

RETURN_CODE: Values can be parsed with `device_getResponseCodeString`

11.9.4.67 `icc_powerOffICC()`

```
int icc_powerOffICC ( )
```

Power Off ICC

Powers down the ICC

Returns

RETURN_CODE: Values can be parsed with `device_getResponseCodeString()`

If Success, empty If Failure, ASCII encoded data of error string

11.9.4.68 `icc_powerOnICC()`

```
int icc_powerOnICC (
    OUT BYTE * ATR,
    IN_OUT int * inLen )
```

Power On ICC

Power up the currently selected microprocessor card in the ICC reader

Parameters

<i>ATR,the</i>	ATR data response when succeeded power on ICC,
<i>inLen,the</i>	length of ATR data

Returns

RETURN_CODE: Values can be parsed with [device_getResponseCodeString\(\)](#)

11.9.4.69 msr_cancelMSRSwipe()

```
int msr_cancelMSRSwipe ( )
```

Disable MSR Swipe Cancels MSR swipe request.

Returns

RETURN_CODE: Values can be parsed with [device_getResponseCodeString\(\)](#)

11.9.4.70 msr_clearMSRData()

```
int msr_clearMSRData ( )
```

Clear MSR Data Clears the MSR Data buffer

Returns

RETURN_CODE: Values can be parsed with [device_getResponseCodeString\(\)](#)

11.9.4.71 msr_getMSRData()

```
int msr_getMSRData (
    OUT BYTE * reData,
    IN_OUT int * reLen )
```

Get MSR Data Reads the MSR Data buffer

Parameters

<i>reData</i>	Card data
<i>reLen</i>	Card data length

Returns

RETURN_CODE: Values can be parsed with [device_getResponseCodeString\(\)](#)

11.9.4.72 msr_registerCallBk()

```
void msr_registerCallBk (
    pMSR_callback pMSRf )
```

To register the msr callback function to get the MSR card data. (Pass NULL to disable the callback.)

11.9.4.73 msr_registerCallBkp()

```
void msr_registerCallBkp (
    pMSR_callbackp pMSRf )
```

To register the msr callback function to get the MSR card data pointer. (Pass NULL to disable the callback.)

11.9.4.74 msr_startMSRSwipe()

```
int msr_startMSRSwipe (
    IN int _timeout )
```

Start 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
----------------	---------------------

Returns

RETURN_CODE: Values can be parsed with [device_getResponseCodeString\(\)](#)

11.9.4.75 parseMSRData()

```
void parseMSRData (
    IN BYTE * resData,
    IN int resLen,
    IN_OUT IDTMSRData * cardData )
```

Parser the MSR data from the buffer into IDTMSTData structure

Parameters

<i>resData</i>	MSR card data buffer
<i>resLen</i>	the length of resData
<i>cardData</i>	the parser result with IDTMSTData structure

11.9.4.76 parsePINBlockData()

```
void parsePINBlockData (
    IN BYTE * resData,
    IN int resLen,
    IN_OUT IDTPINData * cardData )
```

Parse the PIN block data from the buffer into IDTPINData structure

Parameters

<i>resData</i>	PIN card data buffer
<i>resLen</i>	the length of resData
<i>cardData</i>	the parser result with IDTPINData structure

11.9.4.77 parsePINData()

```
void parsePINData (
    IN BYTE * resData,
    IN int resLen,
    IN_OUT IDTPINData * cardData )
```

Parse the PIN data from the buffer into IDTPINData structure

Parameters

<i>resData</i>	PIN card data buffer
<i>resLen</i>	the length of resData
<i>cardData</i>	the parser result with IDTPINData structure

11.9.4.78 pin_cancelPINEntry()

```
int pin_cancelPINEntry ( )
```

Cancel PIN Entry

Cancels PIN entry request

Returns

RETURN_CODE: Values can be parsed with [device_getResponseCodeString\(\)](#)

11.9.4.79 pin_getPIN()

```
int pin_getPIN (
    IN int mode,
    IN int PANSource,
    IN char * iccPAN,
    IN int IN iccPANLen,
    int startTimeout,
    IN int entryTimeout,
    IN char * language,
    IN int languageLen )
```

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)
<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>iccPANLen</i>	the length of <i>iccPAN</i>
<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>languageLen</i>	the length of language

Returns

RETURN_CODE: Values can be parsed with [device_getResponseCodeString\(\)](#)

11.9.4.80 pin_registerCallBk()

```
void pin_registerCallBk (
    pPIN\_callback pPINf )
```

To register the pin callback function to get the PINPad data. (Pass NULL to disable the callback.)

11.9.4.81 registerHotplugCallBk()

```
void registerHotplugCallBk (
    pMessageHotplug pMsgHotplug )
```

To register the USB HID hot-plug callback function which implemented in the application to monitor the hotplug message from the SDK.

11.9.4.82 registerLogCallBk()

```
void registerLogCallBk (
    pSendDataLog pFSend,
    pReadDataLog pFRead )
```

To register the log callback function which implemented in the application to monitor sending/reading data between application and reader.

11.9.4.83 rs232_device_init()

```
int rs232_device_init (
    int deviceType,
    int port_number,
    int brate )
```

Initial the device by RS232

It will try to connect to the device with provided deviceType, port_number, and brate.

Parameters

<i>deviceType</i>	Device to connect to
<i>port_number</i>	Port number of the device

Port nr. | Linux | Windows

0	ttyS0	COM1	1	ttyS1	COM2	2	ttyS2	COM3	3	ttyS3	COM4	4	ttyS4	COM5	5	ttyS5
6	ttyS6	COM7	7	ttyS7	COM8	8	ttyS8	COM9	9	ttyS9	COM10	10	ttyS10	COM11	11	ttyS11
12	ttyS12	COM13	13	ttyS13	COM14	14	ttyS14	COM15	15	ttyS15	COM16	16	ttyUSB0	n.a.	17	ttyUSB1
18	ttyUSB2	n.a.	19	ttyUSB3	n.a.	20	ttyUSB4	n.a.	21	ttyUSB5	n.a.	22	ttyAMA0	n.a.	23	ttyAMA1
24	ttyACM0	n.a.	25	ttyACM1	n.a.	26	rfcomm0	n.a.	27	rfcomm1	n.a.	28	ircomm0	n.a.	29	ircomm1
30	cuau0	n.a.	31	cuau1	n.a.	32	cuau2	n.a.	33	cuau3	n.a.	34	cuaU0	n.a.	35	cuaU1
36	cuaU2	n.a.	37	cuaU3	n.a.											

Parameters

<i>brate</i>	Bitrate of the device
--------------	-----------------------

Returns

RETURN_CODE: Values can be parsed with [device_getResponseCodeString\(\)](#)

11.9.4.84 SDK_Version()

```
char* SDK_Version ( )
```

To Get SDK version

Returns

return the SDK version string

11.9.4.85 setAbsoluteLibraryPath()

```
int setAbsoluteLibraryPath (
    const char * absoluteLibraryPath )
```

Set the path to use when searching for ID TECH's libraries. If this is not set, the libraries will be searched for with the system's default procedures.

Parameters

<i>absoluteLibraryPath</i>	The absolute path to ID TECH's libraries.
----------------------------	---

Returns

RETURN_CODE: Values can be parsed with [device_getResponseCodeString\(\)](#)

11.10 Source_C/libIDT_SREDKey2.h File Reference

SREDKey2 API.

```
#include "IDTDef.h"
```

Macros

- `#define IN`
- `#define OUT`
- `#define IN_OUT`

Typedefs

- `typedef void(* pMessageHotplug) (int, int)`
- `typedef void(* pSendDataLog) (BYTE *, int)`
- `typedef void(* pReadDataLog) (BYTE *, int)`
- `typedef void(* pEMV_callBack) (int, int, BYTE *, int, IDTTransactionData *, EMV_Callback *, int)`
- `typedef void(* pFW_callBack) (int, int, int, int, int)`
- `typedef void(* pMSR_callBack) (int, IDTMSRData)`
- `typedef void(* pMSR_callBackp) (int, IDTMSRData *)`
- `typedef void(* pPIN_callBack) (int, IDTPINData *)`
- `typedef void(* pCMR_callBack) (int, IDTCMRData *)`
- `typedef void(* pCSFS_callBack) (BYTE status)`
- `typedef void(* pLCD_callBack) (int, IDTLCDItem *)`
- `typedef void(* ftpComm_callBack) (int, int, int)`
- `typedef void(* httpComm_callBack) (BYTE *, int)`
- `typedef void(* v4Comm_callBack) (BYTE, BYTE, BYTE *, int)`

Functions

- void [registerHotplugCallBk](#) (pMessageHotplug pMsgHotplug)
- void [registerLogCallBk](#) (pSendDataLog pFSend, pReadDataLog pFRead)
- void [device_registerFWCallBk](#) (pFW_callBack pFWf)
- void [device_registerCameraCallBk](#) (pCMR_callBack pCMRf)
- void [device_registerCardStatusFrontSwitchCallBk](#) (pCSFS_callBack pCSFSf)
- void [emv_registerCallBk](#) (pEMV_callBack pEMVf)
- void [msr_registerCallBk](#) (pMSR_callBack pMSRf)
- void [msr_registerCallBkp](#) (pMSR_callBackp pMSRf)
- void [ctls_registerCallBk](#) (pMSR_callBack pCTLSf)
- void [ctls_registerCallBkp](#) (pMSR_callBackp pCTLSf)
- void [pin_registerCallBk](#) (pPIN_callBack pPINf)
- void [lcd_registerCallBk](#) (pLCD_callBack pLCDf)

- void `comm_registerHTTPCallback` (`httpComm_callback` cBack)
- void `comm_registerV4Callback` (`v4Comm_callback` cBack)
- char * `SDK_Version` ()
- int `setAbsoluteLibraryPath` (const char *absoluteLibraryPath)
- int `device_setConfigPath` (const char *path)
- int `device_setNEO2DevicesConfigs` (IN const char *configs, IN int len)
- int `device_init` ()
- int `rs232_device_init` (int deviceType, int port_number, int brate)
- int `device_setCurrentDevice` (int deviceType)
- int `device_isAttached` (int deviceType)
- int `device_close` ()
- void `device_getIDGStatusCodeString` (IN int returnCode, OUT char *despcrition)
- int `device_isConnected` ()
- int `device_getFirmwareVersion` (OUT char *firmwareVersion)
- int `device_getFirmwareVersion_Len` (OUT char *firmwareVersion, IN_OUT int *firmwareVersionLen)
- int `device_pingDevice` ()
- int `device_getCurrentDeviceType` ()
- int `device_SendDataCommandNEO` (IN int cmd, IN int subCmd, IN BYTE *data, IN int dataLen, OUT BYTE *response, IN_OUT int *respLen)
- int `device_SendDataCommand` (IN BYTE *cmd, IN int cmdLen, IN BYTE *data, IN int dataLen, OUT BYTE *response, IN_OUT int *respLen)
- int `device_rebootDevice` ()
- int `device_SendDataCommandITP` (IN BYTE *cmd, IN int cmdLen, IN BYTE *data, IN int dataLen, OUT BYTE *response, IN_OUT int *respLen)
- void `device_setTransactionExponent` (int exponent)
- int `device_getKeyStatus` (int *newFormat, BYTE *status, int *statusLen)
- int `device_updateFirmware` (IN BYTE *firmwareData, IN int firmwareDataLen, IN char *firmwareName, IN int encryptionType, IN BYTE *keyBlob, IN int keyBlobLen)
- int `config_getModelNumber` (OUT char *sNumber)
- int `config_getModelNumber_Len` (OUT char *sNumber, IN_OUT int *sNumberLen)
- int `config_getSerialNumber` (OUT char *sNumber)
- int `config_getSerialNumber_Len` (OUT char *sNumber, IN_OUT int *sNumberLen)
- int `device_setSystemLanguage` (char *language)
- int `msr_setExpirationMask` (int mask)
- int `msr_getExpirationMask` (BYTE *value)
- int `msr_setClearPANID` (BYTE val)
- int `msr_getClearPANID` (BYTE *value)
- int `msr_setSwipeForcedEncryptionOption` (int track1, int track2, int track3, int track3card0)
- int `msr_getSwipeForcedEncryptionOption` (BYTE *option)
- int `msr_setSwipeMaskOption` (int track1, int track2, int track3)
- int `msr_getSwipeMaskOption` (BYTE *option)
- int `msr_getFunctionStatus` (int *enable, int *isBufferMode, int *withNotification)
- int `msr_disable` ()

11.10.1 Detailed Description

SREDKey2 API.

SREDKey2 Global API methods.

11.10.2 Macro Definition Documentation

11.10.2.1 IN

```
#define IN
```

INPUT parameter.

11.10.2.2 IN_OUT

```
#define IN_OUT
```

INPUT / OUTPUT PARAMETER.

11.10.2.3 OUT

```
#define OUT
```

OUTPUT parameter.

11.10.3 Typedef Documentation**11.10.3.1 ftpComm_callBack**

```
typedef void(* ftpComm_callBack) (int, int, int)
```

Define the comm callback function to get FTP file transfer status

It should be passed as a parameter in a FTP request, Signature (int, int, int) = response code, current block, total blocks
 RESPONSE CODES: 100 = FILE DOWNLOAD STARTED 101 = FILE BLOCK XX OF XX RECEIVED 102 = FILE DOWNLOAD COMPLETED 103 = FILE DOWNLOAD TERMINATED PREMATURELY

11.10.3.2 httpComm_callBack

```
typedef void(* httpComm_callBack) (BYTE *, int)
```

Define the comm callback function to get the async url data

It should be registered using the comm_registerHTTPCallback

11.10.3.3 pCMR_callBack

```
typedef void(* pCMR_callBack) (int, IDTCMRData *)
```

Define the camera callback function to get the image data

It should be registered using the device_registerCameraCallBk,

11.10.3.4 pCSFS_callBack

```
typedef void(* pCSFS_callBack) (BYTE status)
```

Define the card status and front switch callback function to get card and front switch status

It should be registered using the device_registerCardStatusFrontSwitchCallBk,

11.10.3.5 pEMV_callBack

```
typedef void(* pEMV_callBack) (int, int, BYTE *, int, IDTTransactionData *, EMV_Callback *, int)
```

Define the EMV callback function to get the transaction message/data/result.
It should be registered using the `emv_registerCallBk`

11.10.3.6 pFW_callBack

```
typedef void(* pFW_callBack) (int, int, int, int, int)
```

Define the firmware update callback function to get the firmware update status
It should be registered using the `device_registerFWCallBk`

11.10.3.7 pLCD_callBack

```
typedef void(* pLCD_callBack) (int, IDTLCDItem *)
```

Define the LCD callback function to get the input LCDItem
It should be registered using the `lcd_registerCallBk`,

11.10.3.8 pMessageHotplug

```
typedef void(* pMessageHotplug) (int, int)
```

Define the USB hot-plug callback function to monitor the info when plug in/out the reader.
It should be registered using the `registerHotplugCallBk`, The first integer parameter is device type, and the second integer parameter is either 0: Device Plugged Out or 1: Device Plugged In

11.10.3.9 pMSR_callBack

```
typedef void(* pMSR_callBack) (int, IDTMSRData)
```

Define the MSR callback function to get the MSR card data
It should be registered using the `msr_registerCallBk`, this callback function is for backward compatibility

11.10.3.10 pMSR_callBackp

```
typedef void(* pMSR_callBackp) (int, IDTMSRData *)
```

Define the MSR callback function to get pointer to the MSR card data
It should be registered using the `msr_registerCallBk`, this callback function is recommended instead of `pMSR_callBack`

11.10.3.11 pPIN_callBack

```
typedef void(* pPIN_callBack) (int, IDTPINData *)
```

Define the PINPad callback function to get the input PIN Pad data
It should be registered using the `pin_registerCallBk`,

11.10.3.12 pReadDataLog

```
typedef void(* pReadDataLog) (BYTE *, int)
```

Define the read response callback function to monitor the reading response from the reader.
It should be registered using the `registerLogCallBk`

11.10.3.13 pSendDataLog

```
typedef void(* pSendDataLog) (BYTE *, int)
```

Define the send command callback function to monitor the sending command into the reader.
It should be registered using the registerLogCallBk

11.10.3.14 v4Comm_callBack

```
typedef void(* v4Comm_callBack) (BYTE, BYTE, BYTE *, int)
```

Define the comm callback function to receive the V4 Protocol packets received by the device from an external source (IP/USB/RS-232) It should be registered using the comm_registerV4Callback, Data callback will contain command, sub-command, and data from V4 packet

11.10.4 Function Documentation

11.10.4.1 comm_registerHTTPCallback()

```
void comm_registerHTTPCallback (
    httpComm_callBack cBack )
```

Register Comm HTTP Async Callback

Parameters

<i>cBack</i>	- HTTP Comm callback
--------------	----------------------

11.10.4.2 comm_registerV4Callback()

```
void comm_registerV4Callback (
    v4Comm_callBack cBack )
```

Register External V4 Protocol commands Callback

Parameters

<i>cBack</i>	- V4 Protocol Comm callback
--------------	-----------------------------

11.10.4.3 config_getModelNumber()

```
int config_getModelNumber (
    OUT char * sNumber )
```

DEPRECATED : please use [config_getModelNumber_Len\(OUT char* sNumber, IN_OUT int *sNumberLen\)](#)

Polls device for Model Number

Parameters

<i>sNumber</i>	Returns Model Number; needs to have at least 64 bytes of memory
----------------	---

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString

11.10.4.4 config_getModelNumber_Len()

```
int config_getModelNumber_Len (
    OUT char * sNumber,
    IN_OUT int * sNumberLen )
```

Polls device for Model Number

Parameters

<i>sNumber</i>	Returns Model Number
<i>sNumber</i>	length of Model Number

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString

11.10.4.5 config_getSerialNumber()

```
int config_getSerialNumber (
    OUT char * sNumber )
```

DEPRECATED : please use [config_getSerialNumber_Len\(OUT char* sNumber, IN_OUT int *sNumberLen\)](#)

Polls device for Serial Number

Parameters

<i>sNumber</i>	Returns Serial Number; needs to have at least 64 bytes of memory
----------------	--

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString

11.10.4.6 config_getSerialNumber_Len()

```
int config_getSerialNumber_Len (
    OUT char * sNumber,
    IN_OUT int * sNumberLen )
```

Polls device for Serial Number

Parameters

<i>sNumber</i>	Returns Serial Number
<i>sNumberLen</i>	Length of Serial Number

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString

11.10.4.7 ctls_registerCallBk()

```
void ctls_registerCallBk (
    pMSR_callBack pCTLSf )
```

To register the ctls callback function to get the MSR card data. (Pass NULL to disable the callback.)

11.10.4.8 ctls_registerCallBkp()

```
void ctls_registerCallBkp (
    pMSR_callBackp pCTLSf )
```

To register the ctls callback function to get the MSR card data pointer. (Pass NULL to disable the callback.)

11.10.4.9 device_close()

```
int device_close ( )
```

Close the device

Returns

RETURN_CODE: 0: success, 0x0A: failed

11.10.4.10 device_getCurrentDeviceType()

```
int device_getCurrentDeviceType ( )
```

Get current active device type

Returns

: return the device type defined as DEVICE_TYPE in the IDTDef.h

11.10.4.11 device_getFirmwareVersion()

```
int device_getFirmwareVersion (
    OUT char * firmwareVersion )
```

DEPRECATED : please use [device_getFirmwareVersion_Len](#)(OUT char* firmwareVersion, IN_OUT int *firmwareVersionLen)

Polls device for Firmware Version

Parameters

<i>firmwareVersion</i>	Response returned of Firmware Version; needs to have at least 128 bytes of memory
------------------------	---

Returns

RETURN_CODE: Values can be parsed with [device_getIDGStatusCodeString\(\)](#)

11.10.4.12 device_getFirmwareVersion_Len()

```
int device_getFirmwareVersion_Len (
    OUT char * firmwareVersion,
    IN_OUT int * firmwareVersionLen )
```

Polls device for Firmware Version

Parameters

<i>firmwareVersion</i>	Response returned of Firmware Version
<i>firmwareVersionLen</i>	Length of Firmware Version

Returns

RETURN_CODE: Values can be parsed with [device_getIDGStatusCodeString\(\)](#)

11.10.4.13 device_getIDGStatusCodeString()

```
void device_getIDGStatusCodeString (
    IN int returnCode,
    OUT char * despcrition )
```

Review the return code description.

Parameters

<i>returnCode</i>	the response result.
<i>description</i>	

Return values

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

- 0: "no error, beginning task";
- 1: "no response from reader";
- 2: "invalid response data";
- 01: " Incorrect Header Tag";
- 02: " Unknown Command";
- 03: " Unknown Sub-Command";
- 04: " CRC Error in Frame";

- 05: " Incorrect Parameter";
- 06: " Parameter Not Supported";
- 07: " Mal-formatted Data";
- 08: " Timeout";
- 0A: " Failed / NACK";
- 0B: " Command not Allowed";
- 0C: " Sub-Command not Allowed";
- 0D: " Buffer Overflow (Data Length too large for reader buffer)";
- 0E: " User Interface Event";
- 10: " Need clear firmware(apply in boot loader only)";
- 11: " Communication type not supported, VT-1, burst, etc. Need encrypted firmware (apply in boot loader only)";
- 12: " Secure interface is not functional or is in an intermediate state.";
- 13: " Data field is not mod 8";
- 14: " Pad 0x80 not found where expected";
- 15: " Specified key type is invalid";
- 16: " Could not retrieve key from the SAM (InitSecureComm)";
- 17: " Hash code problem";
- 18: " Could not store the key into the SAM (InstallKey)";
- 19: " Frame is too large";
- 1A: " Unit powered up in authentication state but POS must resend the InitSecureComm command";
- 1B: " The EEPROM may not be initialized because SecCommInterface does not make sense";
- 1C: " Problem encoding APDU Module-Specific Status Codes ";
- 20: " Unsupported Index (ILM) SAM Transceiver error - problem communicating with the SAM (Key Mgr)";
- 21: " Unexpected Sequence Counter in multiple frames for single bitmap (ILM)Length error in data returned from the SAM (Key Mgr) - 22: " Improper bit map (ILM)"; - 23: " Request Online Authorization"; - 24: " ViV↔ OCard3 raw data read successful"; - 25: " Message index not available (ILM) ViVOcomm activate transaction card type (ViVOcomm)"; - 26: " Version Information Mismatch (ILM)"; - 27: " Not sending commands in correct index message index (ILM)"; - 28: " Time out or next expected message not received (ILM)"; - 29: " ILM languages not available for viewing (ILM)"; - 2A: " Other language not supported (ILM)"; - 41: " from 41 to 4F, Module-specific errors for Key Manager";
- 50: " Auto-Switch OK";
- 51: " Auto-Switch failed";
- 70: " Antenna Error 80h Use another card";
- 81: " Insert or swipe card";
- 90: " Data encryption Key does not exist";
- 91: " Data encryption Key KSN exhausted";

11.10.4.14 device_getKeyStatus()

```
int device_getKeyStatus (
    int * newFormat,
    BYTE * status,
    int * statusLen )
```

Get Key Status

Gets the status of loaded keys

Parameters

<i>status</i>	newFormat for Augusta and miniSmartII only 1: new format of key status 0: reserved format for support previous device
<i>status</i>	For L80, L100, Augusta and miniSmartII: When the newFormat is 0, data format as follows. For Augusta and miniSmartII: 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 For L80 and L100: byte 0: PIN DUKPT Key byte 1: PIN Master Key byte 2: Standard PIN Session Key byte 3: Desjardins PIN Session Key byte 4: Account/MSR DUKPT Key, 1 Exists, 0 None, 0xFF STOP, Does not support, always 0 byte 5: Account/ICC DUKPT Key, 1 Exists, 0 None, 0xFF STOP, Does not support, always 0 byte 6: Admin DUKPT Key, 1 Exists, 0 None, 0xFF STOP byte 7: Data DUKPT Key, 1 Exists, 0 None, 0xFF STOP byte 8: MAC DUKPT Key, 1 Exists, 0 None, 0xFF STOP

when the newFormat is 1, 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) For NEO2 and SREDKey2: Each unit of three bytes represents one key's parameters (index and slot). Key Name Index (1 byte): 0x14 - LCL-KEK 0x01 - Pin encryption Key (NEO2 only) 0x02 - Data encryption Key 0x05 - MAC DUKPT Key 0x0A - PCI Pairing Key (NEO2 only) Key Slot (2 bytes): Indicate different slots of a certain Key Name Example: slot =5 (0x00 0x05), slot=300 (0x01 0x2C) For BTPay380, slot is always 0 For example, 0x14 0x00 0x00 0x02 0x00 0x00 0x0A 0x00 0x00 will represent [KeyNameIndex=0x14,KeySlot=0x0000], [KeyNameIndex=0x02,KeySlot=0x0000] and [KeyNameIndex=0x0A,KeySlot=0x0000]

Parameters

<i>statusLen</i>	the length of status
------------------	----------------------

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString

11.10.4.15 device_init()

```
int device_init ( )
```

Initial the device by USB

It will detect the device and trying connect.

The connect status can be checked by [device_isConnected\(\)](#).

Note: after the function returns success, the function [device_setCurrentDevice\(\)](#) has to be called to set the device type.

Returns

RETURN_CODE: Values can be parsed with [device_getResponseCodeString\(\)](#)

11.10.4.16 device_isAttached()

```
int device_isAttached (
    int deviceType )
```

Check if the device is attached to the USB port The function [device_init\(\)](#) must be called before this function.

Parameters

<i>deviceType,the</i>	device type of the USB device
-----------------------	-------------------------------

Returns

1 if the device is attached, or 0 if the device is not attached

11.10.4.17 device_isConnected()

```
int device_isConnected ( )
```

Check the device connected status

Returns

DEVICE_DISCONNECT=0, or DEVICE_CONNECTED = 1

11.10.4.18 device_pingDevice()

```
int device_pingDevice ( )
```

Ping Device

Pings the reader. If connected, returns success. Otherwise, returns timeout.

Returns

RETURN_CODE: Values can be parsed with [device_getIDGStatusCodeString\(\)](#)

11.10.4.19 device_rebootDevice()

```
int 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_getIDGStatusCodeString\(\)](#)

11.10.4.20 device_registerCameraCallBk()

```
void device_registerCameraCallBk (
    pCMR_callback pCMRf )
```

To register the camera callback function to get the image data. (Pass NULL to disable the callback.)

11.10.4.21 device_registerCardStatusFrontSwitchCallBk()

```
void device_registerCardStatusFrontSwitchCallBk (
    pCSFS_callback pCSFSf )
```

To register the card status and front switch callback function to get status. (Pass NULL to disable the callback.)

11.10.4.22 device_registerFWCallBk()

```
void device_registerFWCallBk (
    pFW_callback pFWf )
```

To register the firmware update callback function to get the firmware update processing response. (Pass NULL to disable the callback.)

11.10.4.23 device_SendDataCommand()

```
int device_SendDataCommand (
    IN BYTE * cmd,
    IN int cmdLen,
    IN BYTE * data,
    IN int dataLen,
    OUT BYTE * response,
    IN_OUT int * respLen )
```

Send a Command to NGA device

Sends a command to the device .

Parameters

<i>cmd</i>	buffer of command to execute.
<i>cmdLen,the</i>	length of the buffer cmd.
<i>data</i>	buffer of IDG command data.
<i>dataLen,the</i>	length of the buffer data.
<i>response</i>	Response data
<i>respLen,the</i>	length of Response data

Returns

RETURN_CODE: Values can be parsed with [device_getResponseCodeString\(\)](#)

11.10.4.24 device_SendDataCommandITP()

```
int device_SendDataCommandITP (
    IN BYTE * cmd,
    IN int cmdLen,
    IN BYTE * data,
    IN int dataLen,
    OUT BYTE * response,
    IN_OUT int * respLen )
```

Send a Command to ITP device

Sends a command to the device .

Parameters

<i>cmd</i>	buffer of command to execute.
<i>cmdLen,the</i>	length of the buffer cmd.
<i>data</i>	buffer of ITP command data.
<i>dataLen,the</i>	length of the buffer data.
<i>response</i>	Response data
<i>respLen,the</i>	length of Response data

Returns

RETURN_CODE: Values can be parsed with [device_getResponseCodeString\(\)](#)

11.10.4.25 device_SendDataCommandNEO()

```
int device_SendDataCommandNEO (
    IN int cmd,
    IN int subCmd,
    IN BYTE * data,
    IN int dataLen,
    OUT BYTE * response,
    IN_OUT int * respLen )
```

Send a Command to NEO device

Sends a command to the NEO device .

Parameters

<i>cmd</i>	command to execute.
<i>subCmd,sub</i>	command to execute.
<i>data</i>	buffer of NEO command data.
<i>dataLen,the</i>	length of the buffer data.
<i>response</i>	Response data
<i>respLen,the</i>	length of Response data

Returns

RETURN_CODE: Values can be parsed with [device_getIDGStatusCodeString\(\)](#)

11.10.4.26 device_setConfigPath()

```
int device_setConfigPath (
    const char * path )
```

Set the path to the config xml file(s) if any

Parameters

<i>path</i>	The path to the config xml files (such as "NEO2_Devices.xml" which contains the information of NEO2 devices). Only need to specify the path to the folder which contains the config files. File names are not needed. The maximum length of path is 200 characters including the '\0' at the end.
-------------	---

Returns

RETURN_CODE: Values can be parsed with [device_getIDGStatusCodeString\(\)](#)

11.10.4.27 device_setCurrentDevice()

```
int device_setCurrentDevice (
    int deviceType )
```

Sets the current device to talk to

The connect status can be checked by [device_isConnected\(\)](#).

Parameters

<i>deviceType</i>	<p>Device to connect to</p> <pre>enum DEVICE_TYPE { IDT_DEVICE_UNKNOWN=0, IDT_DEVICE_AUGUSTA_HID, IDT_DEVICE_AUGUSTA_KB, IDT_DEVICE_AUGUSTA_S_HID, IDT_DEVICE_AUGUSTA_S_KB, IDT_DEVICE_AUGUSTA_S_TTK_HID, IDT_DEVICE_SPECTRUM_PRO, IDT_DEVICE_MINISMART_II, IDT_DEVICE_L80, IDT_DEVICE_L100, IDT_DEVICE_UNIPAY, IDT_DEVICE_UNIPAY_I_V, IDT_DEVICE_VP3300_AJ, IDT_DEVICE_KIOSK_III, IDT_DEVICE_KIOSK_III_S, IDT_DEVICE_PIP_READER, IDT_DEVICE_VENDI, IDT_DEVICE_VP3300_USB, IDT_DEVICE_UNIPAY_I_V_TTK, IDT_DEVICE_VP3300_BT, IDT_DEVICE_VP8800, IDT_DEVICE_SREDKEY2_HID, IDT_DEVICE_SREDKEY2_KB, IDT_DEVICE_NEO2, IDT_DEVICE_MINISMART_II_COM = IDT_DEVICE_NEO2+5, IDT_DEVICE_SPECTRUM_PRO_COM, IDT_DEVICE_KIOSK_III_COM, IDT_DEVICE_KIOSK_III_S_COM, IDT_DEVICE_PIP_READER_COM, IDT_DEVICE_VP3300_COM, IDT_DEVICE_NEO2_COM, IDT_DEVICE_MAX_DEVICES = IDT_DEVICE_NEO2_COM+5 };</pre>
-------------------	---

Returns

RETURN_CODE: 1: success, 0: failed

11.10.4.28 device_setNEO2DevicesConfigs()

```
int device_setNEO2DevicesConfigs (
```

```
IN const char * configs,
IN int len )
```

Pass the content of the config xml file ("NEO2_Devices.xml") as a string to the SDK instead of reading the config xml file by the SDK. It needs to be called before [device_init\(\)](#), otherwise the SDK will try to read the config xml file.

Parameters

<i>configs</i>	The content read from the config xml file ("NEO2_Devices.xml" which contains the information of NEO2 devices).
<i>len</i>	The length of the string configs. The maximum length is 5000 bytes.

11.10.4.29 device_setSystemLanguage()

```
int device_setSystemLanguage (
    char * language )
```

Set System Language Sets the language for the message displayed in the LCD screen

Parameters

<i>language</i>	2-byte ASCII code, can be "EN" or "JP"
-----------------	--

Returns

success or error code. Values can be parsed with [device_getIDGStatusCodeString](#)

See also

[ErrorCode](#)

11.10.4.30 device_setTransactionExponent()

```
void device_setTransactionExponent (
    int exponent )
```

Sets the transaction exponent to be used with [device_startTransaction](#). Default value is 2

Parameters

<i>exponent, The</i>	exponent to use when calling device_startTransaction
----------------------	--

11.10.4.31 device_updateFirmware()

```
int device_updateFirmware (
    IN BYTE * firmwareData,
    IN int firmwareDataLen,
    IN char * firmwareName,
```

```

    IN int encryptionType,
    IN BYTE * keyBlob,
    IN int keyBlobLen )

```

Update Firmware Updates the firmware of NEO 2 devices.

Parameters

<i>firmwareData</i>	Signed binary data of a firmware file provided by IDTech
<i>firmwareDataLen</i>	Length of firmwareData
<i>firmwareName</i>	Firmware name. <ul style="list-style-type: none">• For example "VP5300_v1.00.023.0167.S_Test.fm"
<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)
<i>keyBlobLen</i>	Length of keyBlob

Returns

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

Firmware update status is returned in the callback with the following values: sender = device type state = DEVICE_↔
E_FIRMWARE_UPDATE current block total blocks ResultCode:

- 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

11.10.4.32 emv_registerCallBk()

```

void emv_registerCallBk (
    pEMV_callback pEMVf )

```

To register the emv callback function to get the EMV processing response. (Pass NULL to disable the callback.)

11.10.4.33 lcd_registerCallBk()

```

void lcd_registerCallBk (
    pLCD_callback pLCDf )

```

To register the lcd callback function to get the LCDItem. (Pass NULL to disable the callback.)

11.10.4.34 msr_disable()

```

int msr_disable ( )

```

Disable MSR Disable MSR functions.

Returns

RETURN_CODE: Values can be parsed with [device_getResponseCodeString\(\)](#)

11.10.4.35 msr_getClearPANID()

```
int msr_getClearPANID (
    BYTE * value )
```

Get Clear PAN ID.

Returns the number of digits that begin the PAN that will be in the clear

Parameters

<i>value</i>	4901 <Setting value>="". setting Value: Number of digits in clear. Values are char '0' - '6'
--------------	--

Returns

RETURN_CODE: Values can be parsed with [device_getResponseCodeString](#)

11.10.4.36 msr_getExpirationMask()

```
int msr_getExpirationMask (
    BYTE * value )
```

Get MSR expiration date mask.

Parameters

<i>value</i>	5001 <Setting value>="". setting Value: '0' = masked, '1' = not-masked
--------------	--

Returns

RETURN_CODE: Values can be parsed with [device_getResponseCodeString](#)

11.10.4.37 msr_getFunctionStatus()

```
int msr_getFunctionStatus (
    int * enable,
    int * isBufferMode,
    int * withNotification )
```

Get MSR Function Status.

Gets the MSR function status

Parameters

<i>enable</i>	1 = MSR enabled, 0 = MSR disabled
<i>isBufferMode</i>	1 = buffer mode, 0 = auto mode
<i>withNotification</i>	1 = with notification, 0 = without notification

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString

11.10.4.38 msr_getSwipeForcedEncryptionOption()

```
int msr_getSwipeForcedEncryptionOption (
    BYTE * option )
```

Get MSR Swipe Forced Encryption Option.

Parameters

<i>option</i>	8401 <Setting value>="". Setting Value 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
---------------	---

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString

11.10.4.39 msr_getSwipeMaskOption()

```
int msr_getSwipeMaskOption (
    BYTE * option )
```

Get MSR Swipe Mask Option.

Gets the swipe mask/clear data sending option

Parameters

<i>option</i>	8601 <Setting value>="". Setting Value 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
---------------	---

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString

11.10.4.40 msr_registerCallBk()

```
void msr_registerCallBk (
    pMSR_callback pMSRf )
```

To register the msr callback function to get the MSR card data. (Pass NULL to disable the callback.)

11.10.4.41 msr_registerCallBkp()

```
void msr_registerCallBkp (
    pMSR_callbackp pMSRf )
```

To register the msr callback function to get the MSR card data pointer. (Pass NULL to disable the callback.)

11.10.4.42 msr_setClearPANID()

```
int msr_setClearPANID (
    BYTE val )
```

Set Clear PAN ID.

Parameters

<i>val</i>	Set Clear PAN ID to value: Number of digits to show in clear. Range 0-6.
------------	--

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString

11.10.4.43 msr_setExpirationMask()

```
int msr_setExpirationMask (
    int mask )
```

Set Expiration Masking

Sets the flag to mask the expiration date

Parameters

<i>mask</i>	TRUE = mask expiration
-------------	------------------------

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString

11.10.4.44 msr_setSwipeForcedEncryptionOption()

```
int msr_setSwipeForcedEncryptionOption (
    int track1,
    int track2,
    int track3,
    int track3card0 )
```

Set MSR Swipe Forced Encryption Option.

Parameters

<i>track1</i>	Set track1 encryption to true or false.
<i>track2</i>	Set track2 encryption to true or false.
<i>track3</i>	Set track3 encryption to true or false.
<i>track3card0</i>	Set track3 card0 encryption to true or false.

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString

11.10.4.45 msr_setSwipeMaskOption()

```
int msr_setSwipeMaskOption (
    int track1,
    int track2,
    int track3 )
```

Set MSR Swipe Mask Option.

Sets the swipe mask/clear data sending option

Parameters

<i>track1</i>	Set track1 mask to true or false.
<i>track2</i>	Set track2 mask to true or false.
<i>track3</i>	Set track3 mask to true or false.

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString

11.10.4.46 pin_registerCallBk()

```
void pin_registerCallBk (
    pPIN_callback pPINf )
```

To register the pin callback function to get the PINPad data. (Pass NULL to disable the callback.)

11.10.4.47 registerHotplugCallBk()

```
void registerHotplugCallBk (
    pMessageHotplug pMsgHotplug )
```

To register the USB HID hot-plug callback function which implemented in the application to monitor the hotplug message from the SDK.

11.10.4.48 registerLogCallBk()

```
void registerLogCallBk (
    pSendDataLog pFSend,
    pReadDataLog pFRead )
```

To register the log callback function which implemented in the application to monitor sending/reading data between application and reader.

11.10.4.49 rs232_device_init()

```
int rs232_device_init (
    int deviceType,
```



```
int port_number,
int brate )
```

Initial the device by RS232

It will try to connect to the device with provided deviceType, port_number, and brate.

Parameters

<i>deviceType</i>	Device to connect to
<i>port_number</i>	Port number of the device

Port nr. | Linux | Windows

```
| 0 | ttyS0 | COM1 | | 1 | ttyS1 | COM2 | | 2 | ttyS2 | COM3 | | 3 | ttyS3 | COM4 | | 4 | ttyS4 | COM5 | | 5 | ttyS5 |
COM6 | | 6 | ttyS6 | COM7 | | 7 | ttyS7 | COM8 | | 8 | ttyS8 | COM9 | | 9 | ttyS9 | COM10 | | 10 | ttyS10 | COM11
| | 11 | ttyS11 | COM12 | | 12 | ttyS12 | COM13 | | 13 | ttyS13 | COM14 | | 14 | ttyS14 | COM15 | | 15 | ttyS15 |
COM16 | | 16 | ttyUSB0 | n.a. | | 17 | ttyUSB1 | n.a. | | 18 | ttyUSB2 | n.a. | | 19 | ttyUSB3 | n.a. | | 20 | ttyUSB4 |
n.a. | | 21 | ttyUSB5 | n.a. | | 22 | ttyAMA0 | n.a. | | 23 | ttyAMA1 | n.a. | | 24 | ttyACM0 | n.a. | | 25 | ttyACM1 | n.a.
| | 26 | rfcomm0 | n.a. | | 27 | rfcomm1 | n.a. | | 28 | ircomm0 | n.a. | | 29 | ircomm1 | n.a. | | 30 | cuau0 | n.a. | | 31
| cuau1 | n.a. | | 32 | cuau2 | n.a. | | 33 | cuau3 | n.a. | | 34 | cuaU0 | n.a. | | 35 | cuaU1 | n.a. | | 36 | cuaU2 | n.a. |
| 37 | cuaU3 | n.a. |
```

Parameters

<i>brate</i>	Bitrate of the device
--------------	-----------------------

Returns

RETURN_CODE: Values can be parsed with [device_getResponseCodeString\(\)](#)

11.10.4.50 SDK_Version()

```
char* SDK_Version ( )
```

To Get SDK version

Returns

return the SDK version string

11.10.4.51 setAbsoluteLibraryPath()

```
int setAbsoluteLibraryPath (
    const char * absoluteLibraryPath )
```

Set the path to use when searching for ID TECH's libraries. If this is not set, the libraries will be searched for with the system's default procedures.

Parameters

<i>absoluteLibraryPath</i>	The absolute path to ID TECH's libraries.
----------------------------	---

Returns

RETURN_CODE: Values can be parsed with [device_getIDGStatusCodeString\(\)](#)

11.11 Source_C/libIDT_UniPayl_V.h File Reference

UniPay 1.5 API.

```
#include "IDTDef.h"
```

Macros

- [#define IN](#)
- [#define OUT](#)
- [#define IN_OUT](#)

Typedefs

- typedef void(* [pMessageHotplug](#)) (int, int)
- typedef void(* [pSendDataLog](#)) (BYTE *, int)
- typedef void(* [pReadDataLog](#)) (BYTE *, int)
- typedef void(* [pEMV_callBack](#)) (int, int, BYTE *, int, IDTTransactionData *, EMV_Callback *, int)
- typedef void(* [pMSR_callBack](#)) (int, IDTMSRData)
- typedef void(* [pMSR_callBackp](#)) (int, IDTMSRData *)
- typedef void(* [pPIN_callBack](#)) (int, IDTPINData *)
- typedef void(* [pCMR_callBack](#)) (int, IDTCMRData *)
- typedef void(* [pCSFS_callBack](#)) (BYTE status)
- typedef void(* [ftpComm_callBack](#)) (int, int, int)
- typedef void(* [httpComm_callBack](#)) (BYTE *, int)
- typedef void(* [v4Comm_callBack](#)) (BYTE, BYTE, BYTE *, int)

Functions

- void [registerHotplugCallBk](#) ([pMessageHotplug](#) pMsgHotplug)
- void [registerLogCallBk](#) ([pSendDataLog](#) pFSend, [pReadDataLog](#) pFRead)
- void [emv_registerCallBk](#) ([pEMV_callBack](#) pEMVf)
- void [msr_registerCallBk](#) ([pMSR_callBack](#) pMSRf)
- void [msr_registerCallBkp](#) ([pMSR_callBackp](#) pMSRf)
- void [pin_registerCallBk](#) ([pPIN_callBack](#) pPINf)
- void [device_registerCameraCallBk](#) ([pCMR_callBack](#) pCMRf)
- void [device_registerCardStatusFrontSwitchCallBk](#) ([pCSFS_callBack](#) pCSFSf)
- void [comm_registerHTTPCallback](#) ([httpComm_callBack](#) cBack)
- void [comm_registerV4Callback](#) ([v4Comm_callBack](#) cBack)
- char * [SDK_Version](#) ()
- int [setAbsoluteLibraryPath](#) (const char *absoluteLibraryPath)
- int [device_init](#) ()
- int [device_setCurrentDevice](#) (int deviceType)
- int [device_close](#) ()
- void [device_getIDGStatusCodeString](#) ([IN](#) int returnCode, [OUT](#) char *despcrition)
- int [device_isConnected](#) ()
- int [device_isAttached](#) (int deviceType)
- int [device_getFirmwareVersion](#) ([OUT](#) char *firmwareVersion)

- int [device_getFirmwareVersion_Len](#) (OUT char *firmwareVersion, IN_OUT int *firmwareVersionLen)
- int [device_pingDevice](#) ()
- int [device_getCurrentDeviceType](#) ()
- int [device_SendDataCommandNEO](#) (IN int cmd, IN int subCmd, IN BYTE *data, IN int dataLen, OUT BYTE *response, IN_OUT int *respLen)
- int [device_enablePassThrough](#) (int enablePassThrough)
- int [device_setMerchantRecord](#) (int index, int enabled, char *merchantID, char *merchantURL)
- int [device_getMerchantRecord](#) (IN int index, OUT BYTE *record)
- int [device_getMerchantRecord_Len](#) (IN int index, OUT BYTE *record, IN_OUT int *recordLen)
- int [device_getSDKWaitTime](#) ()
- void [device_setSDKWaitTime](#) (int waitTime)
- int [device_getThreadStackSize](#) ()
- void [device_setThreadStackSize](#) (int threadSize)
- int [config_getSerialNumber](#) (OUT char *sNumber)
- int [config_getSerialNumber_Len](#) (OUT char *sNumber, IN_OUT int *sNumberLen)
- void [emv_allowFallback](#) (IN int allow)
- void [emv_setAutoAuthenticateTransaction](#) (IN int authenticate)
- void [emv_setAutoCompleteTransaction](#) (IN int complete)
- int [emv_getAutoAuthenticateTransaction](#) ()
- int [emv_getAutoCompleteTransaction](#) ()
- int [emv_startTransaction](#) (IN double amount, IN double amtOther, IN int exponent, IN int type, IN int timeout, IN BYTE *tags, IN int tagsLen, IN int forceOnline)
- int [emv_activateTransaction](#) (IN int timeout, IN BYTE *tags, IN int tagsLen, IN int forceOnline)
- int [emv_authenticateTransaction](#) (IN BYTE *updatedTLV, IN int updatedTLVLen)
- int [emv_authenticateTransactionWithTimeout](#) (IN int timeout, IN BYTE *updatedTLV, IN int updatedTLVLen)
- int [emv_completeTransaction](#) (IN int commError, IN BYTE *authCode, IN int authCodeLen, IN BYTE *iad, IN int iadLen, IN BYTE *tlvScripts, IN int tlvScriptsLen, IN BYTE *tlv, IN int tlvLen)
- int [emv_cancelTransaction](#) ()
- int [emv_retrieveApplicationData](#) (IN BYTE *AID, IN int AIDLen, OUT BYTE *tlv, IN_OUT int *tlvLen)
- int [emv_setApplicationData](#) (IN BYTE *name, IN int nameLen, IN BYTE *tlv, IN int tlvLen)
- int [emv_setApplicationDataTLV](#) (IN BYTE *tlv, IN int tlvLen)
- int [emv_removeApplicationData](#) (IN BYTE *AID, IN int AIDLen)
- int [emv_removeAllApplicationData](#) ()
- int [emv_retrieveAIDList](#) (OUT BYTE *AIDList, IN_OUT int *AIDListLen)
- int [emv_retrieveTerminalData](#) (OUT BYTE *tlv, IN_OUT int *tlvLen)
- int [emv_setTerminalData](#) (IN BYTE *tlv, IN int tlvLen)
- int [emv_setTerminalMajorConfiguration](#) (IN int configuration)
- int [emv_retrieveCAPK](#) (IN BYTE *capk, IN int capkLen, OUT BYTE *key, IN_OUT int *keyLen)
- int [emv_setCAPK](#) (IN BYTE *capk, IN int capkLen)
- int [emv_removeCAPK](#) (IN BYTE *capk, IN int capkLen)
- int [emv_removeAllCAPK](#) ()
- int [emv_retrieveCAPKList](#) (OUT BYTE *keys, IN_OUT int *keysLen)
- int [emv_retrieveCRL](#) (OUT BYTE *list, IN_OUT int *lssLen)
- int [emv_setCRL](#) (IN BYTE *list, IN int lssLen)
- int [emv_removeCRL](#) (IN BYTE *list, IN int lssLen)
- int [emv_removeAllCRL](#) ()
- int [icc_getICCRReaderStatus](#) (OUT BYTE *status)
- int [icc_powerOnICC](#) (OUT BYTE *ATR, IN_OUT int *inLen)
- int [icc_powerOffICC](#) ()
- int [icc_exchangeAPDU](#) (IN BYTE *c_APDU, IN int cLen, OUT BYTE *reData, IN_OUT int *reLen)
- int [msr_cancelMSRSwipe](#) ()
- int [msr_startMSRSwipe](#) (IN int _timeout)
- void [parseMSRData](#) (IN BYTE *resData, IN int resLen, IN_OUT IDTMSRData *cardData)

11.11.1 Detailed Description

UniPay 1.5 API.

UniPay 1.5 Global API methods.

11.11.2 Macro Definition Documentation

11.11.2.1 IN

```
#define IN
```

INPUT parameter.

11.11.2.2 IN_OUT

```
#define IN_OUT
```

INPUT / OUTPUT PARAMETER.

11.11.2.3 OUT

```
#define OUT
```

OUTPUT parameter.

11.11.3 Typedef Documentation

11.11.3.1 ftpComm_callBack

```
typedef void(* ftpComm_callBack) (int, int, int)
```

Define the comm callback function to get FTP file transfer status

It should be passed as a parameter in a FTP request, Signature (int, int, int) = response code, current block, total blocks
RESPONSE CODES: 100 = FILE DOWNLOAD STARTED 101 = FILE BLOCK XX OF XX RECEIVED 102 = FILE DOWNLOAD COMPLETED 103 = FILE DOWNLOAD TERMINATED PREMATURELY

11.11.3.2 httpComm_callBack

```
typedef void(* httpComm_callBack) (BYTE *, int)
```

Define the comm callback function to get the async url data

It should be registered using the comm_registerHTTPCallback

11.11.3.3 pCMR_callBack

```
typedef void(* pCMR_callBack) (int, IDTCMRData *)
```

Define the camera callback function to get the image data

It should be registered using the device_registerCameraCallBk,

11.11.3.4 pCSFS_callBack

```
typedef void(* pCSFS_callBack) (BYTE status)
```

Define the card status and front switch callback function to get card and front switch status
It should be registered using the device_registerCardStatusFrontSwitchCallBk,

11.11.3.5 pEMV_callBack

```
typedef void(* pEMV_callBack) (int, int, BYTE *, int, IDTTransactionData *, EMV_Callback *,
int)
```

Define the EMV callback function to get the transaction message/data/result.
It should be registered using the emv_registerCallBk,

11.11.3.6 pMessageHotplug

```
typedef void(* pMessageHotplug) (int, int)
```

Define the USB hot-plug callback function to monitor the info when plug in/out the reader.
It should be registered using the registerHotplugCallBk, The first integer parameter is device type, and the second integer parameter is either 0: Device Plugged Out or 1: Device Plugged In

11.11.3.7 pMSR_callBack

```
typedef void(* pMSR_callBack) (int, IDTMSRData)
```

Define the MSR callback function to get the MSR card data
It should be registered using the msr_registerCallBk, this callback function is for backward compatibility

11.11.3.8 pMSR_callBackp

```
typedef void(* pMSR_callBackp) (int, IDTMSRData *)
```

Define the MSR callback function to get pointer to the MSR card data
It should be registered using the msr_registerCallBk, this callback function is recommended instead of pMSR_callBack

11.11.3.9 pPIN_callBack

```
typedef void(* pPIN_callBack) (int, IDTPINData *)
```

Define the PINPad callback function to get the input PIN Pad data
It should be registered using the pin_registerCallBk,

11.11.3.10 pReadDataLog

```
typedef void(* pReadDataLog) (BYTE *, int)
```

Define the read response callback function to monitor the reading response from the reader.
It should be registered using the registerLogCallBk,

11.11.3.11 pSendDataLog

```
typedef void(* pSendDataLog) (BYTE *, int)
```

Define the send command callback function to monitor the sending command into the reader.
It should be registered using the registerLogCallBk,

11.11.3.12 v4Comm_callBack

```
typedef void(* v4Comm_callBack) (BYTE, BYTE, BYTE *, int)
```

Define the comm callback function to receive the V4 Protocol packets received by the device from an external source (IP/USB/RS-232) It should be registered using the comm_registerV4Callback, Data callback will contain command, sub-command, and data from V4 packet

11.11.4 Function Documentation

11.11.4.1 comm_registerHTTPCallback()

```
void comm_registerHTTPCallback (
    httpComm_callBack cBack )
```

Register Comm HTTP Async Callback

Parameters

<i>cBack</i>	- HTTP Comm callback
--------------	----------------------

11.11.4.2 comm_registerV4Callback()

```
void comm_registerV4Callback (
    v4Comm_callBack cBack )
```

Register External V4 Protocol commands Callback

Parameters

<i>cBack</i>	- V4 Protocol Comm callback
--------------	-----------------------------

11.11.4.3 config_getSerialNumber()

```
int config_getSerialNumber (
    OUT char * sNumber )
```

DEPRECATED : please use [config_getSerialNumber_Len\(OUT char* sNumber, IN_OUT int *sNumberLen\)](#)

Polls device for Serial Number

Parameters

<i>sNumber</i>	Returns Serial Number; needs to have at least 64 bytes of memory
----------------	--

Returns

RETURN_CODE: Values can be parsed with `device_getIDGStatusCodeString`

11.11.4.4 config_getSerialNumber_Len()

```
int config_getSerialNumber_Len (
    OUT char * sNumber,
    IN_OUT int * sNumberLen )
```

Polls device for Serial Number

Parameters

<i>sNumber</i>	Returns Serial Number
<i>sNumberLen</i>	Length of Serial Number

Returns

RETURN_CODE: Values can be parsed with `device_getIDGStatusCodeString`

11.11.4.5 device_close()

```
int device_close ( )
```

Close the device

Returns

RETURN_CODE: 0: success, 0x0A: failed

11.11.4.6 device_enablePassThrough()

```
int device_enablePassThrough (
    int enablePassThrough )
```

Enable Pass Through

Enables Pass Through Mode for direct communication with L1 interface (power on icc, send apdu, etc).

Parameters

<i>enablePassThrough</i>	1 = pass through ON, 0 = pass through OFF
--------------------------	---

Returns

RETURN_CODE: Values can be parsed with [device_getIDGStatusCodeString\(\)](#)

11.11.4.7 device_getCurrentDeviceType()

```
int device_getCurrentDeviceType ( )
```

Get current active device type

Returns

: return the device type defined as DEVICE_TYPE in the IDTDef.h

11.11.4.8 device_getFirmwareVersion()

```
int device_getFirmwareVersion (
    OUT char * firmwareVersion )
```

DEPRECATED : please use [device_getFirmwareVersion_Len](#)(OUT char* firmwareVersion, IN_OUT int *firmwareVersionLen)

Polls device for Firmware Version

Parameters

<i>firmwareVersion</i>	Response returned of Firmware Version; needs to have at least 128 bytes of memory
------------------------	---

Returns

RETURN_CODE: Values can be parsed with [device_getIDGStatusCodeString](#)()

11.11.4.9 device_getFirmwareVersion_Len()

```
int device_getFirmwareVersion_Len (
    OUT char * firmwareVersion,
    IN_OUT int * firmwareVersionLen )
```

Polls device for Firmware Version

Parameters

<i>firmwareVersion</i>	Response returned of Firmware Version
<i>firmwareVersionLen</i>	Length of Firmware Version

Returns

RETURN_CODE: Values can be parsed with [device_getIDGStatusCodeString](#)()

11.11.4.10 device_getIDGStatusCodeString()

```
void device_getIDGStatusCodeString (
    IN int returnCode,
    OUT char * description )
```


Review the return code description.

Parameters

<i>returnCode</i>	the response result.
<i>description</i>	

Return values

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

- 0: "no error, beginning task";
- 1: "no response from reader";
- 2: "invalid response data";
- 01: " Incorrect Header Tag";
- 02: " Unknown Command";
- 03: " Unknown Sub-Command";
- 04: " CRC Error in Frame";
- 05: " Incorrect Parameter";
- 06: " Parameter Not Supported";
- 07: " Mal-formatted Data";
- 08: " Timeout";
- 0A: " Failed / NACK";
- 0B: " Command not Allowed";
- 0C: " Sub-Command not Allowed";
- 0D: " Buffer Overflow (Data Length too large for reader buffer)";
- 0E: " User Interface Event";
- 10: " Need clear firmware(apply in boot loader only)";
- 11: " Communication type not supported, VT-1, burst, etc. Need encrypted firmware (apply in boot loader only)";
- 12: " Secure interface is not functional or is in an intermediate state.";
- 13: " Data field is not mod 8";
- 14: " Pad 0x80 not found where expected";
- 15: " Specified key type is invalid";
- 16: " Could not retrieve key from the SAM (InitSecureComm)";
- 17: " Hash code problem";
- 18: " Could not store the key into the SAM (InstallKey)";
- 19: " Frame is too large";

- 1A: " Unit powered up in authentication state but POS must resend the InitSecureComm command";
- 1B: " The EEPROM may not be initialized because SecCommInterface does not make sense";
- 1C: " Problem encoding APDU Module-Specific Status Codes ";
- 20: " Unsupported Index (ILM) SAM Transceiver error - problem communicating with the SAM (Key Mgr)";
- 21: " Unexpected Sequence Counter in multiple frames for single bitmap (ILM)Length error in data returned from the SAM (Key Mgr) - 22: " Improper bit map (ILM)"; - 23: " Request Online Authorization"; - 24: " ViV↔ OCard3 raw data read successful"; - 25: " Message index not available (ILM) ViVOcomm activate transaction card type (ViVOcomm)"; - 26: " Version Information Mismatch (ILM)"; - 27: " Not sending commands in correct index message index (ILM)"; - 28: " Time out or next expected message not received (ILM)"; - 29: " ILM languages not available for viewing (ILM)"; - 2A: " Other language not supported (ILM)"; - 41: " from 41 to 4F, Module-specific errors for Key Manager";
- 50: " Auto-Switch OK";
- 51: " Auto-Switch failed";
- 70: " Antenna Error 80h Use another card";
- 81: " Insert or swipe card";
- 90: " Data encryption Key does not exist";
- 91: " Data encryption Key KSN exhausted";

11.11.4.11 device_getMerchantRecord()

```
int device_getMerchantRecord (
    IN int index,
    OUT BYTE * record )
```

DEPRECATED : please use device_getMerchantRecord_Len(IN int index, OUT BYTE * record, IN_OUT int *recordLen)

Get Merchant Record

Gets the merchant record for the device.

Parameters

<i>index</i>	Merchant Record index, valid values 1-6
<i>record;</i>	needs to have at least 99 bytes of memory response data from reader. Merchant Record Index: 1 byte enabled: 1 byte Merchant ID: 32 bytes Length of Merchant URL: 1 byte Merchant URL: 64 bytes

Returns

success or error code. Values can be parsed with [device_getIDGStatusCodeString\(\)](#)

See also

[ErrorCode](#)

11.11.4.12 device_getMerchantRecord_Len()

```
int device_getMerchantRecord_Len (
    IN int index,
    OUT BYTE * record,
    IN_OUT int * recordLen )
```

Get Merchant Record

Gets the merchant record for the device.

Parameters

<i>index</i>	Merchant Record index, valid values 1-6
<i>record</i>	response data from reader. Merchant Record Index: 1 byte enabled: 1 byte Merchant ID: 32 bytes Length of Merchant URL: 1 byte Merchant URL: 64 bytes
<i>recordLen</i>	Length of record

Returns

success or error code. Values can be parsed with [device_getIDGStatusCodeString\(\)](#)

See also

[ErrorCode](#)

11.11.4.13 device_getSDKWaitTime()

```
int device_getSDKWaitTime ( )
```

Get SDK Wait Time

Get the SDK wait time for transactions

Returns

SDK wait time in seconds

11.11.4.14 device_getThreadStackSize()

```
int device_getThreadStackSize ( )
```

Get Thread Stack Size

Get the stack size setting for newly created threads

Returns

Thread Stack Size

11.11.4.15 device_init()

```
int device_init ( )
```

Initial the device by USB

It will detect the device and trying connect.

The connect status can be checked by [device_isConnected\(\)](#).

Note: after the function returns success, the function [device_setCurrentDevice\(\)](#) has to be called to set the device type.

Returns

RETURN_CODE: Values can be parsed with [device_getIDGStatusCodeString\(\)](#)

11.11.4.16 device_isAttached()

```
int device_isAttached (
    int deviceType )
```

Check if the device is attached to the USB port The function [device_init\(\)](#) must be called before this function.

Parameters

<i>deviceType,the</i>	device type of the USB device
-----------------------	-------------------------------

Returns

1 if the device is attached, or 0 if the device is not attached

11.11.4.17 device_isConnected()

```
int device_isConnected ( )
```

Check the device connected status

Returns

DEVICE_DISCONNECT=0, or DEVICE_CONNECTED = 1

11.11.4.18 device_pingDevice()

```
int device_pingDevice ( )
```

Ping Device

Pings the reader. If connected, returns success. Otherwise, returns timeout.

Returns

RETURN_CODE: Values can be parsed with [device_getIDGStatusCodeString\(\)](#)

11.11.4.19 device_registerCameraCallBk()

```
void device_registerCameraCallBk (
    pCMR_callback pCMRf )
```

To register the camera callback function to get the image data. (Pass NULL to disable the callback.)

11.11.4.20 device_registerCardStatusFrontSwitchCallBk()

```
void device_registerCardStatusFrontSwitchCallBk (
    pCSFS_callback pCSFSf )
```

To register the card status and front switch callback function to get status. (Pass NULL to disable the callback.)

11.11.4.21 device_SendDataCommandNEO()

```
int device_SendDataCommandNEO (
    IN int cmd,
    IN int subCmd,
    IN BYTE * data,
    IN int dataLen,
    OUT BYTE * response,
    IN_OUT int * respLen )
```

Send a Command to device

Sends a command to the device .

Parameters

<i>cmd</i>	buffer of command to execute.
<i>cmdLen,the</i>	length of the buffer cmd.
<i>data</i>	buffer of IDG command data.
<i>dataLen,the</i>	length of the buffer data.
<i>response</i>	Response data
<i>respLen,the</i>	length of Response data

Returns

RETURN_CODE: Values can be parsed with [device_getIDGStatusCodeString\(\)](#) Send a Command to NEO device

Sends a command to the NEO device .

Parameters

<i>cmd</i>	command to execute.
<i>subCmd,sub</i>	command to execute.
<i>data</i>	buffer of NEO command data.
<i>dataLen,the</i>	length of the buffer data.
<i>response</i>	Response data
<i>respLen,the</i>	length of Response data

Returns

RETURN_CODE: Values can be parsed with [device_getIDGStatusCodeString\(\)](#)

11.11.4.22 device_setCurrentDevice()

```
int device_setCurrentDevice (
    int deviceType )
```

Sets the current device to talk to

The connect status can be checked by [device_isConnected\(\)](#).

Parameters

<i>deviceType</i>	Device to connect to
	<pre>enum DEVICE_TYPE { IDT_DEVICE_UNKNOWN=0, IDT_DEVICE_AUGUSTA_HID, IDT_DEVICE_AUGUSTA_KB, IDT_DEVICE_AUGUSTA_S_HID, IDT_DEVICE_AUGUSTA_S_KB, IDT_DEVICE_AUGUSTA_S_TTK_HID, IDT_DEVICE_SPECTRUM_PRO, IDT_DEVICE_MINISMART_II, IDT_DEVICE_L80, IDT_DEVICE_L100, IDT_DEVICE_UNIPAY, IDT_DEVICE_UNIPAY_I_V, IDT_DEVICE_VP3300_AJ, IDT_DEVICE_KIOSK_III, IDT_DEVICE_KIOSK_III_S, IDT_DEVICE_PIP_READER, IDT_DEVICE_VENDI, IDT_DEVICE_VP3300_USB, IDT_DEVICE_UNIPAY_I_V_TTK, IDT_DEVICE_VP3300_BT, IDT_DEVICE_VP8800, IDT_DEVICE_SREDKEY2_HID, IDT_DEVICE_SREDKEY2_KB, IDT_DEVICE_NEO2, IDT_DEVICE_MINISMART_II_COM = IDT_DEVICE_NEO2+5, IDT_DEVICE_SPECTRUM_PRO_COM, IDT_DEVICE_KIOSK_III_COM, IDT_DEVICE_KIOSK_III_S_COM, IDT_DEVICE_PIP_READER_COM, IDT_DEVICE_VP3300_COM, IDT_DEVICE_NEO2_COM, IDT_DEVICE_MAX_DEVICES = IDT_DEVICE_NEO2_COM+5 };</pre>

Returns

RETURN_CODE: 1: success, 0: failed

11.11.4.23 device_setMerchantRecord()

```
int device_setMerchantRecord (
    int index,
    int enabled,
    char * merchantID,
    char * merchantURL )
```

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

Returns

RETURN_CODE: Values can be parsed with [device_getIDGStatusCodeString\(\)](#)

11.11.4.24 device_setSDKWaitTime()

```
void device_setSDKWaitTime (
    int waitTime )
```

Set SDK Wait Time

Set the SDK wait time for transactions

Parameters

<i>waitTime</i>	The SDK wait time for transaction in seconds. The Maximum is 2147483 seconds.
-----------------	---

11.11.4.25 device_setThreadStackSize()

```
void device_setThreadStackSize (
    int threadSize )
```

Set Thread Stack Size

Set the stack size setting for newly created threads

11.11.4.26 emv_activateTransaction()

```
int emv_activateTransaction (
    IN int timeout,
    IN BYTE * tags,
    IN int tagsLen,
    IN int forceOnline )
```

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>	Tags to be included in the request. Passed as a TLV stream. Example, tag 9F0C with amount 0x000000000100 would be 0x9F0C06000000000100
<i>tagsLen</i>	Length of tags
<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 = DFEE1A079F029F36959F37

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString >>>>>IMPORTANT: parameters for amount and amtOther MUST BE PASSED AS A DOUBLE VALUE WITH DECIMAL POINT. Example, do not pass 1, but instead pass 1.0 or 1.00. Otherwise, results will be unpredictable

11.11.4.27 emv_allowFallback()

```
void emv_allowFallback (
    IN int allow )
```

Allow fallback for EMV transactions. Default is TRUE

Parameters

<i>allow</i>	TRUE = allow fallback, FALSE = don't allow fallback
--------------	---

11.11.4.28 emv_authenticateTransaction()

```
int emv_authenticateTransaction (
    IN BYTE * updatedTLV,
    IN int updatedTLVLen )
```

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 = DFEE1A079F029F36959F37
<i>updatedTLVLen</i>	

Returns

RETURN_CODE: Values can be parsed with `device_getResponseCodeString`

11.11.4.29 emv_authenticateTransactionWithTimeout()

```
int emv_authenticateTransactionWithTimeout (
    IN int timeout,
    IN BYTE * updatedTLV,
    IN int updatedTLVLen )
```

Authenticate EMV Transaction Request with Timeout

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>timeout</i>	Timeout value in seconds.
<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 = DFEE1A079F029F36959F37
<i>updatedTLVLen</i>	

Returns

RETURN_CODE: Values can be parsed with `device_getResponseCodeString`

11.11.4.30 `emv_cancelTransaction()`

```
int emv_cancelTransaction ( )
```

Cancel EMV Transaction

Cancels the currently executing EMV transaction.

Returns

RETURN_CODE: Values can be parsed with [device_getResponseCodeString\(\)](#)

11.11.4.31 `emv_completeTransaction()`

```
int emv_completeTransaction (
    IN int commError,
    IN BYTE * authCode,
    IN int authCodeLen,
    IN BYTE * iad,
    IN int iadLen,
    IN BYTE * tlvScripts,
    IN int tlvScriptsLen,
    IN BYTE * tlv,
    IN int tlvLen )
```

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(1) if host was unreachable, or FALSE(0) if host response received. If Communication error, authCode, iad, tlvScripts can be null.
------------------	--

Parameters

<i>authCode</i>	Authorization code from host. Two bytes. Example 0x3030. (Tag value 8A). Required
<i>authCodeLen</i>	the length of authCode
<i>iad</i>	Issuer Authentication Data, if any. Example 0x11223344556677883030 (tag value 91).
<i>iadLen</i>	the length of iadLen
<i>tlvScripts</i>	71/72 scripts, if any
<i>tlvScriptsLen</i>	the length of tlvScriptsLen
<i>tlv</i>	Additional TLV data to return with transaction results (if any)
<i>tlvLen</i>	the length of tlv

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString

11.11.4.32 emv_getAutoAuthenticateTransaction()

```
int emv_getAutoAuthenticateTransaction ( )
```

Gets auto authenticate value for EMV transactions.

Returns

RETURN_CODE: TRUE = auto authenticate, FALSE = manually authenticate

11.11.4.33 emv_getAutoCompleteTransaction()

```
int emv_getAutoCompleteTransaction ( )
```

Gets auto complete value for EMV transactions.

Returns

RETURN_CODE: TRUE = auto complete, FALSE = manually complete

11.11.4.34 emv_registerCallBk()

```
void emv_registerCallBk (
    pEMV_callback pEMVf )
```

To register the emv callback function to get the EMV processing response. (Pass NULL to disable the callback.)

11.11.4.35 emv_removeAllApplicationData()

```
int emv_removeAllApplicationData ( )
```

Remove All Application Data

Removes all the Application Data

Returns

RETURN_CODE: Values can be parsed with [device_getIDGStatusCodeString\(\)](#)

11.11.4.36 emv_removeAllCAPK()

```
int emv_removeAllCAPK ( )
```

Remove All Certificate Authority Public Key

Removes all the CAPK

Returns

RETURN_CODE: Values can be parsed with [device_getIDGStatusCodeString\(\)](#)

11.11.4.37 emv_removeAllCRL()

```
int emv_removeAllCRL ( )
```

Remove All Certificate Revocation List Entries

Removes all CRLentry entries

Returns

RETURN_CODE: Values can be parsed with [device_getResponseCodeString\(\)](#)

11.11.4.38 emv_removeApplicationData()

```
int emv_removeApplicationData (
    IN BYTE * AID,
    IN int AIDLen )
```

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>AIDLen</i>	the length of AID data buffer

Returns

RETURN_CODE: Values can be parsed with [device_getIDGStatusCodeString\(\)](#)

11.11.4.39 emv_removeCAPK()

```
int emv_removeCAPK (
    IN BYTE * capk,
```

```
IN int capkLen )
```

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>capkLen</i>	the length of capk data buffer

Returns

RETURN_CODE: Values can be parsed with [device_getIDGStatusCodeString\(\)](#)

11.11.4.40 emv_removeCRL()

```
int emv_removeCRL (
    IN BYTE * list,
    IN int lsLen )
```

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>lsLen</i>	the length of list data buffer

Returns

RETURN_CODE: Values can be parsed with [device_getResponseCodeString\(\)](#)

11.11.4.41 emv_retrieveAIDList()

```
int emv_retrieveAIDList (
    OUT BYTE * AIDList,
    IN_OUT int * AIDListLen )
```

Retrieve AID list

Returns all the AID names installed on the terminal for CTLS. .

Parameters

<i>AIDList</i>	array of AID name byte arrays
<i>AIDListLen</i>	the length of AIDList array buffer

Returns

RETURN_CODE: Values can be parsed with [device_getIDGStatusCodeString\(\)](#)

11.11.4.42 emv_retrieveApplicationData()

```
int emv_retrieveApplicationData (
    IN BYTE * AID,
    IN int AIDLen,
    OUT BYTE * tlv,
    IN_OUT int * tlvLen )
```

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>AIDLen</i>	the length of AID data buffer.
<i>tlv</i>	The TLV elements of the requested AID
<i>tlvLen</i>	the length of tlv data buffer.

Returns

RETURN_CODE: Values can be parsed with [device_getIDGStatusCodeString\(\)](#)

11.11.4.43 emv_retrieveCAPK()

```
int emv_retrieveCAPK (
    IN BYTE * capk,
    IN int capkLen,
    OUT BYTE * key,
    IN_OUT int * keyLen )
```

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>capkLen</i>	the length of capk data buffer

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>keyLen</i>	<p>the length of key data buffer</p> <ul style="list-style-type: none"> •

Returns

RETURN_CODE: Values can be parsed with [device_getIDGStatusCodeString\(\)](#)

11.11.4.44 emv_retrieveCAPKList()

```
int emv_retrieveCAPKList (
    OUT BYTE * keys,
    IN_OUT int * keysLen )
```

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>keysLen</i>	the length of keys data buffer

Returns

RETURN_CODE: Values can be parsed with [device_getIDGStatusCodeString\(\)](#)

11.11.4.45 emv_retrieveCRL()

```
int emv_retrieveCRL (
    OUT BYTE * list,
    IN_OUT int * lssLen )
```

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>lssLen</i>	the length of list data buffer

Returns

RETURN_CODE: Values can be parsed with [device_getResponseCodeString\(\)](#)

11.11.4.46 emv_retrieveTerminalData()

```
int emv_retrieveTerminalData (
    OUT BYTE * tlv,
    IN_OUT int * tlvLen )
```

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>tlvLen</i>	the length of tlv data buffer

Returns

RETURN_CODE: Values can be parsed with [device_getIDGStatusCodeString\(\)](#)

11.11.4.47 emv_setApplicationData()

```
int emv_setApplicationData (
    IN BYTE * name,
    IN int nameLen,
    IN BYTE * tlv,
    IN int tlvLen )
```

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>nameLen</i>	the length of name data buffer of Application name,
<i>tlv</i>	Application data in TLV format
<i>tlvLen</i>	the length of tlv data buffer

Returns

RETURN_CODE: Values can be parsed with [device_getResponseCodeString\(\)](#)

11.11.4.48 emv_setApplicationDataTLV()

```
int emv_setApplicationDataTLV (
    IN BYTE * tlv,
    IN int tlvLen )
```

Set Application Data by TLV

Sets the Application Data as specified by the TLV data

Parameters

<i>tlv</i>	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) Example valid TLV, for Group #2, AID a0000000035010: "dfee2d01029f0607a0000000051010ffe10101ffe50110ffe30114ffe20106"
<i>tlvLen</i>	the length of tlv data buffer

Returns

RETURN_CODE: Values can be parsed with [device_getResponseCodeString\(\)](#)

11.11.4.49 emv_setAutoAuthenticateTransaction()

```
void emv_setAutoAuthenticateTransaction (
    IN int authenticate )
```

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

Parameters

<i>authenticate</i>	TRUE = auto authenticate, FALSE = manually authenticate
---------------------	---

11.11.4.50 emv_setAutoCompleteTransaction()

```
void emv_setAutoCompleteTransaction (
    IN int complete )
```

Enables complete for EMV transactions. If a `emv_authenticateTransaction` results in code 0x0004 (go online), then `emv_completeTransaction` can automatically execute if parameter is set to TRUE

Parameters

<i>complete</i>	TRUE = auto complete, FALSE = manually complete
-----------------	---

11.11.4.51 emv_setCAPK()

```
int emv_setCAPK (
    IN BYTE * capk,
    IN int capkLen )
```

Set Certificate Authority Public Key

Sets the CAPK as specified by the CAKey structure

Parameters

<i>capk</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>capkLen</i>	the length of capk data buffer

Returns

RETURN_CODE: Values can be parsed with [device_getIDGStatusCodeString\(\)](#)

11.11.4.52 emv_setCRL()

```
int emv_setCRL (
    IN BYTE * list,
    IN int lsLen )
```

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>lsLen</i>	the length of list data buffer

Returns

RETURN_CODE: Values can be parsed with [device_getResponseCodeString\(\)](#)

11.11.4.53 emv_setTerminalData()

```
int emv_setTerminalData (
    IN BYTE * tlv,
    IN int tlvLen )
```

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 file
<i>tlvLen</i>	the length of tlv data buffer

Return values

<i>RETURN_CODE</i>	Values can be parsed with device_getIDGStatusCodeString()
--------------------	---

11.11.4.54 emv_setTerminalMajorConfiguration()

```
int emv_setTerminalMajorConfiguration (
    IN int configuration )
```

Sets the terminal major configuration in ICS .

Parameters

<i>configuration</i>	A configuration value, range 1-23 <ul style="list-style-type: none">• 1 = 1C• 2 = 2C• 3 = 3C• 4 = 4C• 5 = 5C ...• 23 = 23C
----------------------	---

Returns

RETURN_CODE: Values can be parsed with [device_getResponseCodeString\(\)](#)

11.11.4.55 emv_startTransaction()

```
int emv_startTransaction (
    IN double amount,
```

```

    IN double amtOther,
    IN int exponent,
    IN int type,
    IN int timeout,
    IN BYTE * tags,
    IN int tagsLen,
    IN int forceOnline )

```

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) - SEE IMPORTANT NOTE BELOW
<i>amtOther</i>	Other amount value, if any (tag value 9F03) - SEE IMPORTANT NOTE BELOW
<i>exponent</i>	Number of characters after decimal 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 stream. Example, tag 9F0C with amount 0x000000000100 would be 0x9F0C06000000000100 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>tagsLen</i>	Length of tags
<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 = DFEE1A079F029F36959F37

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString >>>>IMPORTANT: parameters for amount and amtOther MUST BE PASSED AS A DOUBLE VALUE WITH DECIMAL POINT. Example, do not pass 1, but instead pass 1.0 or 1.00. Otherwise, results will be unpredictable

11.11.4.56 icc_exchangeAPDU()

```

int icc_exchangeAPDU (
    IN BYTE * c_APDU,
    IN int cLen,
    OUT BYTE * reData,
    IN_OUT int * reLen )

```

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>cLen</i>	APDU data packet length
<i>reData</i>	Unencrypted APDU response
<i>reLen</i>	Unencrypted APDU response data length

Returns

RETURN_CODE: Values can be parsed with `device_getIDGStatusCodeString`

11.11.4.57 `icc_getICCReaderStatus()`

```
int icc_getICCReaderStatus (
    OUT BYTE * status )
```

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
---------------	--

Returns

RETURN_CODE: Values can be parsed with `device_getIDGStatusCodeString`

11.11.4.58 `icc_powerOffICC()`

```
int icc_powerOffICC ( )
```

Power Off ICC

Powers down the ICC

Returns

RETURN_CODE: Values can be parsed with `device_getIDGStatusCodeString()`

If Success, empty If Failure, ASCII encoded data of error string

11.11.4.59 `icc_powerOnICC()`

```
int icc_powerOnICC (
    OUT BYTE * ATR,
    IN_OUT int * inLen )
```

Power On ICC

Power up the currently selected microprocessor card in the ICC reader

Parameters

<i>ATR,the</i>	ATR data response when succeeded power on ICC,
<i>inLen,the</i>	length of ATR data

Returns

RETURN_CODE: Values can be parsed with `device_getIDGStatusCodeString()`

11.11.4.60 msr_cancelMSRSwipe()

```
int msr_cancelMSRSwipe ( )
```

Disable MSR Swipe Cancels MSR swipe request.

Returns

RETURN_CODE: Values can be parsed with [device_getIDGStatusCodeString\(\)](#)

11.11.4.61 msr_registerCallBk()

```
void msr_registerCallBk (
    pMSR_callback pMSRf )
```

To register the msr callback function to get the MSR card data. (Pass NULL to disable the callback.)

11.11.4.62 msr_registerCallBkp()

```
void msr_registerCallBkp (
    pMSR_callbackp pMSRf )
```

To register the msr callback function to get the MSR card data pointer. (Pass NULL to disable the callback.)

11.11.4.63 msr_startMSRSwipe()

```
int msr_startMSRSwipe (
    IN int _timeout )
```

Start 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
----------------	---------------------

Returns

RETURN_CODE: Values can be parsed with [device_getIDGStatusCodeString\(\)](#) Note: if auto poll mode is on, it will return command not allowed error

11.11.4.64 parseMSRData()

```
void parseMSRData (
    IN BYTE * resData,
    IN int resLen,
    IN_OUT IDTMSRData * cardData )
```

Parser the MSR data from the buffer into IDTMSTData structure

Parameters

<i>resData</i>	MSR card data buffer
<i>resLen</i>	the length of resData
<i>cardData</i>	the parser result with IDTMSTData structure

11.11.4.65 pin_registerCallBk()

```
void pin_registerCallBk (
    pPIN_callback pPINf )
```

To register the pin callback function to get the PINPad data. (Pass NULL to disable the callback.)

11.11.4.66 registerHotplugCallBk()

```
void registerHotplugCallBk (
    pMessageHotplug pMsgHotplug )
```

To register the USB HID hot-plug callback function which implemented in the application to monitor the hotplug message from the SDK.

11.11.4.67 registerLogCallBk()

```
void registerLogCallBk (
    pSendDataLog pFSend,
    pReadDataLog pFRead )
```

To register the log callback function which implemented in the application to monitor sending/reading data between application and reader.

11.11.4.68 SDK_Version()

```
char* SDK_Version ( )
```

To Get SDK version

Returns

return the SDK version string

11.11.4.69 setAbsoluteLibraryPath()

```
int setAbsoluteLibraryPath (
    const char * absoluteLibraryPath )
```

Set the path to use when searching for ID TECH's libraries. If this is not set, the libraries will be searched for with the system's default procedures.

Parameters

<i>absoluteLibraryPath</i>	The absolute path to ID TECH's libraries.
----------------------------	---

Returns

RETURN_CODE: Values can be parsed with [device_getIDGStatusCodeString\(\)](#)

11.12 Source_C/libIDT_Vendi.h File Reference

Vendi API.

```
#include "IDTDef.h"
```

Macros

- [#define IN](#)
- [#define OUT](#)
- [#define IN_OUT](#)

Typedefs

- typedef void(* [pMessageHotplug](#)) (int, int)
- typedef void(* [pSendDataLog](#)) (BYTE *, int)
- typedef void(* [pReadDataLog](#)) (BYTE *, int)
- typedef void(* [pEMV_callback](#)) (int, int, BYTE *, int, IDTTransactionData *, EMV_Callback *, int)
- typedef void(* [pMSR_callback](#)) (int, IDTMSRData)
- typedef void(* [pMSR_callbackp](#)) (int, IDTMSRData *)
- typedef void(* [pPIN_callback](#)) (int, IDTPINData *)
- typedef void(* [pCMR_callback](#)) (int, IDTCMRData *)
- typedef void(* [pCSFS_callback](#)) (BYTE status)
- typedef void(* [ftpComm_callback](#)) (int, int, int)
- typedef void(* [httpComm_callback](#)) (BYTE *, int)
- typedef void(* [v4Comm_callback](#)) (BYTE, BYTE, BYTE *, int)

Functions

- void [registerHotplugCallBk](#) ([pMessageHotplug](#) pMsgHotplug)
- void [registerLogCallBk](#) ([pSendDataLog](#) pFSend, [pReadDataLog](#) pFRead)
- void [emv_registerCallBk](#) ([pEMV_callback](#) pEMVf)
- void [msr_registerCallBk](#) ([pMSR_callback](#) pMSRf)
- void [msr_registerCallBkp](#) ([pMSR_callbackp](#) pMSRf)
- void [ctls_registerCallBk](#) ([pMSR_callback](#) pCTLSf)
- void [ctls_registerCallBkp](#) ([pMSR_callbackp](#) pCTLSf)
- void [pin_registerCallBk](#) ([pPIN_callback](#) pPINf)
- void [device_registerCameraCallBk](#) ([pCMR_callback](#) pCMRf)
- void [device_registerCardStatusFrontSwitchCallBk](#) ([pCSFS_callback](#) pCSFSf)
- void [comm_registerHTTPCallback](#) ([httpComm_callback](#) cBack)
- void [comm_registerV4Callback](#) ([v4Comm_callback](#) cBack)
- char * [SDK_Version](#) ()
- int [setAbsoluteLibraryPath](#) (const char *absoluteLibraryPath)
- int [device_init](#) ()
- int [device_setCurrentDevice](#) (int deviceType)
- int [device_isAttached](#) (int deviceType)
- int [device_close](#) ()
- void [device_getIDGStatusCodeString](#) ([IN](#) int returnCode, [OUT](#) char *despcriton)

- int [device_isConnected](#) ()
- int [device_getFirmwareVersion](#) (OUT char *firmwareVersion)
- int [device_getFirmwareVersion_Len](#) (OUT char *firmwareVersion, IN_OUT int *firmwareVersionLen)
- int [device_pingDevice](#) ()
- int [device_controlUserInterface](#) (IN BYTE *values)
- int [device_getCurrentDeviceType](#) ()
- int [device_SendDataCommandNEO](#) (IN int cmd, IN int subCmd, IN BYTE *data, IN int dataLen, OUT BYTE *response, IN_OUT int *respLen)
- int [device_enablePassThrough](#) (int enablePassThrough)
- int [device_setBurstMode](#) (IN BYTE mode)
- int [device_setPollMode](#) (IN BYTE mode)
- int [device_getSDKWaitTime](#) ()
- void [device_setSDKWaitTime](#) (int waitTime)
- int [device_getThreadStackSize](#) ()
- void [device_setThreadStackSize](#) (int threadSize)
- int [device_setMerchantRecord](#) (int index, int enabled, char *merchantID, char *merchantURL)
- int [device_getMerchantRecord](#) (IN int index, OUT BYTE *record)
- int [device_getMerchantRecord_Len](#) (IN int index, OUT BYTE *record, IN_OUT int *recordLen)
- int [device_getTransactionResults](#) (IDTMSRData *cardData)
- int [config_getSerialNumber](#) (OUT char *sNumber)
- int [config_getSerialNumber_Len](#) (OUT char *sNumber, IN_OUT int *sNumberLen)
- int [ctls_startTransaction](#) (IN double amount, IN double amtOther, IN int type, IN const int _timeout, IN BYTE *tags, IN int tagsLen)
- int [ctls_activateTransaction](#) (IN const int _timeout, IN BYTE *tags, IN int tagsLen)
- int [ctls_cancelTransaction](#) ()
- int [ctls_retrieveApplicationData](#) (IN BYTE *AID, IN int AIDLen, OUT BYTE *tlv, IN_OUT int *tlvLen)
- int [ctls_setApplicationData](#) (IN BYTE *tlv, IN int tlvLen)
- int [ctls_removeApplicationData](#) (IN BYTE *AID, IN int AIDLen)
- int [ctls_removeAllApplicationData](#) ()
- int [ctls_retrieveAIDList](#) (OUT BYTE *AIDList, IN_OUT int *AIDListLen)
- int [ctls_retrieveTerminalData](#) (OUT BYTE *tlv, IN_OUT int *tlvLen)
- int [ctls_setTerminalData](#) (IN BYTE *tlv, IN int tlvLen)
- int [ctls_retrieveCAPK](#) (IN BYTE *capk, IN int capkLen, OUT BYTE *key, IN_OUT int *keyLen)
- int [ctls_setCAPK](#) (IN BYTE *capk, IN int capkLen)
- int [ctls_removeCAPK](#) (IN BYTE *capk, IN int capkLen)
- int [ctls_removeAllCAPK](#) ()
- int [ctls_retrieveCAPKList](#) (OUT BYTE *keys, IN_OUT int *keysLen)
- int [ctls_setConfigurationGroup](#) (IN BYTE *tlv, IN int tlvLen)
- int [ctls_getConfigurationGroup](#) (IN int group, OUT BYTE *tlv, OUT int *tlvLen)
- int [ctls_getAllConfigurationGroups](#) (OUT BYTE *tlv, IN_OUT int *tlvLen)
- int [ctls_removeConfigurationGroup](#) (int group)
- int [msr_cancelMSRSwipe](#) ()
- int [msr_startMSRSwipe](#) (IN int _timeout)
- void [parseMSRData](#) (IN BYTE *resData, IN int resLen, IN_OUT IDTMSRData *cardData)

11.12.1 Detailed Description

Vendi API.

Vendi Global API methods.

11.12.2 Macro Definition Documentation

11.12.2.1 IN

```
#define IN
```

INPUT parameter.

11.12.2.2 IN_OUT

```
#define IN_OUT
```

INPUT / OUTPUT PARAMETER.

11.12.2.3 OUT

```
#define OUT
```

OUTPUT parameter.

11.12.3 Typedef Documentation**11.12.3.1 ftpComm_callBack**

```
typedef void(* ftpComm_callBack) (int, int, int)
```

Define the comm callback function to get FTP file transfer status

It should be passed as a parameter in a FTP request, Signature (int, int, int) = response code, current block, total blocks
 RESPONSE CODES: 100 = FILE DOWNLOAD STARTED 101 = FILE BLOCK XX OF XX RECEIVED 102 = FILE DOWNLOAD COMPLETED 103 = FILE DOWNLOAD TERMINATED PREMATURELY

11.12.3.2 httpComm_callBack

```
typedef void(* httpComm_callBack) (BYTE *, int)
```

Define the comm callback function to get the async url data

It should be registered using the comm_registerHTTPCallback

11.12.3.3 pCMR_callBack

```
typedef void(* pCMR_callBack) (int, IDTCMRData *)
```

Define the camera callback function to get the image data

It should be registered using the device_registerCameraCallBk,

11.12.3.4 pCSFS_callBack

```
typedef void(* pCSFS_callBack) (BYTE status)
```

Define the card status and front switch callback function to get card and front switch status

It should be registered using the device_registerCardStatusFrontSwitchCallBk,

11.12.3.5 pEMV_callBack

```
typedef void(* pEMV_callBack) (int, int, BYTE *, int, IDTTransactionData *, EMV_Callback *, int)
```

Define the EMV callback function to get the transaction message/data/result.
It should be registered using the `emv_registerCallBk`,

11.12.3.6 pMessageHotplug

```
typedef void(* pMessageHotplug) (int, int)
```

Define the USB hot-plug callback function to monitor the info when plug in/out the reader.
It should be registered using the `registerHotplugCallBk`, The first integer parameter is device type, and the second integer parameter is either 0: Device Plugged Out or 1: Device Plugged In

11.12.3.7 pMSR_callBack

```
typedef void(* pMSR_callBack) (int, IDTMSRData)
```

Define the MSR callback function to get the MSR card data
It should be registered using the `msr_registerCallBk`, this callback function is for backward compatibility

11.12.3.8 pMSR_callBackp

```
typedef void(* pMSR_callBackp) (int, IDTMSRData *)
```

Define the MSR callback function to get pointer to the MSR card data
It should be registered using the `msr_registerCallBk`, this callback function is recommended instead of `pMSR_callBack`

11.12.3.9 pPIN_callBack

```
typedef void(* pPIN_callBack) (int, IDTPINData *)
```

Define the PINPad callback function to get the input PIN Pad data
It should be registered using the `pin_registerCallBk`,

11.12.3.10 pReadDataLog

```
typedef void(* pReadDataLog) (BYTE *, int)
```

Define the read response callback function to monitor the reading response from the reader.
It should be registered using the `registerLogCallBk`,

11.12.3.11 pSendDataLog

```
typedef void(* pSendDataLog) (BYTE *, int)
```

Define the send command callback function to monitor the sending command into the reader.
It should be registered using the `registerLogCallBk`,

11.12.3.12 v4Comm_callBack

```
typedef void(* v4Comm_callBack) (BYTE, BYTE, BYTE *, int)
```

Define the comm callback function to receive the V4 Protocol packets received by the device from an external source (IP/USB/RS-232) It should be registered using the `comm_registerV4Callback`, Data callback will contain command, sub-command, and data from V4 packet

11.12.4 Function Documentation

11.12.4.1 comm_registerHTTPCallback()

```
void comm_registerHTTPCallback (
    httpComm_callBack cBack )
```

Register Comm HTTP Async Callback

Parameters

<i>cBack</i>	HTTP Comm callback
--------------	--------------------

11.12.4.2 comm_registerV4Callback()

```
void comm_registerV4Callback (
    v4Comm_callBack cBack )
```

Register External V4 Protocol commands Callback

Parameters

<i>cBack</i>	V4 Protocol Comm callback
--------------	---------------------------

11.12.4.3 config_getSerialNumber()

```
int config_getSerialNumber (
    OUT char * sNumber )
```

DEPRECATED : please use [config_getSerialNumber_Len](#)(OUT char* sNumber, IN_OUT int *sNumberLen)

Polls device for Serial Number

Parameters

<i>sNumber</i>	Returns Serial Number; needs to have at least 64 bytes of memory
----------------	--

Returns

RETURN_CODE: Values can be parsed with [device_getIDGStatusCodeString](#)

11.12.4.4 config_getSerialNumber_Len()

```
int config_getSerialNumber_Len (
    OUT char * sNumber,
    IN_OUT int * sNumberLen )
```

Polls device for Serial Number

Parameters

<i>sNumber</i>	Returns Serial Number
<i>sNumberLen</i>	Length of Serial Number

Returns

RETURN_CODE: Values can be parsed with `device_getIDGStatusCodeString`

11.12.4.5 `ctls_activateTransaction()`

```
int ctls_activateTransaction (
    IN const int _timeout,
    IN BYTE * tags,
    IN int tagsLen )
```

Start CTLS Transaction Request

Authorizes the CTLS 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 stream. Example, tag 9F0C with amount 0x0000000000100 would be 0x9F0C060000000000100
<i>tagsLen</i>	The length of tags data buffer.

>>>>>IMPORTANT: parameters for amount and amtOther MUST BE PASSED AS A DOUBLE VALUE WITH DECIMAL POINT. Example, do not pass 1, but instead pass 1.0 or 1.00. Otherwise, results will be unpredictable

Returns

RETURN_CODE: Values can be parsed with [device_getIDGStatusCodeString\(\)](#) Note: if auto poll is on, it will return the error IDG_P2_STATUS_CODE_COMMAND_NOT_ALLOWED

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

11.12.4.6 `ctls_cancelTransaction()`

```
int ctls_cancelTransaction ( )
```

Cancel EMV Transaction

Cancels the currently executing EMV transaction.

Returns

RETURN_CODE: Values can be parsed with [device_getIDGStatusCodeString\(\)](#)

11.12.4.7 `ctls_getAllConfigurationGroups()`

```
int ctls_getAllConfigurationGroups (
    OUT BYTE * tlv,
    IN_OUT int * tlvLen )
```

Retrieve All Configuration Groups

Returns all the Configuration Groups installed on the terminal for CTLS

Parameters

<i>tlv</i>	The TLV elements data
<i>tlvLen</i>	the length of tlv data buffer.

Returns

RETURN_CODE: Values can be parsed with [device_getIDGStatusCodeString\(\)](#)

11.12.4.8 ctls_getConfigurationGroup()

```
int ctls_getConfigurationGroup (
    IN int group,
    OUT BYTE * tlv,
    OUT int * tlvLen )
```

Get Configuration Group

Retrieves the Configuration for the specified Group.

Parameters

<i>group</i>	Configuration Group
<i>tlv</i>	return data
<i>tlvLen</i>	the length of tlv data buffer

Returns

RETURN_CODE: Values can be parsed with [device_getIDGStatusCodeString\(\)](#)

11.12.4.9 ctls_registerCallBk()

```
void ctls_registerCallBk (
    pMSR_callBack pCTLSf )
```

To register the ctls callback function to get the MSR card data. (Pass NULL to disable the callback.)

11.12.4.10 ctls_registerCallBkp()

```
void ctls_registerCallBkp (
    pMSR_callBackp pCTLSf )
```

To register the ctls callback function to get the MSR card data pointer. (Pass NULL to disable the callback.)

11.12.4.11 ctls_removeAllApplicationData()

```
int ctls_removeAllApplicationData ( )
```

Remove All Application Data

Removes all the Application Data

Returns

RETURN_CODE: Values can be parsed with [device_getIDGStatusCodeString\(\)](#)

11.12.4.12 ctls_removeAllCAPK()

```
int ctls_removeAllCAPK ( )
```

Remove All Certificate Authority Public Key

Removes all the CAPK

Returns

RETURN_CODE: Values can be parsed with [device_getIDGStatusCodeString\(\)](#)

11.12.4.13 ctls_removeApplicationData()

```
int ctls_removeApplicationData (
    IN BYTE * AID,
    IN int AIDLen )
```

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>AIDLen</i>	the length of AID data buffer

Returns

RETURN_CODE: Values can be parsed with [device_getIDGStatusCodeString\(\)](#)

11.12.4.14 ctls_removeCAPK()

```
int ctls_removeCAPK (
    IN BYTE * capk,
    IN int capkLen )
```

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>capkLen</i>	the length of capk data buffer

Returns

RETURN_CODE: Values can be parsed with [device_getIDGStatusCodeString\(\)](#)

11.12.4.15 ctls_removeConfigurationGroup()

```
int ctls_removeConfigurationGroup (
    int group )
```

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>	Values can be parsed with device_getIDGStatusCodeString()
--------------------	---

11.12.4.16 ctls_retrieveAIDList()

```
int ctls_retrieveAIDList (
    OUT BYTE * AIDList,
    IN_OUT int * AIDListLen )
```

Retrieve AID list

Returns all the AID names installed on the terminal for CTLS. .

Parameters

<i>AIDList</i>	array of AID name byte arrays
<i>AIDListLen</i>	the length of AIDList array buffer

Returns

RETURN_CODE: Values can be parsed with [device_getIDGStatusCodeString\(\)](#)

11.12.4.17 ctls_retrieveApplicationData()

```
int ctls_retrieveApplicationData (
    IN BYTE * AID,
    IN int AIDLen,
    OUT BYTE * tlv,
    IN_OUT int * tlvLen )
```

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>AIDLen</i>	the length of AID data buffer.
<i>tlv</i>	The TLV elements of the requested AID

Parameters

<i>tlvLen</i>	the length of tlv data buffer.
---------------	--------------------------------

Returns

RETURN_CODE: Values can be parsed with [device_getIDGStatusCodeString\(\)](#)

11.12.4.18 `ctls_retrieveCAPK()`

```
int ctls_retrieveCAPK (
    IN BYTE * capk,
    IN int capkLen,
    OUT BYTE * key,
    IN_OUT int * keyLen )
```

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>capkLen</i>	the length of capk data buffer
<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>keyLen</i>	<p>the length of key data buffer</p> <ul style="list-style-type: none"> •

Returns

RETURN_CODE: Values can be parsed with [device_getIDGStatusCodeString\(\)](#)

11.12.4.19 `ctls_retrieveCAPKList()`

```
int ctls_retrieveCAPKList (
```

```

    OUT BYTE * keys,
    IN_OUT int * keysLen )

```

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>keysLen</i>	the length of keys data buffer

Returns

RETURN_CODE: Values can be parsed with [device_getIDGStatusCodeString\(\)](#)

11.12.4.20 ctls_retrieveTerminalData()

```

int ctls_retrieveTerminalData (
    OUT BYTE * tlv,
    IN_OUT int * tlvLen )

```

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>tlvLen</i>	the length of tlv data buffer

Returns

RETURN_CODE: Values can be parsed with [device_getIDGStatusCodeString\(\)](#)

11.12.4.21 ctls_setApplicationData()

```

int ctls_setApplicationData (
    IN BYTE * tlv,
    IN int tlvLen )

```

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>tlvLen</i>	the length of tlv data buffer
---------------	-------------------------------

Returns

RETURN_CODE: Values can be parsed with [device_getIDGStatusCodeString\(\)](#)

11.12.4.22 ctls_setCAPK()

```
int ctls_setCAPK (
    IN BYTE * capk,
    IN int capkLen )
```

Set Certificate Authority Public Key

Sets the CAPK as specified by the CAKey structure

Parameters

<i>capk</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>capkLen</i>	the length of capk data buffer

Returns

RETURN_CODE: Values can be parsed with [device_getIDGStatusCodeString\(\)](#)

11.12.4.23 ctls_setConfigurationGroup()

```
int ctls_setConfigurationGroup (
    IN BYTE * tlv,
    IN int tlvLen )
```

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>tlvLen</i>	the length of tlv data buffer

Returns

RETURN_CODE: Values can be parsed with [device_getIDGStatusCodeString\(\)](#)

11.12.4.24 ctls_setTerminalData()

```
int ctls_setTerminalData (
    IN BYTE * tlv,
    IN int tlvLen )
```

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 [ctls_getConfigurationGroup\(int group\)](#), and deleted with [ctls_removeConfigurationGroup\(int group\)](#). You cannot delete group 0.

Parameters

<i>tlv</i>	TerminalData configuration file
<i>tlvLen</i>	the length of tlv data buffer

Return values

RETURN_CODE	Values can be parsed with device_getIDGStatusCodeString()
--------------------	---

11.12.4.25 ctls_startTransaction()

```
int ctls_startTransaction (
    IN double amount,
    IN double amtOther,
    IN int type,
    IN const int _timeout,
    IN BYTE * tags,
    IN int tagsLen )
```

Start CTLS Transaction Request

Authorizes the CTLS transaction for an ICC card

The tags will be returned in the callback routine.

Parameters

<i>amount</i>	Transaction amount value (tag value 9F02) - SEE IMPORTANT NOTE BELOW
<i>amtOther</i>	Other amount value, if any (tag value 9F03) - SEE IMPORTANT NOTE BELOW

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 TLV stream. Example, tag 9F0C with amount 0x000000000100 would be 0x9F0C06000000000100 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>tagsLen</i>	The length of tags data buffer.

>>>>>IMPORTANT: parameters for amount and amtOther MUST BE PASSED AS A DOUBLE VALUE WITH DECIMAL POINT. Example, do not pass 1, but instead pass 1.0 or 1.00. Otherwise, results will be unpredictable

Returns

RETURN_CODE: Values can be parsed with [device_getIDGStatusCodeString\(\)](#) Note: if auto poll is on, it will return the error IDG_P2_STATUS_CODE_COMMAND_NOT_ALLOWED

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 FFEE06189F220201009F26040000000009F2B050100000000DF010101 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

11.12.4.26 device_close()

```
int device_close ( )
```

Close the device

Returns

RETURN_CODE: 0: success, 0x0A: failed

11.12.4.27 device_controlUserInterface()

```
int device_controlUserInterface (
    IN BYTE * values )
```

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) • 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
---------------	--

Returns

RETURN_CODE: Values can be parsed with [device_getIDGStatusCodeString\(\)](#)

11.12.4.28 device_enablePassThrough()

```
int device_enablePassThrough (
    int enablePassThrough )
```

Enable Pass Through

Enables Pass Through Mode for direct communication with L1 interface (power on icc, send apdu, etc).

Parameters

<i>enablePassThrough</i>	1 = pass through ON, 0 = pass through OFF
--------------------------	---

Returns

RETURN_CODE: Values can be parsed with [device_getIDGStatusCodeString\(\)](#)

11.12.4.29 device_getCurrentDeviceType()

```
int device_getCurrentDeviceType ( )
```

Get current active device type

Returns

: return the device type defined as DEVICE_TYPE in the IDTDef.h

11.12.4.30 device_getFirmwareVersion()

```
int device_getFirmwareVersion (
    OUT char * firmwareVersion )
```

DEPRECATED : please use [device_getFirmwareVersion_Len\(OUT char* firmwareVersion, IN_OUT int *firmwareVersionLen\)](#)

Polls device for Firmware Version

Parameters

<i>firmwareVersion</i>	Response returned of Firmware Version; needs to have at least 128 bytes of memory
------------------------	---

Returns

RETURN_CODE: Values can be parsed with [device_getIDGStatusCodeString\(\)](#)

11.12.4.31 device_getFirmwareVersion_Len()

```
int device_getFirmwareVersion_Len (
    OUT char * firmwareVersion,
    IN_OUT int * firmwareVersionLen )
```

Polls device for Firmware Version

Parameters

<i>firmwareVersion</i>	Response returned of Firmware Version
<i>firmwareVersionLen</i>	Length of Firmware Version

Returns

RETURN_CODE: Values can be parsed with [device_getIDGStatusCodeString\(\)](#)

11.12.4.32 device_getIDGStatusCodeString()

```
void device_getIDGStatusCodeString (
    IN int  returnCode,
    OUT char * despcriton )
```

Review the return code description.

Parameters

<i>returnCode</i>	the response result.
<i>description</i>	

Return values

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

- 0: "no error, beginning task";
- 1: "no response from reader";
- 2: "invalid response data";
- 01: " Incorrect Header Tag";
- 02: " Unknown Command";
- 03: " Unknown Sub-Command";
- 04: " CRC Error in Frame";
- 05: " Incorrect Parameter";
- 06: " Parameter Not Supported";
- 07: " Mal-formatted Data";
- 08: " Timeout";
- 0A: " Failed / NACK";
- 0B: " Command not Allowed";
- 0C: " Sub-Command not Allowed";
- 0D: " Buffer Overflow (Data Length too large for reader buffer)";

- 0E: " User Interface Event";
- 10: " Need clear firmware(apply in boot loader only)";
- 11: " Communication type not supported, VT-1, burst, etc. Need encrypted firmware (apply in boot loader only)";
- 12: " Secure interface is not functional or is in an intermediate state.";
- 13: " Data field is not mod 8";
- 14: " Pad 0x80 not found where expected";
- 15: " Specified key type is invalid";
- 16: " Could not retrieve key from the SAM (InitSecureComm)";
- 17: " Hash code problem";
- 18: " Could not store the key into the SAM (InstallKey)";
- 19: " Frame is too large";
- 1A: " Unit powered up in authentication state but POS must resend the InitSecureComm command";
- 1B: " The EEPROM may not be initialized because SecCommInterface does not make sense";
- 1C: " Problem encoding APDU Module-Specific Status Codes ";
- 20: " Unsupported Index (ILM) SAM Transceiver error - problem communicating with the SAM (Key Mgr)";
- 21: " Unexpected Sequence Counter in multiple frames for single bitmap (ILM)Length error in data returned from the SAM (Key Mgr) - 22: " Improper bit map (ILM)"; - 23: " Request Online Authorization"; - 24: " ViV↔ OCard3 raw data read successful"; - 25: " Message index not available (ILM) ViVOcomm activate transaction card type (ViVOcomm)"; - 26: " Version Information Mismatch (ILM)"; - 27: " Not sending commands in correct index message index (ILM)"; - 28: " Time out or next expected message not received (ILM)"; - 29: " ILM languages not available for viewing (ILM)"; - 2A: " Other language not supported (ILM)"; - 41: " from 41 to 4F, Module-specific errors for Key Manager";
- 50: " Auto-Switch OK";
- 51: " Auto-Switch failed";
- 70: " Antenna Error 80h Use another card";
- 81: " Insert or swipe card";
- 90: " Data encryption Key does not exist";
- 91: " Data encryption Key KSN exhausted";

11.12.4.33 device_getMerchantRecord()

```
int device_getMerchantRecord (
    IN int index,
    OUT BYTE * record )
```

DEPRECATED : please use device_getMerchantRecord_Len(IN int index, OUT BYTE * record, IN_OUT int *recordLen)

Get Merchant Record

Gets the merchant record for the device.

Parameters

<i>index</i>	Merchant Record index, valid values 1-6
<i>record;</i>	needs to have at least 99 bytes of memory response data from reader. Merchant Record Index: 1 byte enabled: 1 byte Merchant ID: 32 bytes Length of Merchant URL: 1 byte Merchant URL: 64 bytes

Returns

success or error code. Values can be parsed with [device_getIDGStatusCodeString\(\)](#)

See also

ErrorCode

11.12.4.34 device_getMerchantRecord_Len()

```
int device_getMerchantRecord_Len (
    IN int index,
    OUT BYTE * record,
    IN_OUT int * recordLen )
```

Get Merchant Record

Gets the merchant record for the device.

Parameters

<i>index</i>	Merchant Record index, valid values 1-6
<i>record</i>	response data from reader. Merchant Record Index: 1 byte enabled: 1 byte Merchant ID: 32 bytes Length of Merchant URL: 1 byte Merchant URL: 64 bytes
<i>recordLen</i>	Length of record

Returns

success or error code. Values can be parsed with [device_getIDGStatusCodeString\(\)](#)

See also

ErrorCode

11.12.4.35 device_getSDKWaitTime()

```
int device_getSDKWaitTime ( )
```

Get SDK Wait Time

Get the SDK wait time for transactions

Returns

SDK wait time in seconds

11.12.4.36 device_getThreadStackSize()

```
int device_getThreadStackSize ( )
```

Get Thread Stack Size

Get the stack size setting for newly created threads

Returns

Thread Stack Size

11.12.4.37 device_getTransactionResults()

```
int device_getTransactionResults (
    IDTMSRData * cardData )
```

Get Transaction Results Gets the transaction results when the reader is functioning in "Auto Poll" mode

Parameters

<i>cardData</i>	The transaction results
-----------------	-------------------------

Returns

success or error code. Values can be parsed with [device_getResponseCodeString](#)

See also

[ErrorCode](#)

11.12.4.38 device_init()

```
int device_init ( )
```

Initial the device by USB

It will detect the device and trying connect.

The connect status can be checked by [device_isConnected\(\)](#).

Note: after the function returns success, the function [device_setCurrentDevice\(\)](#) has to be called to set the device type.

Returns

RETURN_CODE: Values can be parsed with [device_getIDGStatusCodeString\(\)](#)

11.12.4.39 device_isAttached()

```
int device_isAttached (
    int deviceType )
```

Check if the device is attached to the USB port The function [device_init\(\)](#) must be called before this function.

Parameters

<i>deviceType</i> , the	device type of the USB device
-------------------------	-------------------------------

Returns

1 if the device is attached, or 0 if the device is not attached

11.12.4.40 device_isConnected()

```
int device_isConnected ( )
```

Check the device connected status

Returns

DEVICE_DISCONNECT=0, or DEVICE_CONNECTED = 1

11.12.4.41 device_pingDevice()

```
int device_pingDevice ( )
```

Ping Device

Pings the reader. If connected, returns success. Otherwise, returns timeout.

Returns

RETURN_CODE: Values can be parsed with [device_getIDGStatusCodeString\(\)](#)

11.12.4.42 device_registerCameraCallBk()

```
void device_registerCameraCallBk (
    pCMR_callback pCMRf )
```

To register the camera callback function to get the image data. (Pass NULL to disable the callback.)

11.12.4.43 device_registerCardStatusFrontSwitchCallBk()

```
void device_registerCardStatusFrontSwitchCallBk (
    pCSFS_callback pCSFSf )
```

To register the card status and front switch callback function to get status. (Pass NULL to disable the callback.)

11.12.4.44 device_SendDataCommandNEO()

```
int device_SendDataCommandNEO (
    IN int cmd,
    IN int subCmd,
    IN BYTE * data,
    IN int dataLen,
```

```
OUT BYTE * response,
IN_OUT int * respLen )
```

Send a Command to device

Sends a command to the device .

Parameters

<i>cmd</i>	buffer of command to execute.
<i>cmdLen,the</i>	length of the buffer cmd.
<i>data</i>	buffer of IDG command data.
<i>dataLen,the</i>	length of the buffer data.
<i>response</i>	Response data
<i>respLen,the</i>	length of Response data

Returns

RETURN_CODE: Values can be parsed with [device_getIDGStatusCodeString\(\)](#) Send a Command to NEO device

Sends a command to the NEO device .

Parameters

<i>cmd</i>	command to execute.
<i>subCmd,sub</i>	command to execute.
<i>data</i>	buffer of NEO command data.
<i>dataLen,the</i>	length of the buffer data.
<i>response</i>	Response data
<i>respLen,the</i>	length of Response data

Returns

RETURN_CODE: Values can be parsed with [device_getIDGStatusCodeString\(\)](#)

11.12.4.45 device_setBurstMode()

```
int device_setBurstMode (
    IN BYTE mode )
```

Send Burst Mode

Sets the burst mode for the device.

Parameters

<i>mode</i>	0 = OFF, 1 = Always On, 2 = Auto Exit
-------------	---------------------------------------

Returns

success or error code. Values can be parsed with [device_getIDGStatusCodeString](#)

See also

[ErrorCode](#)

11.12.4.46 `device_setCurrentDevice()`

```
int device_setCurrentDevice (
    int deviceType )
```

Sets the current device to talk to

The connect status can be checked by [device_isConnected\(\)](#).

Parameters

<i>deviceType</i>	Device to connect to <pre>enum DEVICE_TYPE { IDT_DEVICE_UNKNOWN=0, IDT_DEVICE_AUGUSTA_HID, IDT_DEVICE_AUGUSTA_KB, IDT_DEVICE_AUGUSTA_S_HID, IDT_DEVICE_AUGUSTA_S_KB, IDT_DEVICE_AUGUSTA_S_TTK_HID, IDT_DEVICE_SPECTRUM_PRO, IDT_DEVICE_MINISMART_II, IDT_DEVICE_L80, IDT_DEVICE_L100, IDT_DEVICE_UNIPAY, IDT_DEVICE_UNIPAY_I_V, IDT_DEVICE_VP3300_AJ, IDT_DEVICE_KIOSK_III, IDT_DEVICE_KIOSK_III_S, IDT_DEVICE_PIP_READER, IDT_DEVICE_VENDI, IDT_DEVICE_VP3300_USB, IDT_DEVICE_UNIPAY_I_V_TTK, IDT_DEVICE_VP3300_BT, IDT_DEVICE_VP8800, IDT_DEVICE_SREDKEY2_HID, IDT_DEVICE_SREDKEY2_KB, IDT_DEVICE_NEO2, IDT_DEVICE_MINISMART_II_COM = IDT_DEVICE_NEO2+5, IDT_DEVICE_SPECTRUM_PRO_COM, IDT_DEVICE_KIOSK_III_COM, IDT_DEVICE_KIOSK_III_S_COM, IDT_DEVICE_PIP_READER_COM, IDT_DEVICE_VP3300_COM, IDT_DEVICE_NEO2_COM, IDT_DEVICE_MAX_DEVICES = IDT_DEVICE_NEO2_COM+5 };</pre>
-------------------	---

Returns

RETURN_CODE: 1: success, 0: failed

11.12.4.47 `device_setMerchantRecord()`

```
int device_setMerchantRecord (
    int index,
    int enabled,
    char * merchantID,
    char * merchantURL )
```

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

Returns

RETURN_CODE: Values can be parsed with [device_getIDGStatusCodeString\(\)](#)

11.12.4.48 device_setPollMode()

```
int device_setPollMode (
    IN BYTE mode )
```

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>	0 = Auto Poll, 1 = Poll On Demand
-------------	-----------------------------------

Returns

RETURN_CODE: Values can be parsed with [device_getIDGStatusCodeString\(\)](#)

11.12.4.49 device_setSDKWaitTime()

```
void device_setSDKWaitTime (
    int waitTime )
```

Set SDK Wait Time

Set the SDK wait time for transactions

Parameters

<i>waitTime</i>	The SDK wait time for transaction in seconds. The Maximum is 2147483 seconds.
-----------------	---

11.12.4.50 device_setThreadStackSize()

```
void device_setThreadStackSize (
    int threadSize )
```

Set Thread Stack Size

Set the stack size setting for newly created threads

11.12.4.51 emv_registerCallBk()

```
void emv_registerCallBk (
    pEMV_callback pEMVf )
```

To register the emv callback function to get the EMV processing response. (Pass NULL to disable the callback.)

11.12.4.52 msr_cancelMSRSwipe()

```
int msr_cancelMSRSwipe ( )
```

Disable MSR Swipe Cancels MSR swipe request.

Returns

RETURN_CODE: Values can be parsed with [device_getIDGStatusCodeString\(\)](#)

11.12.4.53 msr_registerCallBk()

```
void msr_registerCallBk (
    pMSR_callback pMSRf )
```

To register the msr callback function to get the MSR card data. (Pass NULL to disable the callback.)

11.12.4.54 msr_registerCallBkp()

```
void msr_registerCallBkp (
    pMSR_callback pMSRf )
```

To register the msr callback function to get the MSR card data pointer. (Pass NULL to disable the callback.)

11.12.4.55 msr_startMSRSwipe()

```
int msr_startMSRSwipe (
    IN int _timeout )
```

Start 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
----------------	---------------------

Returns

RETURN_CODE: Values can be parsed with [device_getIDGStatusCodeString\(\)](#) Note: if auto poll mode is on, it will return command not allowed error

11.12.4.56 parseMSRData()

```
void parseMSRData (
    IN BYTE * resData,
    IN int resLen,
    IN_OUT IDTMSRData * cardData )
```

Parser the MSR data from the buffer into IDTMSTData structure

Parameters

<i>resData</i>	MSR card data buffer
<i>resLen</i>	the length of resData
<i>cardData</i>	the parser result with IDTMSTData structure

11.12.4.57 pin_registerCallBk()

```
void pin_registerCallBk (
    pPIN_callback pPINf )
```

To register the pin callback function to get the PINPad data. (Pass NULL to disable the callback.)

11.12.4.58 registerHotplugCallBk()

```
void registerHotplugCallBk (
    pMessageHotplug pMsgHotplug )
```

To register the USB HID hot-plug callback function which implemented in the application to monitor the hotplug message from the SDK.

11.12.4.59 registerLogCallBk()

```
void registerLogCallBk (
    pSendDataLog pFSend,
    pReadDataLog pFRead )
```

To register the log callback function which implemented in the application to monitor sending/reading data between application and reader.

11.12.4.60 SDK_Version()

```
char* SDK_Version ( )
```

To Get SDK version

Returns

return the SDK version string

11.12.4.61 setAbsoluteLibraryPath()

```
int setAbsoluteLibraryPath (
    const char * absoluteLibraryPath )
```

Set the path to use when searching for ID TECH's libraries. If this is not set, the libraries will be searched for with the system's default procedures.

Parameters

<i>absoluteLibraryPath</i>	The absolute path to ID TECH's libraries.
----------------------------	---

Returns

RETURN_CODE: Values can be parsed with [device_getIDGStatusCodeString\(\)](#)

11.13 Source_C/libIDT_VP3300_AJ.h File Reference

VP3300 AJ API.

```
#include "IDTDef.h"
```

Macros

- #define [IN](#)
- #define [OUT](#)
- #define [IN_OUT](#)

Typedefs

- typedef void(* [pMessageHotplug](#)) (int, int)
- typedef void(* [pSendDataLog](#)) (BYTE *, int)
- typedef void(* [pReadDataLog](#)) (BYTE *, int)
- typedef void(* [pEMV_callBack](#)) (int, int, BYTE *, int, IDTTransactionData *, EMV_Callback *, int)
- typedef void(* [pMSR_callBack](#)) (int, IDTMSRData)
- typedef void(* [pMSR_callBackp](#)) (int, IDTMSRData *)
- typedef void(* [pPIN_callBack](#)) (int, IDTPINData *)
- typedef void(* [pCMR_callBack](#)) (int, IDTCMRData *)
- typedef void(* [pCSFS_callBack](#)) (BYTE status)
- typedef void(* [pRKI_callBack](#)) (int, char *)
- typedef void(* [ftpComm_callBack](#)) (int, int, int)
- typedef void(* [httpComm_callBack](#)) (BYTE *, int)
- typedef void(* [v4Comm_callBack](#)) (BYTE, BYTE, BYTE *, int)
- typedef void(* [pWP_callBack](#)) (char *, int, int)
- typedef void(* [pWN_callBack](#)) (char *, int, int)

Functions

- void [registerHotplugCallBk](#) ([pMessageHotplug](#) pMsgHotplug)
- void [registerLogCallBk](#) ([pSendDataLog](#) pFSend, [pReadDataLog](#) pFRead)
- void [device_registerRKICallBk](#) ([pRKI_callBack](#) pRKIf)
- void [emv_registerCallBk](#) ([pEMV_callBack](#) pEMVf)
- void [msr_registerCallBk](#) ([pMSR_callBack](#) pMSRf)
- void [msr_registerCallBkp](#) ([pMSR_callBackp](#) pMSRf)
- void [ctls_registerCallBk](#) ([pMSR_callBack](#) pCTLSf)
- void [ctls_registerCallBkp](#) ([pMSR_callBackp](#) pCTLSf)
- void [pin_registerCallBk](#) ([pPIN_callBack](#) pPINf)
- void [device_registerCameraCallBk](#) ([pCMR_callBack](#) pCMRf)
- void [device_registerCardStatusFrontSwitchCallBk](#) ([pCSFS_callBack](#) pCSFSf)
- int [device_getSDKWaitTime](#) ()
- void [device_setSDKWaitTime](#) (int waitTime)
- int [device_getThreadStackSize](#) ()
- void [device_setThreadStackSize](#) (int threadSize)
- void [comm_registerHTTPCallback](#) ([httpComm_callBack](#) cBack)
- void [comm_registerV4Callback](#) ([v4Comm_callBack](#) cBack)
- char * [SDK_Version](#) ()
- int [setAbsoluteLibraryPath](#) (const char *absoluteLibraryPath)
- int [device_init](#) ()
- int [device_setCurrentDevice](#) (int deviceType)
- int [device_close](#) ()
- void [device_getIDGStatusCodeString](#) ([IN](#) int returnCode, [OUT](#) char *despcriton)
- int [device_isConnected](#) ()

- int [device_isAttached](#) (int deviceType)
- int [device_getFirmwareVersion](#) (OUT char *firmwareVersion)
- int [device_getFirmwareVersion_Len](#) (OUT char *firmwareVersion, IN_OUT int *firmwareVersionLen)
- int [device_pingDevice](#) ()
- int [device_controlUserInterface](#) (IN BYTE *values)
- int [device_getCurrentDeviceType](#) ()
- int [device_SendDataCommandNEO](#) (IN int cmd, IN int subCmd, IN BYTE *data, IN int dataLen, OUT BYTE *response, IN_OUT int *respLen)
- int [device_enablePassThrough](#) (int enablePassThrough)
- int [device_setBurstMode](#) (IN BYTE mode)
- int [device_setPollMode](#) (IN BYTE mode)
- int [device_setMerchantRecord](#) (int index, int enabled, char *merchantID, char *merchantURL)
- int [device_pollForToken](#) (IN int timeout, OUT BYTE *respData, IN_OUT int *respDataLen)
- int [device_getMerchantRecord](#) (IN int index, OUT BYTE *record)
- int [device_getMerchantRecord_Len](#) (IN int index, OUT BYTE *record, IN_OUT int *recordLen)
- int [device_getTransactionResults](#) (IDTMSRData *cardData)
- int [device_startTransaction](#) (IN double amount, IN double amtOther, IN int type, IN const int _timeout, IN BYTE *tags, IN int tagsLen)
- void [device_setTransactionExponent](#) (int exponent)
- int [device_activateTransaction](#) (IN const int _timeout, IN BYTE *tags, IN int tagsLen)
- int [device_cancelTransaction](#) ()
- int [device_getRTCDDateTime](#) (IN BYTE *dateTime, IN_OUT int *dateTimeLen)
- int [device_setRTCDDateTime](#) (IN BYTE *dateTime, IN int dateTimeLen)
- int [device_startRKI](#) (IN const char *caPath, IN int isProduction)
- void [device_setRKI_URL](#) (IN char *rkiURL, IN int rkiURLLen)
- int [config_getSerialNumber](#) (OUT char *sNumber)
- int [config_getSerialNumber_Len](#) (OUT char *sNumber, IN_OUT int *sNumberLen)
- int [ctls_startTransaction](#) (IN double amount, IN double amtOther, IN int type, IN const int _timeout, IN BYTE *tags, IN int tagsLen)
- int [ctls_activateTransaction](#) (IN const int _timeout, IN BYTE *tags, IN int tagsLen)
- int [ctls_cancelTransaction](#) ()
- int [ctls_retrieveApplicationData](#) (IN BYTE *AID, IN int AIDLen, OUT BYTE *tlv, IN_OUT int *tlvLen)
- int [ctls_setApplicationData](#) (IN BYTE *tlv, IN int tlvLen)
- int [ctls_removeApplicationData](#) (IN BYTE *AID, IN int AIDLen)
- int [ctls_removeAllApplicationData](#) ()
- int [ctls_retrieveAIDList](#) (OUT BYTE *AIDList, IN_OUT int *AIDListLen)
- int [ctls_retrieveTerminalData](#) (OUT BYTE *tlv, IN_OUT int *tlvLen)
- int [ctls_setTerminalData](#) (IN BYTE *tlv, IN int tlvLen)
- int [ctls_retrieveCAPK](#) (IN BYTE *capk, IN int capkLen, OUT BYTE *key, IN_OUT int *keyLen)
- int [ctls_setCAPK](#) (IN BYTE *capk, IN int capkLen)
- int [ctls_removeCAPK](#) (IN BYTE *capk, IN int capkLen)
- int [ctls_removeAllCAPK](#) ()
- int [ctls_retrieveCAPKList](#) (OUT BYTE *keys, IN_OUT int *keysLen)
- int [ctls_setConfigurationGroup](#) (IN BYTE *tlv, IN int tlvLen)
- int [ctls_getConfigurationGroup](#) (IN int group, OUT BYTE *tlv, OUT int *tlvLen)
- int [ctls_getAllConfigurationGroups](#) (OUT BYTE *tlv, IN_OUT int *tlvLen)
- int [ctls_removeConfigurationGroup](#) (int group)
- void [emv_allowFallback](#) (IN int allow)
- void [emv_setAutoAuthenticateTransaction](#) (IN int authenticate)
- void [emv_setAutoCompleteTransaction](#) (IN int complete)
- int [emv_getAutoAuthenticateTransaction](#) ()
- int [emv_getAutoCompleteTransaction](#) ()
- void [emv_setTransactionParameters](#) (IN double amount, IN double amtOther, IN int type, IN int timeout, IN BYTE *tags, IN int tagsLen)

- int [emv_startTransaction](#) (IN double amount, IN double amtOther, IN int exponent, IN int type, IN int timeout, IN BYTE *tags, IN int tagsLen, IN int forceOnline)
- int [emv_activateTransaction](#) (IN int timeout, IN BYTE *tags, IN int tagsLen, IN int forceOnline)
- int [emv_authenticateTransaction](#) (IN BYTE *updatedTLV, IN int updatedTLVLen)
- int [emv_authenticateTransactionWithTimeout](#) (IN int timeout, IN BYTE *updatedTLV, IN int updatedTLVLen)
- int [emv_completeTransaction](#) (IN int commError, IN BYTE *authCode, IN int authCodeLen, IN BYTE *iad, IN int iadLen, IN BYTE *tlvScripts, IN int tlvScriptsLen, IN BYTE *tlv, IN int tlvLen)
- int [emv_cancelTransaction](#) ()
- int [emv_retrieveApplicationData](#) (IN BYTE *AID, IN int AIDLen, OUT BYTE *tlv, IN_OUT int *tlvLen)
- int [emv_setApplicationData](#) (IN BYTE *name, IN int nameLen, IN BYTE *tlv, IN int tlvLen)
- int [emv_setApplicationDataTLV](#) (IN BYTE *tlv, IN int tlvLen)
- int [emv_removeApplicationData](#) (IN BYTE *AID, IN int AIDLen)
- int [emv_removeAllApplicationData](#) ()
- int [emv_retrieveAIDList](#) (OUT BYTE *AIDList, IN_OUT int *AIDListLen)
- int [emv_retrieveTerminalData](#) (OUT BYTE *tlv, IN_OUT int *tlvLen)
- int [emv_setTerminalData](#) (IN BYTE *tlv, IN int tlvLen)
- int [emv_setTerminalMajorConfiguration](#) (IN int configuration)
- int [emv_retrieveCAPK](#) (IN BYTE *capk, IN int capkLen, OUT BYTE *key, IN_OUT int *keyLen)
- int [emv_setCAPK](#) (IN BYTE *capk, IN int capkLen)
- int [emv_removeCAPK](#) (IN BYTE *capk, IN int capkLen)
- int [emv_removeAllCAPK](#) ()
- int [emv_retrieveCAPKList](#) (OUT BYTE *keys, IN_OUT int *keysLen)
- int [emv_retrieveCRL](#) (OUT BYTE *list, IN_OUT int *lssLen)
- int [emv_setCRL](#) (IN BYTE *list, IN int lsLen)
- int [emv_removeCRL](#) (IN BYTE *list, IN int lsLen)
- int [emv_removeAllCRL](#) ()
- int [icc_getICCRReaderStatus](#) (OUT BYTE *status)
- int [icc_powerOnICC](#) (OUT BYTE *ATR, IN_OUT int *inLen)
- int [icc_powerOffICC](#) ()
- int [icc_exchangeAPDU](#) (IN BYTE *c_APDU, IN int cLen, OUT BYTE *reData, IN_OUT int *reLen)
- int [msr_cancelMSRSwipe](#) ()
- int [msr_startMSRSwipe](#) (IN int _timeout)
- int [executeTransaction](#) (WorldPayData *data, [pWP_callback](#) wpCallback, int requestOnly)
- int [forwardTransaction](#) (IN [pWP_callback](#) wpCallback, IN char *forwardID, IN int forwardIDLen, IN char *password, IN int passwordLen, IN int bypassProcessing)
- int [cancelWorldPay](#) ()
- int [executeTransaction_WorldNet](#) (WorldNetData *data, [pWN_callback](#) wnCallback, int requestOnly)
- int [forwardTransaction_WorldNet](#) (IN char *apiKey, IN int apiKeyLen, IN [pWN_callback](#) wnCallback, IN char *forwardID, IN int forwardIDLen, IN char *password, IN int passwordLen, IN int bypassProcessing)
- int [cancelWorldNet](#) ()
- void [parseMSRData](#) (IN BYTE *resData, IN int resLen, IN_OUT IDTMSRData *cardData)

11.13.1 Detailed Description

VP3300 AJ API.

VP3300 AJ Global API methods.

11.13.2 Macro Definition Documentation

11.13.2.1 IN

```
#define IN
```

INPUT parameter.

11.13.2.2 IN_OUT

```
#define IN_OUT
```

INPUT / OUTPUT PARAMETER.

11.13.2.3 OUT

```
#define OUT
```

OUTPUT parameter.

11.13.3 Typedef Documentation**11.13.3.1 ftpComm_callBack**

```
typedef void(* ftpComm_callBack) (int, int, int)
```

Define the comm callback function to get FTP file transfer status

It should be passed as a parameter in a FTP request, Signature (int, int, int) = response code, current block, total blocks
 RESPONSE CODES: 100 = FILE DOWNLOAD STARTED 101 = FILE BLOCK XX OF XX RECEIVED 102 = FILE DOWNLOAD COMPLETED 103 = FILE DOWNLOAD TERMINATED PREMATURELY

11.13.3.2 httpComm_callBack

```
typedef void(* httpComm_callBack) (BYTE *, int)
```

Define the comm callback function to get the async url data

It should be registered using the comm_registerHTTPCallback

11.13.3.3 pCMR_callBack

```
typedef void(* pCMR_callBack) (int, IDTCMRData *)
```

Define the camera callback function to get the image data

It should be registered using the device_registerCameraCallBk,

11.13.3.4 pCSFS_callBack

```
typedef void(* pCSFS_callBack) (BYTE status)
```

Define the card status and front switch callback function to get card and front switch status

It should be registered using the device_registerCardStatusFrontSwitchCallBk,

11.13.3.5 pEMV_callBack

```
typedef void(* pEMV_callBack) (int, int, BYTE *, int, IDTTransactionData *, EMV_Callback *, int)
```

Define the EMV callback function to get the transaction message/data/result.
It should be registered using the `emv_registerCallBk`,

11.13.3.6 pMessageHotplug

```
typedef void(* pMessageHotplug) (int, int)
```

Define the USB hot-plug callback function to monitor the info when plug in/out the reader.
It should be registered using the `registerHotplugCallBk`, The first integer parameter is device type, and the second integer parameter is either 0: Device Plugged Out or 1: Device Plugged In

11.13.3.7 pMSR_callBack

```
typedef void(* pMSR_callBack) (int, IDTMSRData)
```

Define the MSR callback function to get the MSR card data
It should be registered using the `msr_registerCallBk`, this callback function is for backward compatibility

11.13.3.8 pMSR_callBackp

```
typedef void(* pMSR_callBackp) (int, IDTMSRData *)
```

Define the MSR callback function to get pointer to the MSR card data
It should be registered using the `msr_registerCallBk`, this callback function is recommended instead of `pMSR_callBack`

11.13.3.9 pPIN_callBack

```
typedef void(* pPIN_callBack) (int, IDTPINData *)
```

Define the PINPad callback function to get the input PIN Pad data
It should be registered using the `pin_registerCallBk`,

11.13.3.10 pReadDataLog

```
typedef void(* pReadDataLog) (BYTE *, int)
```

Define the read response callback function to monitor the reading response from the reader.
It should be registered using the `registerLogCallBk`,

11.13.3.11 pRKI_callBack

```
typedef void(* pRKI_callBack) (int, char *)
```

Define the RKI callback function to get the status of the RKI
It should be registered using the `device_registerRKICallBk`,

11.13.3.12 pSendDataLog

```
typedef void(* pSendDataLog) (BYTE *, int)
```

Define the send command callback function to monitor the sending command into the reader.
It should be registered using the `registerLogCallBk`,

11.13.3.13 pWN_callBack

```
typedef void(* pWN_callBack) (char *, int, int)
```

Define the Worldnet callback function to get the transaction message/data/result.

11.13.3.14 pWP_callBack

```
typedef void(* pWP_callBack) (char *, int, int)
```

Define the Worldpay callback function to get the transaction message/data/result.

11.13.3.15 v4Comm_callBack

```
typedef void(* v4Comm_callBack) (BYTE, BYTE, BYTE *, int)
```

Define the comm callback function to receive the V4 Protocol packets received by the device from an external source (IP/USB/RS-232) It should be registered using the `comm_registerV4Callback`, Data callback will contain command, sub-command, and data from V4 packet

11.13.4 Function Documentation**11.13.4.1 cancelWorldNet()**

```
int cancelWorldNet ( )
```

Cancels WorldNet transaction.

Returns

RETURN_CODE: Values can be parsed with [device_getIDGStatusCodeString\(\)](#)

11.13.4.2 cancelWorldPay()

```
int cancelWorldPay ( )
```

Cancels WorldPay transaction.

Returns

RETURN_CODE: Values can be parsed with [device_getIDGStatusCodeString\(\)](#)

11.13.4.3 comm_registerHTTPCallback()

```
void comm_registerHTTPCallback (
    httpComm_callBack cBack )
```

Register Comm HTTP Async Callback

Parameters

<i>cBack</i>	- HTTP Comm callback
--------------	----------------------

11.13.4.4 comm_registerV4Callback()

```
void comm_registerV4Callback (
    v4Comm_callBack cBack )
```

Register External V4 Protocol commands Callback

Parameters

<i>cBack</i>	- V4 Protocol Comm callback
--------------	-----------------------------

11.13.4.5 config_getSerialNumber()

```
int config_getSerialNumber (
    OUT char * sNumber )
```

DEPRECATED : please use [config_getSerialNumber_Len\(OUT char* sNumber, IN_OUT int *sNumberLen\)](#)

Polls device for Serial Number

Parameters

<i>sNumber</i>	Returns Serial Number; needs to have at least 64 bytes of memory
----------------	--

Returns

RETURN_CODE: Values can be parsed with [device_getIDGStatusCodeString](#)

11.13.4.6 config_getSerialNumber_Len()

```
int config_getSerialNumber_Len (
    OUT char * sNumber,
    IN_OUT int * sNumberLen )
```

Polls device for Serial Number

Parameters

<i>sNumber</i>	Returns Serial Number
<i>sNumberLen</i>	Length of Serial Number

Returns

RETURN_CODE: Values can be parsed with `device_getIDGStatusCodeString`

11.13.4.7 ctls_activateTransaction()

```
int ctls_activateTransaction (
    IN const int _timeout,
    IN BYTE * tags,
    IN int tagsLen )
```

Start CTLS Transaction Request

Authorizes the CTLS 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 stream. Example, tag 9F0C with amount 0x000000000100 would be 0x9F0C06000000000100
<i>tagsLen</i>	The length of tags data buffer.

>>>>>IMPORTANT: parameters for amount and amtOther MUST BE PASSED AS A DOUBLE VALUE WITH DECIMAL POINT. Example, do not pass 1, but instead pass 1.0 or 1.00. Otherwise, results will be unpredictable

Returns

RETURN_CODE: Values can be parsed with [device_getIDGStatusCodeString\(\)](#) Note: if auto poll is on, it will return the error IDG_P2_STATUS_CODE_COMMAND_NOT_ALLOWED

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

11.13.4.8 ctls_cancelTransaction()

```
int ctls_cancelTransaction ( )
```

Cancel EMV Transaction

Cancels the currently executing EMV transaction.

Returns

RETURN_CODE: Values can be parsed with [device_getIDGStatusCodeString\(\)](#)

11.13.4.9 ctls_getAllConfigurationGroups()

```
int ctls_getAllConfigurationGroups (
    OUT BYTE * tlv,
    IN_OUT int * tlvLen )
```

Retrieve All Configuration Groups

Returns all the Configuration Groups installed on the terminal for CTLS

Parameters

<i>tlv</i>	The TLV elements data
<i>tlvLen</i>	the length of tlv data buffer.

Returns

RETURN_CODE: Values can be parsed with [device_getIDGStatusCodeString\(\)](#)

11.13.4.10 ctls_getConfigurationGroup()

```
int ctls_getConfigurationGroup (
    IN int group,
    OUT BYTE * tlv,
    OUT int * tlvLen )
```

Get Configuration Group

Retrieves the Configuration for the specified Group.

Parameters

<i>group</i>	Configuration Group
<i>tlv</i>	return data
<i>tlvLen</i>	the length of tlv data buffer

Returns

RETURN_CODE: Values can be parsed with [device_getIDGStatusCodeString\(\)](#)

11.13.4.11 ctls_registerCallBk()

```
void ctls_registerCallBk (
    pMSR_callback pCTLSf )
```

To register the ctls callback function to get the MSR card data. (Pass NULL to disable the callback.)

11.13.4.12 ctls_registerCallBkp()

```
void ctls_registerCallBkp (
    pMSR_callbackp pCTLSf )
```

To register the ctls callback function to get the MSR card data pointer. (Pass NULL to disable the callback.)

11.13.4.13 ctls_removeAllApplicationData()

```
int ctls_removeAllApplicationData ( )
```

Remove All Application Data

Removes all the Application Data

Returns

RETURN_CODE: Values can be parsed with [device_getIDGStatusCodeString\(\)](#)

11.13.4.14 ctls_removeAllCAPK()

```
int ctls_removeAllCAPK ( )
```

Remove All Certificate Authority Public Key

Removes all the CAPK

Returns

RETURN_CODE: Values can be parsed with [device_getIDGStatusCodeString\(\)](#)

11.13.4.15 ctls_removeApplicationData()

```
int ctls_removeApplicationData (
    IN BYTE * AID,
    IN int AIDLen )
```

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>AIDLen</i>	the length of AID data buffer

Returns

RETURN_CODE: Values can be parsed with [device_getIDGStatusCodeString\(\)](#)

11.13.4.16 ctls_removeCAPK()

```
int ctls_removeCAPK (
    IN BYTE * capk,
    IN int capkLen )
```

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>capkLen</i>	the length of capk data buffer

Returns

RETURN_CODE: Values can be parsed with [device_getIDGStatusCodeString\(\)](#)

11.13.4.17 ctls_removeConfigurationGroup()

```
int ctls_removeConfigurationGroup (
    int group )
```

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>	Values can be parsed with device_getIDGStatusCodeString()
--------------------	---

11.13.4.18 ctls_retrieveAIDList()

```
int ctls_retrieveAIDList (
    OUT BYTE * AIDList,
    IN_OUT int * AIDListLen )
```

Retrieve AID list

Returns all the AID names installed on the terminal for CTLS. .

Parameters

<i>AIDList</i>	array of AID name byte arrays
<i>AIDListLen</i>	the length of AIDList array buffer

Returns

RETURN_CODE: Values can be parsed with [device_getIDGStatusCodeString\(\)](#)

11.13.4.19 ctls_retrieveApplicationData()

```
int ctls_retrieveApplicationData (
    IN BYTE * AID,
    IN int AIDLen,
    OUT BYTE * tlv,
    IN_OUT int * tlvLen )
```

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>AIDLen</i>	the length of AID data buffer.
<i>tlv</i>	The TLV elements of the requested AID
<i>tlvLen</i>	the length of tlv data buffer.

Returns

RETURN_CODE: Values can be parsed with [device_getIDGStatusCodeString\(\)](#)

11.13.4.20 ctls_retrieveCAPK()

```
int ctls_retrieveCAPK (
    IN BYTE * capk,
    IN int capkLen,
    OUT BYTE * key,
    IN_OUT int * keyLen )
```

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>capkLen</i>	the length of capk data buffer
<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>keyLen</i>	<p>the length of key data buffer</p> <ul style="list-style-type: none"> •

Returns

RETURN_CODE: Values can be parsed with [device_getIDGStatusCodeString\(\)](#)

11.13.4.21 ctls_retrieveCAPKList()

```
int ctls_retrieveCAPKList (
    OUT BYTE * keys,
    IN_OUT int * keysLen )
```

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>keysLen</i>	the length of keys data buffer

Returns

RETURN_CODE: Values can be parsed with [device_getIDGStatusCodeString\(\)](#)

11.13.4.22 ctls_retrieveTerminalData()

```
int ctls_retrieveTerminalData (
    OUT BYTE * tlv,
    IN_OUT int * tlvLen )
```

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>tlvLen</i>	the length of tlv data buffer

Returns

RETURN_CODE: Values can be parsed with [device_getIDGStatusCodeString\(\)](#)

11.13.4.23 ctls_setApplicationData()

```
int ctls_setApplicationData (
    IN BYTE * tlv,
    IN int tlvLen )
```

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>tlvLen</i>	the length of tlv data buffer
---------------	-------------------------------

Returns

RETURN_CODE: Values can be parsed with [device_getIDGStatusCodeString\(\)](#)

11.13.4.24 ctls_setCAPK()

```
int ctls_setCAPK (
    IN BYTE * capk,
    IN int capkLen )
```

Set Certificate Authority Public Key

Sets the CAPK as specified by the CAKey structure

Parameters

<i>capk</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>capkLen</i>	the length of capk data buffer

Returns

RETURN_CODE: Values can be parsed with [device_getIDGStatusCodeString\(\)](#)

11.13.4.25 ctls_setConfigurationGroup()

```
int ctls_setConfigurationGroup (
    IN BYTE * tlv,
    IN int tlvLen )
```

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>tlvLen</i>	the length of tlv data buffer

Returns

RETURN_CODE: Values can be parsed with [device_getIDGStatusCodeString\(\)](#)

11.13.4.26 ctls_setTerminalData()

```
int ctls_setTerminalData (
    IN BYTE * tlv,
    IN int tlvLen )
```

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 [ctls_getConfigurationGroup\(int group\)](#), and deleted with [ctls_removeConfigurationGroup\(int group\)](#). You cannot delete group 0.

Parameters

<i>tlv</i>	TerminalData configuration file
<i>tlvLen</i>	the length of tlv data buffer

Return values

<i>RETURN_CODE</i>	Values can be parsed with device_getIDGStatusCodeString()
--------------------	---

11.13.4.27 ctls_startTransaction()

```
int ctls_startTransaction (
    IN double amount,
    IN double amtOther,
    IN int type,
    IN const int _timeout,
    IN BYTE * tags,
    IN int tagsLen )
```

Start CTLS Transaction Request

Authorizes the CTLS transaction for an ICC card

The tags will be returned in the callback routine.

Parameters

<i>amount</i>	Transaction amount value (tag value 9F02) - SEE IMPORTANT NOTE BELOW
<i>amtOther</i>	Other amount value, if any (tag value 9F03) - SEE IMPORTANT NOTE BELOW
<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 stream. Example, tag 9F0C with amount 0x000000000100 would be 0x9F0C06000000000100 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>tagsLen</i>	The length of tags data buffer.

>>>>>IMPORTANT: parameters for amount and amtOther MUST BE PASSED AS A DOUBLE VALUE WITH DECIMAL POINT. Example, do not pass 1, but instead pass 1.0 or 1.00. Otherwise, results will be unpredictable

Returns

RETURN_CODE: Values can be parsed with [device_getIDGStatusCodeString\(\)](#) Note: if auto poll is on, it will return the error IDG_P2_STATUS_CODE_COMMAND_NOT_ALLOWED

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) DFO1 = 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

11.13.4.28 device_activateTransaction()

```
int device_activateTransaction (
    IN const int _timeout,
    IN BYTE * tags,
    IN int tagsLen )
```

Start Transaction Request

Authorizes the transaction for an MSR/CTLS/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. Example, tag 9F0C with amount 0x0000000000100 would be 0x9F0C060000000000100 Be sure to include 9F02 (amount)and9C (transaction type).
<i>tagsLen</i>	The length of tags data buffer.

>>>>>IMPORTANT: parameters for amount and amtOther MUST BE PASSED AS A DOUBLE VALUE WITH DECIMAL POINT. Example, do not pass 1, but instead pass 1.0 or 1.00. Otherwise, results will be unpredictable

Returns

RETURN_CODE: Values can be parsed with [device_getIDGStatusCodeString\(\)](#) Note: if auto poll is on, it will return the error IDG_P2_STATUS_CODE_COMMAND_NOT_ALLOWED

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 device_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

11.13.4.29 device_cancelTransaction()

```
int device_cancelTransaction ( )
```

Disable Transaction Cancel Transaction request.

Returns

RETURN_CODE: Values can be parsed with [device_getIDGStatusCodeString\(\)](#)

11.13.4.30 device_close()

```
int device_close ( )
```

Close the device

Returns

RETURN_CODE: 0: success, 0x0A: failed

11.13.4.31 device_controlUserInterface()

```
int device_controlUserInterface (
    IN BYTE * values )
```

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) • 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
---------------	--

Returns

RETURN_CODE: Values can be parsed with [device_getIDGStatusCodeString\(\)](#)

11.13.4.32 device_enablePassThrough()

```
int device_enablePassThrough (
    int enablePassThrough )
```

Enable Pass Through

Enables Pass Through Mode for direct communication with L1 interface (power on icc, send apdu, etc).

Parameters

<i>enablePassThrough</i>	1 = pass through ON, 0 = pass through OFF
--------------------------	---

Returns

RETURN_CODE: Values can be parsed with [device_getIDGStatusCodeString\(\)](#)

11.13.4.33 device_getCurrentDeviceType()

```
int device_getCurrentDeviceType ( )
```

Get current active device type

Returns

: return the device type defined as DEVICE_TYPE in the IDTDef.h

11.13.4.34 device_getFirmwareVersion()

```
int device_getFirmwareVersion (
    OUT char * firmwareVersion )
```

DEPRECATED : please use [device_getFirmwareVersion_Len](#)(OUT char* firmwareVersion, IN_OUT int *firmwareVersionLen)

Polls device for Firmware Version

Parameters

<i>firmwareVersion</i>	Response returned of Firmware Version; needs to have at least 128 bytes of memory
------------------------	---

Returns

RETURN_CODE: Values can be parsed with [device_getIDGStatusCodeString\(\)](#)

11.13.4.35 device_getFirmwareVersion_Len()

```
int device_getFirmwareVersion_Len (
    OUT char * firmwareVersion,
    IN_OUT int * firmwareVersionLen )
```

Polls device for Firmware Version

Parameters

<i>firmwareVersion</i>	Response returned of Firmware Version
<i>firmwareVersionLen</i>	Length of Firmware Version

Returns

RETURN_CODE: Values can be parsed with [device_getIDGStatusCodeString\(\)](#)

11.13.4.36 device_getIDGStatusCodeString()

```
void device_getIDGStatusCodeString (
    IN int  returnCode,
    OUT char * despcrition )
```

Review the return code description.

Parameters

<i>returnCode</i>	the response result.
<i>description</i>	

Return values

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

- 0: "no error, beginning task";
- 1: "no response from reader";
- 2: "invalid response data";
- 01: " Incorrect Header Tag";
- 02: " Unknown Command";
- 03: " Unknown Sub-Command";
- 04: " CRC Error in Frame";
- 05: " Incorrect Parameter";
- 06: " Parameter Not Supported";
- 07: " Mal-formatted Data";
- 08: " Timeout";
- 0A: " Failed / NACK";
- 0B: " Command not Allowed";
- 0C: " Sub-Command not Allowed";
- 0D: " Buffer Overflow (Data Length too large for reader buffer)";

- 0E: " User Interface Event";
- 10: " Need clear firmware(apply in boot loader only)";
- 11: " Communication type not supported, VT-1, burst, etc. Need encrypted firmware (apply in boot loader only)";
- 12: " Secure interface is not functional or is in an intermediate state.";
- 13: " Data field is not mod 8";
- 14: " Pad 0x80 not found where expected";
- 15: " Specified key type is invalid";
- 16: " Could not retrieve key from the SAM (InitSecureComm)";
- 17: " Hash code problem";
- 18: " Could not store the key into the SAM (InstallKey)";
- 19: " Frame is too large";
- 1A: " Unit powered up in authentication state but POS must resend the InitSecureComm command";
- 1B: " The EEPROM may not be initialized because SecCommInterface does not make sense";
- 1C: " Problem encoding APDU Module-Specific Status Codes ";
- 20: " Unsupported Index (ILM) SAM Transceiver error - problem communicating with the SAM (Key Mgr)";
- 21: " Unexpected Sequence Counter in multiple frames for single bitmap (ILM)Length error in data returned from the SAM (Key Mgr) - 22: " Improper bit map (ILM)"; - 23: " Request Online Authorization"; - 24: " ViV↔ OCard3 raw data read successful"; - 25: " Message index not available (ILM) ViVOcomm activate transaction card type (ViVOcomm)"; - 26: " Version Information Mismatch (ILM)"; - 27: " Not sending commands in correct index message index (ILM)"; - 28: " Time out or next expected message not received (ILM)"; - 29: " ILM languages not available for viewing (ILM)"; - 2A: " Other language not supported (ILM)"; - 41: " from 41 to 4F, Module-specific errors for Key Manager";
- 50: " Auto-Switch OK";
- 51: " Auto-Switch failed";
- 70: " Antenna Error 80h Use another card";
- 81: " Insert or swipe card";
- 90: " Data encryption Key does not exist";
- 91: " Data encryption Key KSN exhausted";

11.13.4.37 device_getMerchantRecord()

```
int device_getMerchantRecord (
    IN int index,
    OUT BYTE * record )
```

DEPRECATED : please use device_getMerchantRecord_Len(IN int index, OUT BYTE * record, IN_OUT int *recordLen)

Get Merchant Record

Gets the merchant record for the device.

Parameters

<i>index</i>	Merchant Record index, valid values 1-6
<i>record;</i>	needs to have at least 99 bytes of memory response data from reader. Merchant Record Index: 1 byte enabled: 1 byte Merchant ID: 32 bytes Length of Merchant URL: 1 byte Merchant URL: 64 bytes

Returns

success or error code. Values can be parsed with [device_getIDGStatusCodeString\(\)](#)

See also

[ErrorCode](#)

11.13.4.38 device_getMerchantRecord_Len()

```
int device_getMerchantRecord_Len (
    IN int index,
    OUT BYTE * record,
    IN_OUT int * recordLen )
```

Get Merchant Record

Gets the merchant record for the device.

Parameters

<i>index</i>	Merchant Record index, valid values 1-6
<i>record</i>	response data from reader. Merchant Record Index: 1 byte enabled: 1 byte Merchant ID: 32 bytes Length of Merchant URL: 1 byte Merchant URL: 64 bytes
<i>recordLen</i>	Length of record

Returns

success or error code. Values can be parsed with [device_getIDGStatusCodeString\(\)](#)

See also

[ErrorCode](#)

11.13.4.39 device_getRTCDateTime()

```
int device_getRTCDateTime (
    IN BYTE * dateTime,
    IN_OUT int * dateTimeLen )
```

get RTC date and time of the device

Parameters

<i>dateTime</i>	<dateTime data>="" is: 6 byte data for YYMMDDHHMMSS in hex. For example 0x171003102547 stands for 2017 Oct 3rd 10:25:47
<i>dateTimeLen</i>	return 6 bytes if successful

Returns

success or error code. Values can be parsed with device_getResponseCodeString

See also

ErrorCode

11.13.4.40 device_getSDKWaitTime()

```
int device_getSDKWaitTime ( )
```

Get SDK Wait Time

Get the SDK wait time for transactions

Returns

SDK wait time in seconds

11.13.4.41 device_getThreadStackSize()

```
int device_getThreadStackSize ( )
```

Get Thread Stack Size

Get the stack size setting for newly created threads

Returns

Thread Stack Size

11.13.4.42 device_getTransactionResults()

```
int device_getTransactionResults (
    IDTMSRData * cardData )
```

Get Transaction Results Gets the transaction results when the reader is functioning in "Auto Poll" mode

Parameters

<i>cardData</i>	The transaction results
-----------------	-------------------------

Returns

success or error code. Values can be parsed with device_getResponseCodeString

See also

ErrorCode

11.13.4.43 device_init()

```
int device_init ( )
```

Initial the device by USB

It will detect the device and trying connect.

The connect status can be checked by [device_isConnected\(\)](#).

Note: after the function returns success, the function [device_setCurrentDevice\(\)](#) has to be called to set the device type.

Returns

RETURN_CODE: Values can be parsed with [device_getIDGStatusCodeString\(\)](#)

11.13.4.44 device_isAttached()

```
int device_isAttached (
    int deviceType )
```

Check if the device is attached to the USB port The function [device_init\(\)](#) must be called before this function.

Parameters

<i>deviceType,the</i>	device type of the USB device
-----------------------	-------------------------------

Returns

1 if the device is attached, or 0 if the device is not attached

11.13.4.45 device_isConnected()

```
int device_isConnected ( )
```

Check the device connected status

Returns

DEVICE_DISCONNECT=0, or DEVICE_CONNECTED = 1

11.13.4.46 device_pingDevice()

```
int device_pingDevice ( )
```

Ping Device

Pings the reader. If connected, returns success. Otherwise, returns timeout.

Returns

RETURN_CODE: Values can be parsed with [device_getIDGStatusCodeString\(\)](#)

11.13.4.47 device_pollForToken()

```
int device_pollForToken (
    IN int timeout,
    OUT BYTE * respData,
    IN_OUT int * respDataLen )
```

Poll for Token

Polls for a PICC

Parameters

<i>timeout</i>	timeout in milliseconds, must be multiple of 10 milliseconds. 30, 120, 630, or 1150 for example.
<i>respData</i>	Response data will be stored in respData. 1 byte of card type, and the Serial Number (or the UID) of the PICC if available.
<i>respDataLen</i>	Length of systemCode.

Returns

RETURN_CODE: Values can be parsed with [device_getIDGStatusCodeString\(\)](#)

11.13.4.48 device_registerCameraCallBk()

```
void device_registerCameraCallBk (
    pCMR_callBack pCMRf )
```

To register the camera callback function to get the image data. (Pass NULL to disable the callback.)

11.13.4.49 device_registerCardStatusFrontSwitchCallBk()

```
void device_registerCardStatusFrontSwitchCallBk (
    pCSFS_callBack pCSFSf )
```

To register the card status and front switch callback function to get status. (Pass NULL to disable the callback.)

11.13.4.50 device_registerRKICallBk()

```
void device_registerRKICallBk (
    pRKI_callBack pRKIf )
```

To register the RKI callback function to get the RKI status. (Pass NULL to disable the callback.)

11.13.4.51 device_SendDataCommandNEO()

```
int device_SendDataCommandNEO (
    IN int cmd,
    IN int subCmd,
    IN BYTE * data,
    IN int dataLen,
    OUT BYTE * response,
    IN_OUT int * respLen )
```

Send a Command to device

Sends a command to the device .

Parameters

<i>cmd</i>	buffer of command to execute.
<i>cmdLen,the</i>	length of the buffer cmd.
<i>data</i>	buffer of IDG command data.
<i>dataLen,the</i>	length of the buffer data.
<i>response</i>	Response data
<i>respLen,the</i>	length of Response data

Returns

RETURN_CODE: Values can be parsed with [device_getIDGStatusCodeString\(\)](#) Send a Command to NEO device

Sends a command to the NEO device .

Parameters

<i>cmd</i>	command to execute.
<i>subCmd,sub</i>	command to execute.
<i>data</i>	buffer of NEO command data.
<i>dataLen,the</i>	length of the buffer data.
<i>response</i>	Response data
<i>respLen,the</i>	length of Response data

Returns

RETURN_CODE: Values can be parsed with [device_getIDGStatusCodeString\(\)](#)

11.13.4.52 device_setBurstMode()

```
int device_setBurstMode (
    IN BYTE mode )
```

Send Burst Mode

Sets the burst mode for the device.

Parameters

<i>mode</i>	0 = OFF, 1 = Always On, 2 = Auto Exit
-------------	---------------------------------------

Returns

success or error code. Values can be parsed with [device_getIDGStatusCodeString](#)

See also

ErrorCode

11.13.4.53 device_setCurrentDevice()

```
int device_setCurrentDevice (
    int deviceType )
```

Sets the current device to talk to

The connect status can be checked by [device_isConnected\(\)](#).

Parameters

<i>deviceType</i>	<p>Device to connect to</p> <pre>enum DEVICE_TYPE { IDT_DEVICE_UNKNOWN=0, IDT_DEVICE_AUGUSTA_HID, IDT_DEVICE_AUGUSTA_KB, IDT_DEVICE_AUGUSTA_S_HID, IDT_DEVICE_AUGUSTA_S_KB, IDT_DEVICE_AUGUSTA_S_TTK_HID, IDT_DEVICE_SPECTRUM_PRO, IDT_DEVICE_MINISMART_II, IDT_DEVICE_L80, IDT_DEVICE_L100, IDT_DEVICE_UNIPAY, IDT_DEVICE_UNIPAY_I_V, IDT_DEVICE_VP3300_AJ, IDT_DEVICE_KIOSK_III, IDT_DEVICE_KIOSK_III_S, IDT_DEVICE_PIP_READER, IDT_DEVICE_VENDI, IDT_DEVICE_VP3300_USB, IDT_DEVICE_UNIPAY_I_V_TTK, IDT_DEVICE_VP3300_BT, IDT_DEVICE_VP8800, IDT_DEVICE_SREDKEY2_HID, IDT_DEVICE_SREDKEY2_KB, IDT_DEVICE_NEO2, IDT_DEVICE_MINISMART_II_COM = IDT_DEVICE_NEO2+5, IDT_DEVICE_SPECTRUM_PRO_COM, IDT_DEVICE_KIOSK_III_COM, IDT_DEVICE_KIOSK_III_S_COM, IDT_DEVICE_PIP_READER_COM, IDT_DEVICE_VP3300_COM, IDT_DEVICE_NEO2_COM, IDT_DEVICE_MAX_DEVICES = IDT_DEVICE_NEO2_COM+5 };</pre>
-------------------	---

Returns

RETURN_CODE: 1: success, 0: failed

11.13.4.54 device_setMerchantRecord()

```
int device_setMerchantRecord (
    int index,
    int enabled,
    char * merchantID,
    char * merchantURL )
```

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

Returns

RETURN_CODE: Values can be parsed with [device_getIDGStatusCodeString\(\)](#)

11.13.4.55 device_setPollMode()

```
int device_setPollMode (
    IN BYTE mode )
```

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>	0 = Auto Poll, 1 = Poll On Demand
-------------	-----------------------------------

Returns

RETURN_CODE: Values can be parsed with [device_getIDGStatusCodeString\(\)](#)

11.13.4.56 device_setRKI_URL()

```
void device_setRKI_URL (
    IN char * rkiURL,
    IN int rkiURLLen )
```

Set the URL for RKI

Parameters

<i>rkiURL</i>	The URL for RKI (less than 100 characters). Pass NULL to reset to default URL
<i>rkiURLLen</i>	The length of rkiURL. Pass 0 to reset to default URL

Returns

11.13.4.57 device_setRTCDateTime()

```
int device_setRTCDateTime (
    IN BYTE * dateTime,
    IN int dateTimeLen )
```

set RTC date and time of the device

Parameters

<i>dateTime</i>	<dateTime data>=""> is: 6 byte data for YYMMDDHHMMSS in hex. For example 0x171003102547 stands for 2017 Oct 3rd 10:25:47
<i>dateTimeLen</i>	should be always 6 bytes

Returns

success or error code. Values can be parsed with [device_getResponseCodeString](#)

See also

[ErrorCode](#)

11.13.4.58 device_setSDKWaitTime()

```
void device_setSDKWaitTime (
    int waitTime )
```

Set SDK Wait Time

Set the SDK wait time for transactions

Parameters

<i>waitTime</i>	The SDK wait time for transaction in seconds. The Maximum is 2147483 seconds.
-----------------	---

11.13.4.59 device_setThreadStackSize()

```
void device_setThreadStackSize (
    int threadSize )
```

Set Thread Stack Size

Set the stack size setting for newly created threads

11.13.4.60 device_setTransactionExponent()

```
void device_setTransactionExponent (
    int exponent )
```

Sets the transaction exponent to be used with device_startTransaction. Default value is 2

Parameters

<i>exponent, The</i>	exponent to use when calling device_startTransaction
----------------------	--

11.13.4.61 device_startRKI()

```
int device_startRKI (
    IN const char * caPath,
    IN int isProduction )
```

Start remote key injection.

Parameters

<i>caPath</i>	The path to ca-certificates.crt It should be NULL, because the file is not used anymore.
<i>isProduction</i>	1: The reader is a production unit, 0: The reader is not a production unit

Returns

success or error code.

See also

ErrorCode

11.13.4.62 device_startTransaction()

```
int device_startTransaction (
    IN double amount,
    IN double amtOther,
    IN int type,
    IN const int _timeout,
    IN BYTE * tags,
    IN int tagsLen )
```

Start Transaction Request

Authorizes the transaction for an MSR/CTLS/ICC card

The tags will be returned in the callback routine.

Parameters

<i>amount</i>	Transaction amount value (tag value 9F02) - SEE IMPORTANT NOTE BELOW
<i>amtOther</i>	Other amount value, if any (tag value 9F03) - SEE IMPORTANT NOTE BELOW
<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. Example, tag 9F0C with amount 0x000000000100 would be 0x9F0C06000000000100 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>tagsLen</i>	The length of tags data buffer.

>>>>IMPORTANT: parameters for amount and amtOther MUST BE PASSED AS A DOUBLE VALUE WITH DECIMAL POINT. Example, do not pass 1, but instead pass 1.0 or 1.00. Otherwise, results will be unpredictable

Returns

RETURN_CODE: Values can be parsed with [device_getIDGStatusCodeString\(\)](#) Note: if auto poll is on, it will return the error IDG_P2_STATUS_CODE_COMMAND_NOT_ALLOWED

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 device_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

11.13.4.63 emv_activateTransaction()

```
int emv_activateTransaction (
    IN int timeout,
    IN BYTE * tags,
    IN int tagsLen,
    IN int forceOnline )
```

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>	Tags to be included in the request. Passed as a TLV stream. Example, tag 9F0C with amount 0x000000000100 would be 0x9F0C06000000000100
<i>tagsLen</i>	Length of tags
<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 = DFEE1A079F029F36959F37

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString >>>>IMPORTANT: parameters for amount and amtOther MUST BE PASSED AS A DOUBLE VALUE WITH DECIMAL POINT. Example, do not pass 1, but instead pass 1.0 or 1.00. Otherwise, results will be unpredictable

11.13.4.64 emv_allowFallback()

```
void emv_allowFallback (
    IN int allow )
```

Allow fallback for EMV transactions. Default is TRUE

Parameters

<i>allow</i>	TRUE = allow fallback, FALSE = don't allow fallback
--------------	---

11.13.4.65 emv_authenticateTransaction()

```
int emv_authenticateTransaction (
    IN BYTE * updatedTLV,
    IN int updatedTLVLen )
```

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 = DFEE1A079F029F36959F37
<i>updatedTLVLen</i>	

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString

11.13.4.66 emv_authenticateTransactionWithTimeout()

```
int emv_authenticateTransactionWithTimeout (
```

```

    IN int timeout,
    IN BYTE * updatedTLV,
    IN int updatedTLVLen )

```

Authenticate EMV Transaction Request with Timeout

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>timeout</i>	Timeout value in seconds.
<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 = DFEE1A079F029F36959F37
<i>updatedTLVLen</i>	

Returns

RETURN_CODE: Values can be parsed with `device_getResponseCodeString`

11.13.4.67 `emv_cancelTransaction()`

```
int emv_cancelTransaction ( )
```

Cancel EMV Transaction

Cancels the currently executing EMV transaction.

Returns

RETURN_CODE: Values can be parsed with [device_getResponseCodeString\(\)](#)

11.13.4.68 `emv_completeTransaction()`

```

int emv_completeTransaction (
    IN int commError,
    IN BYTE * authCode,
    IN int authCodeLen,
    IN BYTE * iad,
    IN int iadLen,
    IN BYTE * tlvScripts,
    IN int tlvScriptsLen,
    IN BYTE * tlv,
    IN int tlvLen )

```

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(1) if host was unreachable, or FALSE(0) 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>authCodeLen</i>	the length of <i>authCode</i>
<i>iad</i>	Issuer Authentication Data, if any. Example 0x11223344556677883030 (tag value 91).
<i>iadLen</i>	the length of <i>iadLen</i>
<i>tlvScripts</i>	71/72 scripts, if any
<i>tlvScriptsLen</i>	the length of <i>tlvScriptsLen</i>
<i>tlv</i>	Additional TLV data to return with transaction results (if any)
<i>tlvLen</i>	the length of <i>tlv</i>

Returns

RETURN_CODE: Values can be parsed with `device_getResponseCodeString`

11.13.4.69 `emv_getAutoAuthenticateTransaction()`

```
int emv_getAutoAuthenticateTransaction ( )
```

Gets auto authenticate value for EMV transactions.

Returns

RETURN_CODE: TRUE = auto authenticate, FALSE = manually authenticate

11.13.4.70 `emv_getAutoCompleteTransaction()`

```
int emv_getAutoCompleteTransaction ( )
```

Gets auto complete value for EMV transactions.

Returns

RETURN_CODE: TRUE = auto complete, FALSE = manually complete

11.13.4.71 `emv_registerCallBk()`

```
void emv_registerCallBk (
    pEMV_callback pEMVf )
```

To register the emv callback function to get the EMV processing response. (Pass NULL to disable the callback.)

11.13.4.72 emv_removeAllApplicationData()

```
int emv_removeAllApplicationData ( )
```

Remove All Application Data

Removes all the Application Data

Returns

RETURN_CODE: Values can be parsed with [device_getIDGStatusCodeString\(\)](#)

11.13.4.73 emv_removeAllCAPK()

```
int emv_removeAllCAPK ( )
```

Remove All Certificate Authority Public Key

Removes all the CAPK

Returns

RETURN_CODE: Values can be parsed with [device_getIDGStatusCodeString\(\)](#)

11.13.4.74 emv_removeAllCRL()

```
int emv_removeAllCRL ( )
```

Remove All Certificate Revocation List Entries

Removes all CRLentry entries

Returns

RETURN_CODE: Values can be parsed with [device_getResponseCodeString\(\)](#)

11.13.4.75 emv_removeApplicationData()

```
int emv_removeApplicationData (
    IN BYTE * AID,
    IN int AIDLen )
```

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>AIDLen</i>	the length of AID data buffer

Returns

RETURN_CODE: Values can be parsed with [device_getIDGStatusCodeString\(\)](#)

11.13.4.76 emv_removeCAPK()

```
int emv_removeCAPK (
    IN BYTE * capk,
    IN int capkLen )
```

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>capkLen</i>	the length of capk data buffer

Returns

RETURN_CODE: Values can be parsed with [device_getIDGStatusCodeString\(\)](#)

11.13.4.77 emv_removeCRL()

```
int emv_removeCRL (
    IN BYTE * list,
    IN int lsLen )
```

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>lsLen</i>	the length of list data buffer

Returns

RETURN_CODE: Values can be parsed with [device_getResponseCodeString\(\)](#)

11.13.4.78 emv_retrieveAIDList()

```
int emv_retrieveAIDList (
    OUT BYTE * AIDList,
    IN_OUT int * AIDListLen )
```

Retrieve AID list

Returns all the AID names installed on the terminal for CTLS. .

Parameters

<i>AIDList</i>	array of AID name byte arrays
<i>AIDListLen</i>	the length of AIDList array buffer

Returns

RETURN_CODE: Values can be parsed with [device_getIDGStatusCodeString\(\)](#)

11.13.4.79 emv_retrieveApplicationData()

```
int emv_retrieveApplicationData (
    IN BYTE * AID,
    IN int AIDLen,
    OUT BYTE * tlv,
    IN_OUT int * tlvLen )
```

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>AIDLen</i>	the length of AID data buffer.
<i>tlv</i>	The TLV elements of the requested AID
<i>tlvLen</i>	the length of tlv data buffer.

Returns

RETURN_CODE: Values can be parsed with [device_getIDGStatusCodeString\(\)](#)

11.13.4.80 emv_retrieveCAPK()

```
int emv_retrieveCAPK (
    IN BYTE * capk,
    IN int capkLen,
    OUT BYTE * key,
    IN_OUT int * keyLen )
```

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>capkLen</i>	the length of capk data buffer

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>keyLen</i>	<p>the length of key data buffer</p> <ul style="list-style-type: none"> •

Returns

RETURN_CODE: Values can be parsed with [device_getIDGStatusCodeString\(\)](#)

11.13.4.81 emv_retrieveCAPKList()

```
int emv_retrieveCAPKList (
    OUT BYTE * keys,
    IN_OUT int * keysLen )
```

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>keysLen</i>	the length of keys data buffer

Returns

RETURN_CODE: Values can be parsed with [device_getIDGStatusCodeString\(\)](#)

11.13.4.82 emv_retrieveCRL()

```
int emv_retrieveCRL (
    OUT BYTE * list,
    IN_OUT int * lssLen )
```

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>lssLen</i>	the length of list data buffer

Returns

RETURN_CODE: Values can be parsed with [device_getResponseCodeString\(\)](#)

11.13.4.83 emv_retrieveTerminalData()

```
int emv_retrieveTerminalData (
    OUT BYTE * tlv,
    IN_OUT int * tlvLen )
```

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>tlvLen</i>	the length of tlv data buffer

Returns

RETURN_CODE: Values can be parsed with [device_getIDGStatusCodeString\(\)](#)

11.13.4.84 emv_setApplicationData()

```
int emv_setApplicationData (
    IN BYTE * name,
    IN int nameLen,
    IN BYTE * tlv,
    IN int tlvLen )
```

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>nameLen</i>	the length of name data buffer of Application name,
<i>tlv</i>	Application data in TLV format
<i>tlvLen</i>	the length of tlv data buffer

Returns

RETURN_CODE: Values can be parsed with [device_getResponseCodeString\(\)](#)

11.13.4.85 emv_setApplicationDataTLV()

```
int emv_setApplicationDataTLV (
    IN BYTE * tlv,
    IN int tlvLen )
```

Set Application Data by TLV

Sets the Application Data as specified by the TLV data

Parameters

<i>tlv</i>	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) Example valid TLV, for Group #2, AID a0000000035010: "dfee2d01029f0607a0000000051010ffe10101ffe50110ffe30114ffe20106"
<i>tlvLen</i>	the length of tlv data buffer

Returns

RETURN_CODE: Values can be parsed with [device_getResponseCodeString\(\)](#)

11.13.4.86 emv_setAutoAuthenticateTransaction()

```
void emv_setAutoAuthenticateTransaction (
    IN int authenticate )
```

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

Parameters

<i>authenticate</i>	TRUE = auto authenticate, FALSE = manually authenticate
---------------------	---

11.13.4.87 emv_setAutoCompleteTransaction()

```
void emv_setAutoCompleteTransaction (
    IN int complete )
```

Enables complete for EMV transactions. If a `emv_authenticateTransaction` results in code 0x0004 (go online), then `emv_completeTransaction` can automatically execute if parameter is set to TRUE

Parameters

<i>complete</i>	TRUE = auto complete, FALSE = manually complete
-----------------	---

11.13.4.88 emv_setCAPK()

```
int emv_setCAPK (
    IN BYTE * capk,
    IN int capkLen )
```

Set Certificate Authority Public Key

Sets the CAPK as specified by the CAKey structure

Parameters

<i>capk</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>capkLen</i>	the length of capk data buffer

ReturnsRETURN_CODE: Values can be parsed with [device_getIDGStatusCodeString\(\)](#)**11.13.4.89 emv_setCRL()**

```
int emv_setCRL (
    IN BYTE * list,
    IN int lsLen )
```

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>lsLen</i>	the length of list data buffer

ReturnsRETURN_CODE: Values can be parsed with [device_getResponseCodeString\(\)](#)

11.13.4.90 emv_setTerminalData()

```
int emv_setTerminalData (
    IN BYTE * tlv,
    IN int tlvLen )
```

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 file
<i>tlvLen</i>	the length of tlv data buffer

Return values

<i>RETURN_CODE</i>	Values can be parsed with device_getIDGStatusCodeString()
--------------------	---

11.13.4.91 emv_setTerminalMajorConfiguration()

```
int emv_setTerminalMajorConfiguration (
    IN int configuration )
```

Sets the terminal major configuration in ICS .

Parameters

<i>configuration</i>	A configuration value, range 1-23 <ul style="list-style-type: none">• 1 = 1C• 2 = 2C• 3 = 3C• 4 = 4C• 5 = 5C ...• 23 = 23C
----------------------	---

Returns

RETURN_CODE: Values can be parsed with [device_getResponseCodeString\(\)](#)

11.13.4.92 emv_setTransactionParameters()

```
void emv_setTransactionParameters (
    IN double amount,
```

```

    IN double amtOther,
    IN int type,
    IN int timeout,
    IN BYTE * tags,
    IN int tagsLen )

```

Set EMV Transaction Parameters

Set the parameters to be used on EMV transactions for an ICC card when Auto Poll is on

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 (Maximum Length = 500 bytes). Passed as a string. Example, tag 9F0C with amount 0x000000000100 would be "9F0C06000000000100" 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 = DFEE1A079F029F36959F37
<i>tagsLen</i>	the length of tags

11.13.4.93 emv_startTransaction()

```

int emv_startTransaction (
    IN double amount,
    IN double amtOther,
    IN int exponent,
    IN int type,
    IN int timeout,
    IN BYTE * tags,
    IN int tagsLen,
    IN int forceOnline )

```

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) - SEE IMPORTANT NOTE BELOW
<i>amtOther</i>	Other amount value, if any (tag value 9F03) - SEE IMPORTANT NOTE BELOW
<i>exponent</i>	Number of characters after decimal 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 stream. Example, tag 9F0C with amount 0x000000000100 would be 0x9F0C06000000000100 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>tagsLen</i>	Length of tags

Parameters

<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 = DFEE1A079F029F36959F37
--------------------	--

Returns

RETURN_CODE: Values can be parsed with `device_getResponseCodeString` >>>>IMPORTANT: parameters for amount and amtOther MUST BE PASSED AS A DOUBLE VALUE WITH DECIMAL POINT. Example, do not pass 1, but instead pass 1.0 or 1.00. Otherwise, results will be unpredictable

11.13.4.94 executeTransaction()

```
int executeTransaction (
    WorldPayData * data,
    pWP_callback wpCallback,
    int requestOnly )
```

executeTransaction Requests 3 sets of public keys: encrypting Keys, signing/validating keys, signing/validating 3rd party apps

Parameters

<i>data</i>	WorldPay data object
<i>wpCallback</i>	WorldPay callback
<i>requestOnly</i>	<ul style="list-style-type: none"> • 0 = send transaction and return response, • 1 = send transaction and return both request and response, • 2 = do not send transaction, instead return request

Returns

RETURN_CODE: Values can be parsed with [device_getIDGStatusCodeString\(\)](#)

11.13.4.95 executeTransaction_WorldNet()

```
int executeTransaction_WorldNet (
    WorldNetData * data,
    pWN_callback wnCallback,
    int requestOnly )
```

executeTransaction_WorldNet Requests 3 sets of public keys: encrypting Keys, signing/validating keys, signing/validating 3rd party apps

Parameters

<i>data</i>	WorldNet data object
-------------	----------------------

Parameters

<i>wnCallback</i>	WorldNet callback
<i>requestOnly</i>	<ul style="list-style-type: none"> • 0 = send transaction and return response, • 1 = send transaction and return both request and response, • 2 = do not send transaction, instead return request

Returns

RETURN_CODE: Values can be parsed with [device_getIDGStatusCodeString\(\)](#)

11.13.4.96 forwardTransaction()

```
int forwardTransaction (
    IN pWP_callback wpCallback,
    IN char * forwardID,
    IN int forwardIDLen,
    IN char * password,
    IN int passwordLen,
    IN int bypassProcessing )
```

forwardTransaction Send the saved data to WorldPay and complete the transaction.

Parameters

<i>wpCallback</i>	WPCallback is the callback to send the results. Should the the same as executeTransaction callback.
<i>forwardID</i>	= ID, which could be either unique ID or Memo.
<i>forwardIDLen</i>	The length of forwardID.
<i>password</i>	= password. If null/blank, no password.
<i>passwordLen</i>	The length of passwordLen.
<i>bypassProcess</i>	= true means read the file from disk, but don't send to WorldPay, then delete the transaction as if you did send to WorldPay. The purpose of this is to allow them to delete transactions from the storage without sending to WorldPay.

Returns

RETURN_CODE: Values can be parsed with [device_getIDGStatusCodeString\(\)](#)

11.13.4.97 forwardTransaction_WorldNet()

```
int forwardTransaction_WorldNet (
    IN char * apiKey,
    IN int apiKeyLen,
    IN pWN_callback wnCallback,
    IN char * forwardID,
    IN int forwardIDLen,
```

```

    IN char * password,
    IN int passwordLen,
    IN int bypassProcessing )

```

forwardTransaction_WorldNet Send the saved data to WorldNet and complete the transaction.

Parameters

<i>wnCallback</i>	WNCallback is the callback to send the results. Should be the same as executeTransaction_WorldNet callback.
<i>forwardID</i>	= ID, which could be either unique ID or Memo.
<i>forwardIDLen</i>	The length of forwardID.
<i>password</i>	= password. If null/blank, no password.
<i>passwordLen</i>	The length of passwordLen.
<i>bypassProcess</i>	= true means read the file from disk, but don't send to WorldPay, then delete the transaction as if you did send to WorldPay. The purpose of this is to allow them to delete transactions from the storage without sending to WorldPay.

Returns

RETURN_CODE: Values can be parsed with [device_getIDGStatusCodeString\(\)](#)

11.13.4.98 `icc_exchangeAPDU()`

```

int icc_exchangeAPDU (
    IN BYTE * c_APDU,
    IN int cLen,
    OUT BYTE * reData,
    IN_OUT int * reLen )

```

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>cLen</i>	APDU data packet length
<i>reData</i>	Unencrypted APDU response
<i>reLen</i>	Unencrypted APDU response data length

Returns

RETURN_CODE: Values can be parsed with [device_getIDGStatusCodeString](#)

11.13.4.99 `icc_getICCRReaderStatus()`

```

int icc_getICCRReaderStatus (
    OUT BYTE * status )

```

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
---------------	--

Returns

RETURN_CODE: Values can be parsed with `device_getIDGStatusCodeString`

11.13.4.100 `icc_powerOffICC()`

```
int icc_powerOffICC ( )
```

Power Off ICC

Powers down the ICC

Returns

RETURN_CODE: Values can be parsed with `device_getIDGStatusCodeString()`

If Success, empty If Failure, ASCII encoded data of error string

11.13.4.101 `icc_powerOnICC()`

```
int icc_powerOnICC (
    OUT BYTE * ATR,
    IN_OUT int * inLen )
```

Power On ICC

Power up the currently selected microprocessor card in the ICC reader

Parameters

<i>ATR,the</i>	ATR data response when succeeded power on ICC,
<i>inLen,the</i>	length of ATR data

Returns

RETURN_CODE: Values can be parsed with `device_getIDGStatusCodeString()`

11.13.4.102 `msr_cancelMSRSwipe()`

```
int msr_cancelMSRSwipe ( )
```

Disable MSR Swipe Cancels MSR swipe request.

Returns

RETURN_CODE: Values can be parsed with `device_getIDGStatusCodeString()`

11.13.4.103 msr_registerCallBk()

```
void msr_registerCallBk (
    pMSR_callback pMSRf )
```

To register the msr callback function to get the MSR card data. (Pass NULL to disable the callback.)

11.13.4.104 msr_registerCallBkp()

```
void msr_registerCallBkp (
    pMSR_callbackp pMSRf )
```

To register the msr callback function to get the MSR card data pointer. (Pass NULL to disable the callback.)

11.13.4.105 msr_startMSRSwipe()

```
int msr_startMSRSwipe (
    IN int _timeout )
```

Start 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
----------------	---------------------

Returns

RETURN_CODE: Values can be parsed with [device_getIDGStatusCodeString\(\)](#) Note: if auto poll mode is on, it will return command not allowed error

11.13.4.106 parseMSRData()

```
void parseMSRData (
    IN BYTE * resData,
    IN int resLen,
    IN_OUT IDTMSRData * cardData )
```

Parser the MSR data from the buffer into IDTMSTData structure

Parameters

<i>resData</i>	MSR card data buffer
<i>resLen</i>	the length of resData
<i>cardData</i>	the parser result with IDTMSTData structure

11.13.4.107 pin_registerCallBk()

```
void pin_registerCallBk (
    pPIN_callback pPINf )
```

To register the pin callback function to get the PINPad data. (Pass NULL to disable the callback.)

11.13.4.108 registerHotplugCallBk()

```
void registerHotplugCallBk (
    pMessageHotplug pMsgHotplug )
```

To register the USB HID hot-plug callback function which implemented in the application to monitor the hotplug message from the SDK.

11.13.4.109 registerLogCallBk()

```
void registerLogCallBk (
    pSendDataLog pFSend,
    pReadDataLog pFRead )
```

To register the log callback function which implemented in the application to monitor sending/reading data between application and reader.

11.13.4.110 SDK_Version()

```
char* SDK_Version ( )
```

To Get SDK version

Returns

return the SDK version string

11.13.4.111 setAbsoluteLibraryPath()

```
int setAbsoluteLibraryPath (
    const char * absoluteLibraryPath )
```

Set the path to use when searching for ID TECH's libraries. If this is not set, the libraries will be searched for with the system's default procedures.

Parameters

<i>absoluteLibraryPath</i>	The absolute path to ID TECH's libraries.
----------------------------	---

Returns

RETURN_CODE: Values can be parsed with [device_getIDGStatusCodeString\(\)](#)

11.14 Source_C/libIDT_VP3300_BT.h File Reference

VP3300 BT API.

```
#include "IDTDef.h"
```

Macros

- #define [IN](#)
- #define [OUT](#)

- `#define IN_OUT`

Typedefs

- `typedef void(* pMessageHotplug) (int, int)`
- `typedef void(* pSendDataLog) (BYTE *, int)`
- `typedef void(* pReadDataLog) (BYTE *, int)`
- `typedef void(* pEMV_callBack) (int, int, BYTE *, int, IDTTransactionData *, EMV_Callback *, int)`
- `typedef void(* pMSR_callBack) (int, IDTMSRData)`
- `typedef void(* pMSR_callBackp) (int, IDTMSRData *)`
- `typedef void(* pPIN_callBack) (int, IDTPINData *)`
- `typedef void(* pCMR_callBack) (int, IDTCMRData *)`
- `typedef void(* pCSFS_callBack) (BYTE status)`
- `typedef void(* pRKI_callBack) (int, char *)`
- `typedef void(* ftpComm_callBack) (int, int, int)`
- `typedef void(* httpComm_callBack) (BYTE *, int)`
- `typedef void(* v4Comm_callBack) (BYTE, BYTE, BYTE *, int)`
- `typedef void(* pWP_callBack) (char *, int, int)`
- `typedef void(* pWN_callBack) (char *, int, int)`

Functions

- `void registerHotplugCallBk (pMessageHotplug pMsgHotplug)`
- `void registerLogCallBk (pSendDataLog pFSend, pReadDataLog pFRead)`
- `void device_registerRKICallBk (pRKI_callBack pRKIf)`
- `void emv_registerCallBk (pEMV_callBack pEMVf)`
- `void msr_registerCallBk (pMSR_callBack pMSRf)`
- `void msr_registerCallBkp (pMSR_callBackp pMSRf)`
- `void ctls_registerCallBk (pMSR_callBack pCTLSf)`
- `void ctls_registerCallBkp (pMSR_callBackp pCTLSf)`
- `void pin_registerCallBk (pPIN_callBack pPINf)`
- `void device_registerCameraCallBk (pCMR_callBack pCMRf)`
- `void device_registerCardStatusFrontSwitchCallBk (pCSFS_callBack pCSFSf)`
- `void comm_registerHTTPCallback (httpComm_callBack cBack)`
- `void comm_registerV4Callback (v4Comm_callBack cBack)`
- `char * SDK_Version ()`
- `int setAbsoluteLibraryPath (const char *absoluteLibraryPath)`
- `int device_init ()`
- `int device_setCurrentDevice (int deviceType)`
- `int device_close ()`
- `void device_getIDGStatusCodeString (IN int returnCode, OUT char *despcrition)`
- `int device_isConnected ()`
- `int device_isAttached (int deviceType)`
- `int device_getFirmwareVersion (OUT char *firmwareVersion)`
- `int device_getFirmwareVersion_Len (OUT char *firmwareVersion, IN_OUT int *firmwareVersionLen)`
- `int device_pingDevice ()`
- `int device_controlUserInterface (IN BYTE *values)`
- `int device_getCurrentDeviceType ()`
- `int device_SendDataCommandNEO (IN int cmd, IN int subCmd, IN BYTE *data, IN int dataLen, OUT BYTE *response, IN_OUT int *respLen)`
- `int device_enablePassThrough (int enablePassThrough)`
- `int device_setBurstMode (IN BYTE mode)`
- `int device_setPollMode (IN BYTE mode)`
- `int device_setMerchantRecord (int index, int enabled, char *merchantID, char *merchantURL)`

- int [device_pollForToken](#) (IN int timeout, OUT BYTE *respData, IN_OUT int *respDataLen)
- int [device_getMerchantRecord](#) (IN int index, OUT BYTE *record)
- int [device_getMerchantRecord_Len](#) (IN int index, OUT BYTE *record, IN_OUT int *recordLen)
- int [device_getTransactionResults](#) (IDTMSRData *cardData)
- int [device_startTransaction](#) (IN double amount, IN double amtOther, IN int type, IN const int _timeout, IN BYTE *tags, IN int tagsLen)
- void [device_setTransactionExponent](#) (int exponent)
- int [device_activateTransaction](#) (IN const int _timeout, IN BYTE *tags, IN int tagsLen)
- int [device_cancelTransaction](#) ()
- int [device_getRTCDatetime](#) (IN BYTE *dateTime, IN_OUT int *dateTimeLen)
- int [device_setRTCDatetime](#) (IN BYTE *dateTime, IN int dateTimeLen)
- int [device_startRKI](#) (IN const char *caPath, IN int isProduction)
- void [device_setRKI_URL](#) (IN char *rkiURL, IN int rkiURLLen)
- int [device_getSDKWaitTime](#) ()
- void [device_setSDKWaitTime](#) (int waitTime)
- int [device_getThreadStackSize](#) ()
- void [device_setThreadStackSize](#) (int threadSize)
- int [config_getSerialNumber](#) (OUT char *sNumber)
- int [config_getSerialNumber_Len](#) (OUT char *sNumber, IN_OUT int *sNumberLen)
- int [ctls_startTransaction](#) (IN double amount, IN double amtOther, IN int type, IN const int _timeout, IN BYTE *tags, IN int tagsLen)
- int [ctls_activateTransaction](#) (IN const int _timeout, IN BYTE *tags, IN int tagsLen)
- int [ctls_cancelTransaction](#) ()
- int [ctls_retrieveApplicationData](#) (IN BYTE *AID, IN int AIDLen, OUT BYTE *tlv, IN_OUT int *tlvLen)
- int [ctls_setApplicationData](#) (IN BYTE *tlv, IN int tlvLen)
- int [ctls_removeApplicationData](#) (IN BYTE *AID, IN int AIDLen)
- int [ctls_removeAllApplicationData](#) ()
- int [ctls_retrieveAIDList](#) (OUT BYTE *AIDList, IN_OUT int *AIDListLen)
- int [ctls_retrieveTerminalData](#) (OUT BYTE *tlv, IN_OUT int *tlvLen)
- int [ctls_setTerminalData](#) (IN BYTE *tlv, IN int tlvLen)
- int [ctls_retrieveCAPK](#) (IN BYTE *capk, IN int capkLen, OUT BYTE *key, IN_OUT int *keyLen)
- int [ctls_setCAPK](#) (IN BYTE *capk, IN int capkLen)
- int [ctls_removeCAPK](#) (IN BYTE *capk, IN int capkLen)
- int [ctls_removeAllCAPK](#) ()
- int [ctls_retrieveCAPKList](#) (OUT BYTE *keys, IN_OUT int *keysLen)
- int [ctls_setConfigurationGroup](#) (IN BYTE *tlv, IN int tlvLen)
- int [ctls_getConfigurationGroup](#) (IN int group, OUT BYTE *tlv, OUT int *tlvLen)
- int [ctls_getAllConfigurationGroups](#) (OUT BYTE *tlv, IN_OUT int *tlvLen)
- int [ctls_removeConfigurationGroup](#) (int group)
- void [emv_allowFallback](#) (IN int allow)
- void [emv_setAutoAuthenticateTransaction](#) (IN int authenticate)
- void [emv_setAutoCompleteTransaction](#) (IN int complete)
- int [emv_getAutoAuthenticateTransaction](#) ()
- int [emv_getAutoCompleteTransaction](#) ()
- void [emv_setTransactionParameters](#) (IN double amount, IN double amtOther, IN int type, IN int timeout, IN BYTE *tags, IN int tagsLen)
- int [emv_startTransaction](#) (IN double amount, IN double amtOther, IN int exponent, IN int type, IN int timeout, IN BYTE *tags, IN int tagsLen, IN int forceOnline)
- int [emv_activateTransaction](#) (IN int timeout, IN BYTE *tags, IN int tagsLen, IN int forceOnline)
- int [emv_authenticateTransaction](#) (IN BYTE *updatedTLV, IN int updatedTLVLen)
- int [emv_authenticateTransactionWithTimeout](#) (IN int timeout, IN BYTE *updatedTLV, IN int updatedTLVLen)
- int [emv_completeTransaction](#) (IN int commError, IN BYTE *authCode, IN int authCodeLen, IN BYTE *iad, IN int iadLen, IN BYTE *tlvScripts, IN int tlvScriptsLen, IN BYTE *tlv, IN int tlvLen)
- int [emv_cancelTransaction](#) ()
- int [emv_retrieveApplicationData](#) (IN BYTE *AID, IN int AIDLen, OUT BYTE *tlv, IN_OUT int *tlvLen)

- int [emv_setApplicationData](#) (IN BYTE *name, IN int nameLen, IN BYTE *tlv, IN int tlvLen)
- int [emv_setApplicationDataTLV](#) (IN BYTE *tlv, IN int tlvLen)
- int [emv_removeApplicationData](#) (IN BYTE *AID, IN int AIDLen)
- int [emv_removeAllApplicationData](#) ()
- int [emv_retrieveAIDList](#) (OUT BYTE *AIDList, IN_OUT int *AIDListLen)
- int [emv_retrieveTerminalData](#) (OUT BYTE *tlv, IN_OUT int *tlvLen)
- int [emv_setTerminalData](#) (IN BYTE *tlv, IN int tlvLen)
- int [emv_setTerminalMajorConfiguration](#) (IN int configuration)
- int [emv_retrieveCAPK](#) (IN BYTE *capk, IN int capkLen, OUT BYTE *key, IN_OUT int *keyLen)
- int [emv_setCAPK](#) (IN BYTE *capk, IN int capkLen)
- int [emv_removeCAPK](#) (IN BYTE *capk, IN int capkLen)
- int [emv_removeAllCAPK](#) ()
- int [emv_retrieveCAPKList](#) (OUT BYTE *keys, IN_OUT int *keysLen)
- int [emv_retrieveCRL](#) (OUT BYTE *list, IN_OUT int *lssLen)
- int [emv_setCRL](#) (IN BYTE *list, IN int lssLen)
- int [emv_removeCRL](#) (IN BYTE *list, IN int lssLen)
- int [emv_removeAllCRL](#) ()
- int [icc_getICCReaderStatus](#) (OUT BYTE *status)
- int [icc_powerOnICC](#) (OUT BYTE *ATR, IN_OUT int *inLen)
- int [icc_powerOffICC](#) ()
- int [icc_exchangeAPDU](#) (IN BYTE *c_APDU, IN int cLen, OUT BYTE *reData, IN_OUT int *reLen)
- int [msr_cancelMSRSwipe](#) ()
- int [msr_startMSRSwipe](#) (IN int _timeout)
- int [executeTransaction](#) (WorldPayData *data, [pWP_callback](#) wpCallback, int requestOnly)
- int [forwardTransaction](#) (IN [pWP_callback](#) wpCallback, IN char *forwardID, IN int forwardIDLen, IN char *password, IN int passwordLen, IN int bypassProcessing)
- int [cancelWorldPay](#) ()
- int [executeTransaction_WorldNet](#) (WorldNetData *data, [pWN_callback](#) wnCallback, int requestOnly)
- int [forwardTransaction_WorldNet](#) (IN char *apiKey, IN int apiKeyLen, IN [pWN_callback](#) wnCallback, IN char *forwardID, IN int forwardIDLen, IN char *password, IN int passwordLen, IN int bypassProcessing)
- int [cancelWorldNet](#) ()
- void [parseMSRData](#) (IN BYTE *resData, IN int resLen, IN_OUT IDTMSRData *cardData)

11.14.1 Detailed Description

VP3300 BT API.

VP3300 BT Global API methods.

11.14.2 Macro Definition Documentation

11.14.2.1 IN

```
#define IN
```

INPUT parameter.

11.14.2.2 IN_OUT

```
#define IN_OUT
```

INPUT / OUTPUT PARAMETER.

11.14.2.3 OUT

```
#define OUT
```

OUTPUT parameter.

11.14.3 Typedef Documentation

11.14.3.1 ftpComm_callBack

```
typedef void(* ftpComm_callBack) (int, int, int)
```

Define the comm callback function to get FTP file transfer status

It should be passed as a parameter in a FTP request, Signature (int, int, int) = response code, current block, total blocks
 RESPONSE CODES: 100 = FILE DOWNLOAD STARTED 101 = FILE BLOCK XX OF XX RECEIVED 102 = FILE DOWNLOAD COMPLETED 103 = FILE DOWNLOAD TERMINATED PREMATURELY

11.14.3.2 httpComm_callBack

```
typedef void(* httpComm_callBack) (BYTE *, int)
```

Define the comm callback function to get the async url data

It should be registered using the comm_registerHTTPCallback

11.14.3.3 pCMR_callBack

```
typedef void(* pCMR_callBack) (int, IDTCMRData *)
```

Define the camera callback function to get the image data

It should be registered using the device_registerCameraCallBk,

11.14.3.4 pCSFS_callBack

```
typedef void(* pCSFS_callBack) (BYTE status)
```

Define the card status and front switch callback function to get card and front switch status

It should be registered using the device_registerCardStatusFrontSwitchCallBk,

11.14.3.5 pEMV_callBack

```
typedef void(* pEMV_callBack) (int, int, BYTE *, int, IDTTransactionData *, EMV_Callback *, int)
```

Define the EMV callback function to get the transaction message/data/result.

It should be registered using the emv_registerCallBk,

11.14.3.6 pMessageHotplug

```
typedef void(* pMessageHotplug) (int, int)
```

Define the USB hot-plug callback function to monitor the info when plug in/out the reader.

It should be registered using the registerHotplugCallBk, The first integer parameter is device type, and the second integer parameter is either 0: Device Plugged Out or 1: Device Plugged In

11.14.3.7 pMSR_callBack

```
typedef void(* pMSR_callBack) (int, IDTMSRData)
```

Define the MSR callback function to get the MSR card data

It should be registered using the msr_registerCallBk, this callback function is for backward compatibility

11.14.3.8 pMSR_callBackp

```
typedef void(* pMSR_callBackp) (int, IDTMSRData *)
```

Define the MSR callback function to get pointer to the MSR card data

It should be registered using the msr_registerCallBk, this callback function is recommended instead of pMSR_callBack

11.14.3.9 pPIN_callBack

```
typedef void(* pPIN_callBack) (int, IDTPINData *)
```

Define the PINPad callback function to get the input PIN Pad data

It should be registered using the pin_registerCallBk,

11.14.3.10 pReadDataLog

```
typedef void(* pReadDataLog) (BYTE *, int)
```

Define the read response callback function to monitor the reading response from the reader.

It should be registered using the registerLogCallBk,

11.14.3.11 pRKI_callBack

```
typedef void(* pRKI_callBack) (int, char *)
```

Define the RKI callback function to get the status of the RKI

It should be registered using the device_registerRKICallBk,

11.14.3.12 pSendDataLog

```
typedef void(* pSendDataLog) (BYTE *, int)
```

Define the send command callback function to monitor the sending command into the reader.

It should be registered using the registerLogCallBk,

11.14.3.13 pWN_callBack

```
typedef void(* pWN_callBack) (char *, int, int)
```

Define the Worldnet callback function to get the transaction message/data/result.

11.14.3.14 pWP_callBack

```
typedef void(* pWP_callBack) (char *, int, int)
```

Define the Worldpay callback function to get the transaction message/data/result.

11.14.3.15 v4Comm_callBack

```
typedef void(* v4Comm_callBack) (BYTE, BYTE, BYTE *, int)
```

Define the comm callback function to receive the V4 Protocol packets received by the device from an external source (IP/USB/RS-232) It should be registered using the `comm_registerV4Callback`, Data callback will contain command, sub-command, and data from V4 packet

11.14.4 Function Documentation

11.14.4.1 cancelWorldNet()

```
int cancelWorldNet ( )
```

Cancels WorldNet transaction.

Returns

RETURN_CODE: Values can be parsed with [device_getIDGStatusCodeString\(\)](#)

11.14.4.2 cancelWorldPay()

```
int cancelWorldPay ( )
```

Cancels WorldPay transaction.

Returns

RETURN_CODE: Values can be parsed with [device_getIDGStatusCodeString\(\)](#)

11.14.4.3 comm_registerHTTPCallback()

```
void comm_registerHTTPCallback (
    httpComm_callBack cBack )
```

Register Comm HTTP Async Callback

Parameters

<i>cBack</i>	- HTTP Comm callback
--------------	----------------------

11.14.4.4 comm_registerV4Callback()

```
void comm_registerV4Callback (
    v4Comm_callBack cBack )
```

Register External V4 Protocol commands Callback

Parameters

<i>cBack</i>	- V4 Protocol Comm callback
--------------	-----------------------------

11.14.4.5 config_getSerialNumber()

```
int config_getSerialNumber (
    OUT char * sNumber )
```

DEPRECATED : please use [config_getSerialNumber_Len](#)(OUT char* sNumber, IN_OUT int *sNumberLen)

Polls device for Serial Number

Parameters

<i>sNumber</i>	Returns Serial Number; needs to have at least 64 bytes of memory
----------------	--

Returns

RETURN_CODE: Values can be parsed with [device_getIDGStatusCodeString](#)

11.14.4.6 config_getSerialNumber_Len()

```
int config_getSerialNumber_Len (
    OUT char * sNumber,
    IN_OUT int * sNumberLen )
```

Polls device for Serial Number

Parameters

<i>sNumber</i>	Returns Serial Number
<i>sNumberLen</i>	Length of Serial Number

Returns

RETURN_CODE: Values can be parsed with [device_getIDGStatusCodeString](#)

11.14.4.7 ctls_activateTransaction()

```
int ctls_activateTransaction (
    IN const int _timeout,
    IN BYTE * tags,
    IN int tagsLen )
```

Start CTLS Transaction Request

Authorizes the CTLS 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 stream. Example, tag 9F0C with amount 0x000000000100 would be 0x9F0C06000000000100
<i>tagsLen</i>	The length of tags data buffer.

>>>>>IMPORTANT: parameters for amount and amtOther MUST BE PASSED AS A DOUBLE VALUE WITH DECIMAL POINT. Example, do not pass 1, but instead pass 1.0 or 1.00. Otherwise, results will be unpredictable

Returns

RETURN_CODE: Values can be parsed with [device_getIDGStatusCodeString\(\)](#) Note: if auto poll is on, it will return the error IDG_P2_STATUS_CODE_COMMAND_NOT_ALLOWED

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

11.14.4.8 ctls_cancelTransaction()

```
int ctls_cancelTransaction ( )
```

Cancel EMV Transaction

Cancels the currently executing EMV transaction.

Returns

RETURN_CODE: Values can be parsed with [device_getIDGStatusCodeString\(\)](#)

11.14.4.9 ctls_getAllConfigurationGroups()

```
int ctls_getAllConfigurationGroups (
    OUT BYTE * tlv,
    IN_OUT int * tlvLen )
```

Retrieve All Configuration Groups

Returns all the Configuration Groups installed on the terminal for CTLS

Parameters

<i>tlv</i>	The TLV elements data
<i>tlvLen</i>	the length of tlv data buffer.

Returns

RETURN_CODE: Values can be parsed with [device_getIDGStatusCodeString\(\)](#)

11.14.4.10 ctls_getConfigurationGroup()

```
int ctls_getConfigurationGroup (
    IN int group,
    OUT BYTE * tlv,
    OUT int * tlvLen )
```

Get Configuration Group

Retrieves the Configuration for the specified Group.

Parameters

<i>group</i>	Configuration Group
<i>tlv</i>	return data
<i>tlvLen</i>	the length of tlv data buffer

Returns

RETURN_CODE: Values can be parsed with [device_getIDGStatusCodeString\(\)](#)

11.14.4.11 ctls_registerCallBk()

```
void ctls_registerCallBk (
    pMSR_callback pCTLSf )
```

To register the ctls callback function to get the MSR card data. (Pass NULL to disable the callback.)

11.14.4.12 ctls_registerCallBkp()

```
void ctls_registerCallBkp (
    pMSR_callback pCTLSf )
```

To register the ctls callback function to get the MSR card data pointer. (Pass NULL to disable the callback.)

11.14.4.13 ctls_removeAllApplicationData()

```
int ctls_removeAllApplicationData ( )
```

Remove All Application Data

Removes all the Application Data

Returns

RETURN_CODE: Values can be parsed with [device_getIDGStatusCodeString\(\)](#)

11.14.4.14 ctls_removeAllCAPK()

```
int ctls_removeAllCAPK ( )
```

Remove All Certificate Authority Public Key

Removes all the CAPK

Returns

RETURN_CODE: Values can be parsed with [device_getIDGStatusCodeString\(\)](#)

11.14.4.15 ctls_removeApplicationData()

```
int ctls_removeApplicationData (
    IN BYTE * AID,
    IN int AIDLLen )
```

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>AIDLLen</i>	the length of AID data buffer

Returns

RETURN_CODE: Values can be parsed with [device_getIDGStatusCodeString\(\)](#)

11.14.4.16 ctls_removeCAPK()

```
int ctls_removeCAPK (
    IN BYTE * capk,
    IN int capkLen )
```

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>capkLen</i>	the length of capk data buffer

Returns

RETURN_CODE: Values can be parsed with [device_getIDGStatusCodeString\(\)](#)

11.14.4.17 ctls_removeConfigurationGroup()

```
int ctls_removeConfigurationGroup (
    int group )
```

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>	Values can be parsed with device_getIDGStatusCodeString()
--------------------	---

11.14.4.18 ctls_retrieveAIDList()

```
int ctls_retrieveAIDList (
    OUT BYTE * AIDList,
    IN_OUT int * AIDListLen )
```

Retrieve AID list

Returns all the AID names installed on the terminal for CTLS. .

Parameters

<i>AIDList</i>	array of AID name byte arrays
<i>AIDListLen</i>	the length of AIDList array buffer

Returns

RETURN_CODE: Values can be parsed with [device_getIDGStatusCodeString\(\)](#)

11.14.4.19 ctls_retrieveApplicationData()

```
int ctls_retrieveApplicationData (
    IN BYTE * AID,
    IN int AIDLen,
    OUT BYTE * tlv,
    IN_OUT int * tlvLen )
```

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>AIDLen</i>	the length of AID data buffer.
<i>tlv</i>	The TLV elements of the requested AID
<i>tlvLen</i>	the length of tlv data buffer.

Returns

RETURN_CODE: Values can be parsed with [device_getIDGStatusCodeString\(\)](#)

11.14.4.20 ctls_retrieveCAPK()

```
int ctls_retrieveCAPK (
    IN BYTE * capk,
    IN int capkLen,
    OUT BYTE * key,
    IN_OUT int * keyLen )
```

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>capkLen</i>	the length of capk data buffer

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>keyLen</i>	<p>the length of key data buffer</p> <ul style="list-style-type: none"> •

Returns

RETURN_CODE: Values can be parsed with [device_getIDGStatusCodeString\(\)](#)

11.14.4.21 `ctls_retrieveCAPKList()`

```
int ctls_retrieveCAPKList (
    OUT BYTE * keys,
    IN_OUT int * keyLen )
```

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>keyLen</i>	the length of keys data buffer

Returns

RETURN_CODE: Values can be parsed with [device_getIDGStatusCodeString\(\)](#)

11.14.4.22 `ctls_retrieveTerminalData()`

```
int ctls_retrieveTerminalData (
    OUT BYTE * tlv,
    IN_OUT int * tlvLen )
```

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>tlvLen</i>	the length of tlv data buffer

Returns

RETURN_CODE: Values can be parsed with [device_getIDGStatusCodeString\(\)](#)

11.14.4.23 `ctls_setApplicationData()`

```
int ctls_setApplicationData (  
    IN BYTE * tlv,  
    IN int tlvLen )
```

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>tlvLen</i>	the length of tlv data buffer
---------------	-------------------------------

Returns

RETURN_CODE: Values can be parsed with [device_getIDGStatusCodeString\(\)](#)

11.14.4.24 `ctls_setCAPK()`

```
int ctls_setCAPK (  
    IN BYTE * capk,  
    IN int capkLen )
```

Set Certificate Authority Public Key

Sets the CAPK as specified by the CAKey structure

Parameters

<i>capk</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>capkLen</i>	the length of capk data buffer

Returns

RETURN_CODE: Values can be parsed with [device_getIDGStatusCodeString\(\)](#)

11.14.4.25 `ctls_setConfigurationGroup()`

```
int ctls_setConfigurationGroup (
    IN BYTE * tlv,
    IN int tlvLen )
```

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>tlvLen</i>	the length of tlv data buffer

Returns

RETURN_CODE: Values can be parsed with [device_getIDGStatusCodeString\(\)](#)

11.14.4.26 `ctls_setTerminalData()`

```
int ctls_setTerminalData (
    IN BYTE * tlv,
    IN int tlvLen )
```

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 `ctls_getConfigurationGroup(int group)`, and deleted with `ctls_removeConfigurationGroup(int group)`. You cannot delete group 0.

Parameters

<i>tlv</i>	TerminalData configuration file
<i>tlvLen</i>	the length of tlv data buffer

Return values

RETURN_CODE	Values can be parsed with <code>device_getIDGStatusCodeString()</code>
--------------------	--

11.14.4.27 ctls_startTransaction()

```
int ctls_startTransaction (
    IN double amount,
    IN double amtOther,
    IN int type,
    IN const int _timeout,
    IN BYTE * tags,
    IN int tagsLen )
```

Start CTLS Transaction Request

Authorizes the CTLS transaction for an ICC card

The tags will be returned in the callback routine.

Parameters

<i>amount</i>	Transaction amount value (tag value 9F02) - SEE IMPORTANT NOTE BELOW
<i>amtOther</i>	Other amount value, if any (tag value 9F03) - SEE IMPORTANT NOTE BELOW
<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 stream. Example, tag 9F0C with amount 0x000000000100 would be 0x9F0C06000000000100 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>tagsLen</i>	The length of tags data buffer.

>>>>>IMPORTANT: parameters for amount and amtOther MUST BE PASSED AS A DOUBLE VALUE WITH DECIMAL POINT. Example, do not pass 1, but instead pass 1.0 or 1.00. Otherwise, results will be unpredictable

Returns

RETURN_CODE: Values can be parsed with `device_getIDGStatusCodeString()` Note: if auto poll is on, it will return the error IDG_P2_STATUS_CODE_COMMAND_NOT_ALLOWED

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

11.14.4.28 device_activateTransaction()

```
int device_activateTransaction (
    IN const int _timeout,
    IN BYTE * tags,
    IN int tagsLen )
```

Start Transaction Request

Authorizes the transaction for an MSR/CTLS/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. Example, tag 9F0C with amount 0x000000000100 would be 0x9F0C06000000000100 Be sure to include 9F02 (amount) and 9C (transaction type).
<i>tagsLen</i>	The length of tags data buffer.

>>>>>IMPORTANT: parameters for amount and amtOther MUST BE PASSED AS A DOUBLE VALUE WITH DECIMAL POINT. Example, do not pass 1, but instead pass 1.0 or 1.00. Otherwise, results will be unpredictable

Returns

RETURN_CODE: Values can be parsed with [device_getIDGStatusCodeString\(\)](#) Note: if auto poll is on, it will return the error IDG_P2_STATUS_CODE_COMMAND_NOT_ALLOWED

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 `device_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

11.14.4.29 device_cancelTransaction()

```
int device_cancelTransaction ( )
```

Disable Transaction Cancel Transaction request.

Returns

RETURN_CODE: Values can be parsed with [device_getIDGStatusCodeString\(\)](#)

11.14.4.30 device_close()

```
int device_close ( )
```

Close the device

Returns

RETURN_CODE: 0: success, 0x0A: failed

11.14.4.31 device_controlUserInterface()

```
int device_controlUserInterface (
    IN BYTE * values )
```

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) • 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
---------------	--

Returns

RETURN_CODE: Values can be parsed with [device_getIDGStatusCodeString\(\)](#)

11.14.4.32 device_enablePassThrough()

```
int device_enablePassThrough (
    int enablePassThrough )
```

Enable Pass Through

Enables Pass Through Mode for direct communication with L1 interface (power on icc, send apdu, etc).

Parameters

<i>enablePassThrough</i>	1 = pass through ON, 0 = pass through OFF
--------------------------	---

Returns

RETURN_CODE: Values can be parsed with [device_getIDGStatusCodeString\(\)](#)

11.14.4.33 device_getCurrentDeviceType()

```
int device_getCurrentDeviceType ( )
```

Get current active device type

Returns

: return the device type defined as DEVICE_TYPE in the IDTDef.h

11.14.4.34 device_getFirmwareVersion()

```
int device_getFirmwareVersion (
    OUT char * firmwareVersion )
```

DEPRECATED : please use [device_getFirmwareVersion_Len\(\)](#)(OUT char* firmwareVersion, IN_OUT int *firmwareVersionLen)

Polls device for Firmware Version

Parameters

<i>firmwareVersion</i>	Response returned of Firmware Version; needs to have at least 128 bytes of memory
------------------------	---

Returns

RETURN_CODE: Values can be parsed with [device_getIDGStatusCodeString\(\)](#)

11.14.4.35 device_getFirmwareVersion_Len()

```
int device_getFirmwareVersion_Len (
    OUT char * firmwareVersion,
    IN_OUT int * firmwareVersionLen )
```

Polls device for Firmware Version

Parameters

<i>firmwareVersion</i>	Response returned of Firmware Version
<i>firmwareVersionLen</i>	Length of Firmware Version

Returns

RETURN_CODE: Values can be parsed with [device_getIDGStatusCodeString\(\)](#)

11.14.4.36 device_getIDGStatusCodeString()

```
void device_getIDGStatusCodeString (
    IN int  returnCode,
    OUT char * despcrition )
```

Review the return code description.

Parameters

<i>returnCode</i>	the response result.
<i>description</i>	

Return values

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

- 0: "no error, beginning task";
- 1: "no response from reader";
- 2: "invalid response data";
- 01: " Incorrect Header Tag";
- 02: " Unknown Command";
- 03: " Unknown Sub-Command";
- 04: " CRC Error in Frame";
- 05: " Incorrect Parameter";
- 06: " Parameter Not Supported";
- 07: " Mal-formatted Data";
- 08: " Timeout";
- 0A: " Failed / NACK";
- 0B: " Command not Allowed";
- 0C: " Sub-Command not Allowed";
- 0D: " Buffer Overflow (Data Length too large for reader buffer)";

- 0E: " User Interface Event";
- 10: " Need clear firmware(apply in boot loader only)";
- 11: " Communication type not supported, VT-1, burst, etc. Need encrypted firmware (apply in boot loader only)";
- 12: " Secure interface is not functional or is in an intermediate state.";
- 13: " Data field is not mod 8";
- 14: " Pad 0x80 not found where expected";
- 15: " Specified key type is invalid";
- 16: " Could not retrieve key from the SAM (InitSecureComm)";
- 17: " Hash code problem";
- 18: " Could not store the key into the SAM (InstallKey)";
- 19: " Frame is too large";
- 1A: " Unit powered up in authentication state but POS must resend the InitSecureComm command";
- 1B: " The EEPROM may not be initialized because SecCommInterface does not make sense";
- 1C: " Problem encoding APDU Module-Specific Status Codes ";
- 20: " Unsupported Index (ILM) SAM Transceiver error - problem communicating with the SAM (Key Mgr)";
- 21: " Unexpected Sequence Counter in multiple frames for single bitmap (ILM)Length error in data returned from the SAM (Key Mgr) - 22: " Improper bit map (ILM)"; - 23: " Request Online Authorization"; - 24: " ViV↔ OCard3 raw data read successful"; - 25: " Message index not available (ILM) ViVOcomm activate transaction card type (ViVOcomm)"; - 26: " Version Information Mismatch (ILM)"; - 27: " Not sending commands in correct index message index (ILM)"; - 28: " Time out or next expected message not received (ILM)"; - 29: " ILM languages not available for viewing (ILM)"; - 2A: " Other language not supported (ILM)"; - 41: " from 41 to 4F, Module-specific errors for Key Manager";
- 50: " Auto-Switch OK";
- 51: " Auto-Switch failed";
- 70: " Antenna Error 80h Use another card";
- 81: " Insert or swipe card";
- 90: " Data encryption Key does not exist";
- 91: " Data encryption Key KSN exhausted";

11.14.4.37 device_getMerchantRecord()

```
int device_getMerchantRecord (
    IN int index,
    OUT BYTE * record )
```

DEPRECATED : please use device_getMerchantRecord_Len(IN int index, OUT BYTE * record, IN_OUT int *recordLen)

Get Merchant Record

Gets the merchant record for the device.

Parameters

<i>index</i>	Merchant Record index, valid values 1-6
<i>record;</i>	needs to have at least 99 bytes of memory response data from reader. Merchant Record Index: 1 byte enabled: 1 byte Merchant ID: 32 bytes Length of Merchant URL: 1 byte Merchant URL: 64 bytes

Returns

success or error code. Values can be parsed with [device_getIDGStatusCodeString\(\)](#)

See also

[ErrorCode](#)

11.14.4.38 [device_getMerchantRecord_Len\(\)](#)

```
int device_getMerchantRecord_Len (
    IN int index,
    OUT BYTE * record,
    IN_OUT int * recordLen )
```

Get Merchant Record

Gets the merchant record for the device.

Parameters

<i>index</i>	Merchant Record index, valid values 1-6
<i>record</i>	response data from reader. Merchant Record Index: 1 byte enabled: 1 byte Merchant ID: 32 bytes Length of Merchant URL: 1 byte Merchant URL: 64 bytes
<i>recordLen</i>	Length of record

Returns

success or error code. Values can be parsed with [device_getIDGStatusCodeString\(\)](#)

See also

[ErrorCode](#)

11.14.4.39 [device_getRTCDateTime\(\)](#)

```
int device_getRTCDateTime (
    IN BYTE * dateTime,
    IN_OUT int * dateTimeLen )
```

get RTC date and time of the device

Parameters

<i>dateTime</i>	<dateTime data>="" is: 6 byte data for YYMMDDHHMMSS in hex. For example 0x171003102547 stands for 2017 Oct 3rd 10:25:47
<i>dateTimeLen</i>	return 6 bytes if successful

Returns

success or error code. Values can be parsed with device_getResponseCodeString

See also

ErrorCode

11.14.4.40 device_getSDKWaitTime()

```
int device_getSDKWaitTime ( )
```

Get SDK Wait Time

Get the SDK wait time for transactions

Returns

SDK wait time in seconds

11.14.4.41 device_getThreadStackSize()

```
int device_getThreadStackSize ( )
```

Get Thread Stack Size

Get the stack size setting for newly created threads

Returns

Thread Stack Size

11.14.4.42 device_getTransactionResults()

```
int device_getTransactionResults (
    IDTMSRData * cardData )
```

Get Transaction Results Gets the transaction results when the reader is functioning in "Auto Poll" mode

Parameters

<i>cardData</i>	The transaction results
-----------------	-------------------------

Returns

success or error code. Values can be parsed with device_getResponseCodeString

See also

ErrorCode

11.14.4.43 device_init()

```
int device_init ( )
```

Initial the device by USB

It will detect the device and trying connect.

The connect status can be checked by [device_isConnected\(\)](#).

Note: after the function returns success, the function [device_setCurrentDevice\(\)](#) has to be called to set the device type.

Returns

RETURN_CODE: Values can be parsed with [device_getIDGStatusCodeString\(\)](#)

11.14.4.44 device_isAttached()

```
int device_isAttached (
    int deviceType )
```

Check if the device is attached to the USB port The function [device_init\(\)](#) must be called before this function.

Parameters

<i>deviceType,the</i>	device type of the USB device
-----------------------	-------------------------------

Returns

1 if the device is attached, or 0 if the device is not attached

11.14.4.45 device_isConnected()

```
int device_isConnected ( )
```

Check the device connected status

Returns

DEVICE_DISCONNECT=0, or DEVICE_CONNECTED = 1

11.14.4.46 device_pingDevice()

```
int device_pingDevice ( )
```

Ping Device

Pings the reader. If connected, returns success. Otherwise, returns timeout.

Returns

RETURN_CODE: Values can be parsed with [device_getIDGStatusCodeString\(\)](#)

11.14.4.47 device_pollForToken()

```
int device_pollForToken (
    IN int timeout,
    OUT BYTE * respData,
    IN_OUT int * respDataLen )
```

Poll for Token

Polls for a PICC

Parameters

<i>timeout</i>	timeout in milliseconds, must be multiple of 10 milliseconds. 30, 120, 630, or 1150 for example.
<i>respData</i>	Response data will be stored in respData. 1 byte of card type, and the Serial Number (or the UID) of the PICC if available.
<i>respDataLen</i>	Length of systemCode.

Returns

RETURN_CODE: Values can be parsed with [device_getIDGStatusCodeString\(\)](#)

11.14.4.48 device_registerCameraCallBk()

```
void device_registerCameraCallBk (
    pCMR_callback pCMRf )
```

To register the camera callback function to get the image data. (Pass NULL to disable the callback.)

11.14.4.49 device_registerCardStatusFrontSwitchCallBk()

```
void device_registerCardStatusFrontSwitchCallBk (
    pCSFS_callback pCSFSf )
```

To register the card status and front switch callback function to get status. (Pass NULL to disable the callback.)

11.14.4.50 device_registerRKICallBk()

```
void device_registerRKICallBk (
    pRKI_callback pRKIf )
```

To register the RKI callback function to get the RKI status. (Pass NULL to disable the callback.)

11.14.4.51 device_SendDataCommandNEO()

```
int device_SendDataCommandNEO (
    IN int cmd,
    IN int subCmd,
    IN BYTE * data,
    IN int dataLen,
    OUT BYTE * response,
    IN_OUT int * respLen )
```

Send a Command to device

Sends a command to the device .

Parameters

<i>cmd</i>	buffer of command to execute.
<i>cmdLen,the</i>	length of the buffer cmd.
<i>data</i>	buffer of IDG command data.
<i>dataLen,the</i>	length of the buffer data.
<i>response</i>	Response data
<i>respLen,the</i>	length of Response data

Returns

RETURN_CODE: Values can be parsed with [device_getIDGStatusCodeString\(\)](#) Send a Command to NEO device

Sends a command to the NEO device .

Parameters

<i>cmd</i>	command to execute.
<i>subCmd,sub</i>	command to execute.
<i>data</i>	buffer of NEO command data.
<i>dataLen,the</i>	length of the buffer data.
<i>response</i>	Response data
<i>respLen,the</i>	length of Response data

Returns

RETURN_CODE: Values can be parsed with [device_getIDGStatusCodeString\(\)](#)

11.14.4.52 device_setBurstMode()

```
int device_setBurstMode (
    IN BYTE mode )
```

Send Burst Mode

Sets the burst mode for the device.

Parameters

<i>mode</i>	0 = OFF, 1 = Always On, 2 = Auto Exit
-------------	---------------------------------------

Returns

success or error code. Values can be parsed with [device_getIDGStatusCodeString](#)

See also

ErrorCode

11.14.4.53 device_setCurrentDevice()

```
int device_setCurrentDevice (
    int deviceType )
```

Sets the current device to talk to

The connect status can be checked by [device_isConnected\(\)](#).

Parameters

<i>deviceType</i>	<p>Device to connect to</p> <pre>enum DEVICE_TYPE { IDT_DEVICE_UNKNOWN=0, IDT_DEVICE_AUGUSTA_HID, IDT_DEVICE_AUGUSTA_KB, IDT_DEVICE_AUGUSTA_S_HID, IDT_DEVICE_AUGUSTA_S_KB, IDT_DEVICE_AUGUSTA_S_TTK_HID, IDT_DEVICE_SPECTRUM_PRO, IDT_DEVICE_MINISMART_II, IDT_DEVICE_L80, IDT_DEVICE_L100, IDT_DEVICE_UNIPAY, IDT_DEVICE_UNIPAY_I_V, IDT_DEVICE_VP3300_AJ, IDT_DEVICE_KIOSK_III, IDT_DEVICE_KIOSK_III_S, IDT_DEVICE_PIP_READER, IDT_DEVICE_VENDI, IDT_DEVICE_VP3300_USB, IDT_DEVICE_UNIPAY_I_V_TTK, IDT_DEVICE_VP3300_BT, IDT_DEVICE_VP8800, IDT_DEVICE_SREDKEY2_HID, IDT_DEVICE_SREDKEY2_KB, IDT_DEVICE_NEO2, IDT_DEVICE_MINISMART_II_COM = IDT_DEVICE_NEO2+5, IDT_DEVICE_SPECTRUM_PRO_COM, IDT_DEVICE_KIOSK_III_COM, IDT_DEVICE_KIOSK_III_S_COM, IDT_DEVICE_PIP_READER_COM, IDT_DEVICE_VP3300_COM, IDT_DEVICE_NEO2_COM, IDT_DEVICE_MAX_DEVICES = IDT_DEVICE_NEO2_COM+5 };</pre>
-------------------	---

Returns

RETURN_CODE: 1: success, 0: failed

11.14.4.54 device_setMerchantRecord()

```
int device_setMerchantRecord (
    int index,
    int enabled,
    char * merchantID,
    char * merchantURL )
```

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

Returns

RETURN_CODE: Values can be parsed with [device_getIDGStatusCodeString\(\)](#)

11.14.4.55 device_setPollMode()

```
int device_setPollMode (
    IN BYTE mode )
```

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>	0 = Auto Poll, 1 = Poll On Demand
-------------	-----------------------------------

Returns

RETURN_CODE: Values can be parsed with [device_getIDGStatusCodeString\(\)](#)

11.14.4.56 device_setRKI_URL()

```
void device_setRKI_URL (
    IN char * rkiURL,
    IN int rkiURLLen )
```

Set the URL for RKI

Parameters

<i>rkiURL</i>	The URL for RKI (less than 100 characters). Pass NULL to reset to default URL
<i>rkiURLLen</i>	The length of rkiURL. Pass 0 to reset to default URL

Returns

11.14.4.57 device_setRTCDateTime()

```
int device_setRTCDateTime (
    IN BYTE * dateTime,
    IN int dateTimeLen )
```

set RTC date and time of the device

Parameters

<i>dateTime</i>	<dateTime data>=""> is: 6 byte data for YYMMDDHHMMSS in hex. For example 0x171003102547 stands for 2017 Oct 3rd 10:25:47
<i>dateTimeLen</i>	should be always 6 bytes

Returns

success or error code. Values can be parsed with [device_getResponseCodeString](#)

See also

[ErrorCode](#)

11.14.4.58 device_setSDKWaitTime()

```
void device_setSDKWaitTime (
    int waitTime )
```

Set SDK Wait Time

Set the SDK wait time for transactions

Parameters

<i>waitTime</i>	The SDK wait time for transaction in seconds. The Maximum is 2147483 seconds.
-----------------	---

11.14.4.59 device_setThreadStackSize()

```
void device_setThreadStackSize (
    int threadSize )
```

Set Thread Stack Size

Set the stack size setting for newly created threads

11.14.4.60 device_setTransactionExponent()

```
void device_setTransactionExponent (
    int exponent )
```

Sets the transaction exponent to be used with device_startTransaction. Default value is 2

Parameters

<i>exponent, The</i>	exponent to use when calling device_startTransaction
----------------------	--

11.14.4.61 device_startRKI()

```
int device_startRKI (
    IN const char * caPath,
    IN int isProduction )
```

Start remote key injection.

Parameters

<i>caPath</i>	The path to ca-certificates.crt It should be NULL, because the file is not used anymore.
<i>isProduction</i>	1: The reader is a production unit, 0: The reader is not a production unit

Returns

success or error code.

See also

[ErrorCode](#)

11.14.4.62 device_startTransaction()

```
int device_startTransaction (
    IN double amount,
    IN double amtOther,
    IN int type,
    IN const int _timeout,
    IN BYTE * tags,
    IN int tagsLen )
```

Start Transaction Request

Authorizes the transaction for an MSR/CTLS/ICC card

The tags will be returned in the callback routine.

Parameters

<i>amount</i>	Transaction amount value (tag value 9F02) - SEE IMPORTANT NOTE BELOW
<i>amtOther</i>	Other amount value, if any (tag value 9F03) - SEE IMPORTANT NOTE BELOW
<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. Example, tag 9F0C with amount 0x000000000100 would be 0x9F0C06000000000100 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>tagsLen</i>	The length of tags data buffer.

>>>>IMPORTANT: parameters for amount and amtOther MUST BE PASSED AS A DOUBLE VALUE WITH DECIMAL POINT. Example, do not pass 1, but instead pass 1.0 or 1.00. Otherwise, results will be unpredictable

Returns

RETURN_CODE: Values can be parsed with [device_getIDGStatusCodeString\(\)](#) Note: if auto poll is on, it will return the error IDG_P2_STATUS_CODE_COMMAND_NOT_ALLOWED

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 device_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

11.14.4.63 emv_activateTransaction()

```
int emv_activateTransaction (
    IN int timeout,
    IN BYTE * tags,
    IN int tagsLen,
    IN int forceOnline )
```

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>	Tags to be included in the request. Passed as a TLV stream. Example, tag 9F0C with amount 0x000000000100 would be 0x9F0C06000000000100
<i>tagsLen</i>	Length of tags
<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 = DFEE1A079F029F36959F37

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString >>>>IMPORTANT: parameters for amount and amtOther MUST BE PASSED AS A DOUBLE VALUE WITH DECIMAL POINT. Example, do not pass 1, but instead pass 1.0 or 1.00. Otherwise, results will be unpredictable

11.14.4.64 emv_allowFallback()

```
void emv_allowFallback (
    IN int allow )
```

Allow fallback for EMV transactions. Default is TRUE

Parameters

<i>allow</i>	TRUE = allow fallback, FALSE = don't allow fallback
--------------	---

11.14.4.65 emv_authenticateTransaction()

```
int emv_authenticateTransaction (
    IN BYTE * updatedTLV,
    IN int updatedTLVLen )
```

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 = DFEE1A079F029F36959F37
<i>updatedTLVLen</i>	

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString

11.14.4.66 emv_authenticateTransactionWithTimeout()

```
int emv_authenticateTransactionWithTimeout (
```



```

    IN int timeout,
    IN BYTE * updatedTLV,
    IN int updatedTLVLen )

```

Authenticate EMV Transaction Request with Timeout

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>timeout</i>	Timeout value in seconds.
<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 = DFEE1A079F029F36959F37
<i>updatedTLVLen</i>	

Returns

RETURN_CODE: Values can be parsed with `device_getResponseCodeString`

11.14.4.67 `emv_cancelTransaction()`

```
int emv_cancelTransaction ( )
```

Cancel EMV Transaction

Cancels the currently executing EMV transaction.

Returns

RETURN_CODE: Values can be parsed with [device_getResponseCodeString\(\)](#)

11.14.4.68 `emv_completeTransaction()`

```

int emv_completeTransaction (
    IN int commError,
    IN BYTE * authCode,
    IN int authCodeLen,
    IN BYTE * iad,
    IN int iadLen,
    IN BYTE * tlvScripts,
    IN int tlvScriptsLen,
    IN BYTE * tlv,
    IN int tlvLen )

```

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(1) if host was unreachable, or FALSE(0) 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>authCodeLen</i>	the length of <i>authCode</i>
<i>iad</i>	Issuer Authentication Data, if any. Example 0x11223344556677883030 (tag value 91).
<i>iadLen</i>	the length of <i>iadLen</i>
<i>tlvScripts</i>	71/72 scripts, if any
<i>tlvScriptsLen</i>	the length of <i>tlvScriptsLen</i>
<i>tlv</i>	Additional TLV data to return with transaction results (if any)
<i>tlvLen</i>	the length of <i>tlv</i>

Returns

RETURN_CODE: Values can be parsed with `device_getResponseCodeString`

11.14.4.69 `emv_getAutoAuthenticateTransaction()`

```
int emv_getAutoAuthenticateTransaction ( )
```

Gets auto authenticate value for EMV transactions.

Returns

RETURN_CODE: TRUE = auto authenticate, FALSE = manually authenticate

11.14.4.70 `emv_getAutoCompleteTransaction()`

```
int emv_getAutoCompleteTransaction ( )
```

Gets auto complete value for EMV transactions.

Returns

RETURN_CODE: TRUE = auto complete, FALSE = manually complete

11.14.4.71 `emv_registerCallBk()`

```
void emv_registerCallBk (
    pEMV_callback pEMVf )
```

To register the emv callback function to get the EMV processing response. (Pass NULL to disable the callback.)

11.14.4.72 emv_removeAllApplicationData()

```
int emv_removeAllApplicationData ( )
```

Remove All Application Data

Removes all the Application Data

Returns

RETURN_CODE: Values can be parsed with [device_getIDGStatusCodeString\(\)](#)

11.14.4.73 emv_removeAllCAPK()

```
int emv_removeAllCAPK ( )
```

Remove All Certificate Authority Public Key

Removes all the CAPK

Returns

RETURN_CODE: Values can be parsed with [device_getIDGStatusCodeString\(\)](#)

11.14.4.74 emv_removeAllCRL()

```
int emv_removeAllCRL ( )
```

Remove All Certificate Revocation List Entries

Removes all CRLentry entries

Returns

RETURN_CODE: Values can be parsed with [device_getResponseCodeString\(\)](#)

11.14.4.75 emv_removeApplicationData()

```
int emv_removeApplicationData (
    IN BYTE * AID,
    IN int AIDLen )
```

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>AIDLen</i>	the length of AID data buffer

Returns

RETURN_CODE: Values can be parsed with [device_getIDGStatusCodeString\(\)](#)

11.14.4.76 emv_removeCAPK()

```
int emv_removeCAPK (
    IN BYTE * capk,
    IN int capkLen )
```

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>capkLen</i>	the length of capk data buffer

Returns

RETURN_CODE: Values can be parsed with [device_getIDGStatusCodeString\(\)](#)

11.14.4.77 emv_removeCRL()

```
int emv_removeCRL (
    IN BYTE * list,
    IN int lsLen )
```

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>lsLen</i>	the length of list data buffer

Returns

RETURN_CODE: Values can be parsed with [device_getResponseCodeString\(\)](#)

11.14.4.78 emv_retrieveAIDList()

```
int emv_retrieveAIDList (
    OUT BYTE * AIDList,
    IN_OUT int * AIDListLen )
```

Retrieve AID list

Returns all the AID names installed on the terminal for CTLS. .

Parameters

<i>AIDList</i>	array of AID name byte arrays
<i>AIDListLen</i>	the length of AIDList array buffer

Returns

RETURN_CODE: Values can be parsed with [device_getIDGStatusCodeString\(\)](#)

11.14.4.79 emv_retrieveApplicationData()

```
int emv_retrieveApplicationData (
    IN BYTE * AID,
    IN int AIDLen,
    OUT BYTE * tlv,
    IN_OUT int * tlvLen )
```

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>AIDLen</i>	the length of AID data buffer.
<i>tlv</i>	The TLV elements of the requested AID
<i>tlvLen</i>	the length of tlv data buffer.

Returns

RETURN_CODE: Values can be parsed with [device_getIDGStatusCodeString\(\)](#)

11.14.4.80 emv_retrieveCAPK()

```
int emv_retrieveCAPK (
    IN BYTE * capk,
    IN int capkLen,
    OUT BYTE * key,
    IN_OUT int * keyLen )
```

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>capkLen</i>	the length of capk data buffer

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>keyLen</i>	<p>the length of key data buffer</p> <ul style="list-style-type: none"> •

Returns

RETURN_CODE: Values can be parsed with [device_getIDGStatusCodeString\(\)](#)

11.14.4.81 emv_retrieveCAPKList()

```
int emv_retrieveCAPKList (
    OUT BYTE * keys,
    IN_OUT int * keysLen )
```

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>keysLen</i>	the length of keys data buffer

Returns

RETURN_CODE: Values can be parsed with [device_getIDGStatusCodeString\(\)](#)

11.14.4.82 emv_retrieveCRL()

```
int emv_retrieveCRL (
    OUT BYTE * list,
    IN_OUT int * lssLen )
```

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>lssLen</i>	the length of list data buffer

Returns

RETURN_CODE: Values can be parsed with [device_getResponseCodeString\(\)](#)

11.14.4.83 emv_retrieveTerminalData()

```
int emv_retrieveTerminalData (
    OUT BYTE * tlv,
    IN_OUT int * tlvLen )
```

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>tlvLen</i>	the length of tlv data buffer

Returns

RETURN_CODE: Values can be parsed with [device_getIDGStatusCodeString\(\)](#)

11.14.4.84 emv_setApplicationData()

```
int emv_setApplicationData (
    IN BYTE * name,
    IN int nameLen,
    IN BYTE * tlv,
    IN int tlvLen )
```

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>nameLen</i>	the length of name data buffer of Application name,
<i>tlv</i>	Application data in TLV format
<i>tlvLen</i>	the length of tlv data buffer

Returns

RETURN_CODE: Values can be parsed with [device_getResponseCodeString\(\)](#)

11.14.4.85 emv_setApplicationDataTLV()

```
int emv_setApplicationDataTLV (
    IN BYTE * tlv,
    IN int tlvLen )
```

Set Application Data by TLV

Sets the Application Data as specified by the TLV data

Parameters

<i>tlv</i>	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) Example valid TLV, for Group #2, AID a0000000035010: "dfee2d01029f0607a0000000051010ffe10101ffe50110ffe30114ffe20106"
<i>tlvLen</i>	the length of tlv data buffer

Returns

RETURN_CODE: Values can be parsed with [device_getResponseCodeString\(\)](#)

11.14.4.86 emv_setAutoAuthenticateTransaction()

```
void emv_setAutoAuthenticateTransaction (
    IN int authenticate )
```

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

Parameters

<i>authenticate</i>	TRUE = auto authenticate, FALSE = manually authenticate
---------------------	---

11.14.4.87 emv_setAutoCompleteTransaction()

```
void emv_setAutoCompleteTransaction (
    IN int complete )
```

Enables complete for EMV transactions. If a `emv_authenticateTransaction` results in code 0x0004 (go online), then `emv_completeTransaction` can automatically execute if parameter is set to TRUE

Parameters

<i>complete</i>	TRUE = auto complete, FALSE = manually complete
-----------------	---

11.14.4.88 emv_setCAPK()

```
int emv_setCAPK (
    IN BYTE * capk,
    IN int capkLen )
```

Set Certificate Authority Public Key

Sets the CAPK as specified by the CAKey structure

Parameters

<i>capk</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>capkLen</i>	the length of capk data buffer

Returns

RETURN_CODE: Values can be parsed with [device_getIDGStatusCodeString\(\)](#)

11.14.4.89 emv_setCRL()

```
int emv_setCRL (
    IN BYTE * list,
    IN int lsLen )
```

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>lsLen</i>	the length of list data buffer

Returns

RETURN_CODE: Values can be parsed with [device_getResponseCodeString\(\)](#)

11.14.4.90 emv_setTerminalData()

```
int emv_setTerminalData (
    IN BYTE * tlv,
    IN int tlvLen )
```

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 file
<i>tlvLen</i>	the length of tlv data buffer

Return values

RETURN_CODE	Values can be parsed with device_getIDGStatusCodeString()
--------------------	---

11.14.4.91 emv_setTerminalMajorConfiguration()

```
int emv_setTerminalMajorConfiguration (
    IN int configuration )
```

Sets the terminal major configuration in ICS .

Parameters

<i>configuration</i>	A configuration value, range 1-23 <ul style="list-style-type: none"> • 1 = 1C • 2 = 2C • 3 = 3C • 4 = 4C • 5 = 5C ... • 23 = 23C
----------------------	--

Returns

RETURN_CODE: Values can be parsed with [device_getResponseCodeString\(\)](#)

11.14.4.92 emv_setTransactionParameters()

```
void emv_setTransactionParameters (
    IN double amount,
```

```

    IN double amtOther,
    IN int type,
    IN int timeout,
    IN BYTE * tags,
    IN int tagsLen )

```

Set EMV Transaction Parameters

Set the parameters to be used on EMV transactions for an ICC card when Auto Poll is on

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 (Maximum Length = 500 bytes). Passed as a string. Example, tag 9F0C with amount 0x000000000100 would be "9F0C06000000000100" 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 = DFEE1A079F029F36959F37
<i>tagsLen</i>	the length of tags

11.14.4.93 emv_startTransaction()

```

int emv_startTransaction (
    IN double amount,
    IN double amtOther,
    IN int exponent,
    IN int type,
    IN int timeout,
    IN BYTE * tags,
    IN int tagsLen,
    IN int forceOnline )

```

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) - SEE IMPORTANT NOTE BELOW
<i>amtOther</i>	Other amount value, if any (tag value 9F03) - SEE IMPORTANT NOTE BELOW
<i>exponent</i>	Number of characters after decimal 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 stream. Example, tag 9F0C with amount 0x000000000100 would be 0x9F0C06000000000100 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>tagsLen</i>	Length of tags

Parameters

<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 = DFEE1A079F029F36959F37
--------------------	--

Returns

RETURN_CODE: Values can be parsed with `device_getResponseCodeString` >>>>IMPORTANT: parameters for amount and amtOther MUST BE PASSED AS A DOUBLE VALUE WITH DECIMAL POINT. Example, do not pass 1, but instead pass 1.0 or 1.00. Otherwise, results will be unpredictable

11.14.4.94 executeTransaction()

```
int executeTransaction (
    WorldPayData * data,
    pWP_callback wpCallback,
    int requestOnly )
```

executeTransaction Requests 3 sets of public keys: encrypting Keys, signing/validating keys, signing/validating 3rd party apps

Parameters

<i>data</i>	WorldPay data object
<i>wpCallback</i>	WorldPay callback
<i>requestOnly</i>	<ul style="list-style-type: none"> • 0 = send transaction and return response, • 1 = send transaction and return both request and response, • 2 = do not send transaction, instead return request

Returns

RETURN_CODE: Values can be parsed with `device_getIDGStatusCodeString()`

11.14.4.95 executeTransaction_WorldNet()

```
int executeTransaction_WorldNet (
    WorldNetData * data,
    pWN_callback wnCallback,
    int requestOnly )
```

executeTransaction_WorldNet Requests 3 sets of public keys: encrypting Keys, signing/validating keys, signing/validating 3rd party apps

Parameters

<i>data</i>	WorldNet data object
-------------	----------------------

Parameters

<i>wnCallback</i>	WorldNet callback
<i>requestOnly</i>	<ul style="list-style-type: none"> • 0 = send transaction and return response, • 1 = send transaction and return both request and response, • 2 = do not send transaction, instead return request

Returns

RETURN_CODE: Values can be parsed with [device_getIDGStatusCodeString\(\)](#)

11.14.4.96 forwardTransaction()

```
int forwardTransaction (
    IN pWP_callback wpCallback,
    IN char * forwardID,
    IN int forwardIDLen,
    IN char * password,
    IN int passwordLen,
    IN int bypassProcessing )
```

forwardTransaction Send the saved data to WorldPay and complete the transaction.

Parameters

<i>wpCallback</i>	WPCallback is the callback to send the results. Should the the same as executeTransaction callback.
<i>forwardID</i>	= ID, which could be either unique ID or Memo.
<i>forwardIDLen</i>	The length of forwardID.
<i>password</i>	= password. If null/blank, no password.
<i>passwordLen</i>	The length of passwordLen.
<i>bypassProcess</i>	= true means read the file from disk, but don't send to WorldPay, then delete the transaction as if you did send to WorldPay. The purpose of this is to allow them to delete transactions from the storage without sending to WorldPay.

Returns

RETURN_CODE: Values can be parsed with [device_getIDGStatusCodeString\(\)](#)

11.14.4.97 forwardTransaction_WorldNet()

```
int forwardTransaction_WorldNet (
    IN char * apiKey,
    IN int apiKeyLen,
    IN pWN_callback wnCallback,
    IN char * forwardID,
    IN int forwardIDLen,
```

```

    IN char * password,
    IN int passwordLen,
    IN int bypassProcessing )

```

forwardTransaction_WorldNet Send the saved data to WorldNet and complete the transaction.

Parameters

<i>wnCallback</i>	WNCallback is the callback to send the results. Should be the same as executeTransaction_WorldNet callback.
<i>forwardID</i>	= ID, which could be either unique ID or Memo.
<i>forwardIDLen</i>	The length of forwardID.
<i>password</i>	= password. If null/blank, no password.
<i>passwordLen</i>	The length of passwordLen.
<i>bypassProcess</i>	= true means read the file from disk, but don't send to WorldPay, then delete the transaction as if you did send to WorldPay. The purpose of this is to allow them to delete transactions from the storage without sending to WorldPay.

Returns

RETURN_CODE: Values can be parsed with [device_getIDGStatusCodeString\(\)](#)

11.14.4.98 icc_exchangeAPDU()

```

int icc_exchangeAPDU (
    IN BYTE * c_APDU,
    IN int cLen,
    OUT BYTE * reData,
    IN_OUT int * reLen )

```

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>cLen</i>	APDU data packet length
<i>reData</i>	Unencrypted APDU response
<i>reLen</i>	Unencrypted APDU response data length

Returns

RETURN_CODE: Values can be parsed with [device_getIDGStatusCodeString](#)

11.14.4.99 icc_getICCRReaderStatus()

```

int icc_getICCRReaderStatus (
    OUT BYTE * status )

```

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
---------------	--

Returns

RETURN_CODE: Values can be parsed with `device_getIDGStatusCodeString`

11.14.4.100 `icc_powerOffICC()`

```
int icc_powerOffICC ( )
```

Power Off ICC

Powers down the ICC

Returns

RETURN_CODE: Values can be parsed with `device_getIDGStatusCodeString()`

If Success, empty If Failure, ASCII encoded data of error string

11.14.4.101 `icc_powerOnICC()`

```
int icc_powerOnICC (
    OUT BYTE * ATR,
    IN_OUT int * inLen )
```

Power On ICC

Power up the currently selected microprocessor card in the ICC reader

Parameters

<i>ATR,the</i>	ATR data response when succeeded power on ICC,
<i>inLen,the</i>	length of ATR data

Returns

RETURN_CODE: Values can be parsed with `device_getIDGStatusCodeString()`

11.14.4.102 `msr_cancelMSRSwipe()`

```
int msr_cancelMSRSwipe ( )
```

Disable MSR Swipe Cancels MSR swipe request.

Returns

RETURN_CODE: Values can be parsed with `device_getIDGStatusCodeString()`

11.14.4.103 msr_registerCallBk()

```
void msr_registerCallBk (
    pMSR_callback pMSRf )
```

To register the msr callback function to get the MSR card data. (Pass NULL to disable the callback.)

11.14.4.104 msr_registerCallBkp()

```
void msr_registerCallBkp (
    pMSR_callbackp pMSRf )
```

To register the msr callback function to get the MSR card data pointer. (Pass NULL to disable the callback.)

11.14.4.105 msr_startMSRSwipe()

```
int msr_startMSRSwipe (
    IN int _timeout )
```

Start 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
----------------	---------------------

Returns

RETURN_CODE: Values can be parsed with [device_getIDGStatusCodeString\(\)](#) Note: if auto poll mode is on, it will return command not allowed error

11.14.4.106 parseMSRData()

```
void parseMSRData (
    IN BYTE * resData,
    IN int resLen,
    IN_OUT IDTMSRData * cardData )
```

Parser the MSR data from the buffer into IDTMSTData structure

Parameters

<i>resData</i>	MSR card data buffer
<i>resLen</i>	the length of resData
<i>cardData</i>	the parser result with IDTMSTData structure

11.14.4.107 pin_registerCallBk()

```
void pin_registerCallBk (
    pPIN_callback pPINf )
```

To register the pin callback function to get the PINPad data. (Pass NULL to disable the callback.)

11.14.4.108 registerHotplugCallBk()

```
void registerHotplugCallBk (
    pMessageHotplug pMsgHotplug )
```

To register the USB HID hot-plug callback function which implemented in the application to monitor the hotplug message from the SDK.

11.14.4.109 registerLogCallBk()

```
void registerLogCallBk (
    pSendDataLog pFSend,
    pReadDataLog pFRead )
```

To register the log callback function which implemented in the application to monitor sending/reading data between application and reader.

11.14.4.110 SDK_Version()

```
char* SDK_Version ( )
```

To Get SDK version

Returns

return the SDK version string

11.14.4.111 setAbsoluteLibraryPath()

```
int setAbsoluteLibraryPath (
    const char * absoluteLibraryPath )
```

Set the path to use when searching for ID TECH's libraries. If this is not set, the libraries will be searched for with the system's default procedures.

Parameters

<i>absoluteLibraryPath</i>	The absolute path to ID TECH's libraries.
----------------------------	---

Returns

RETURN_CODE: Values can be parsed with [device_getIDGStatusCodeString\(\)](#)

11.15 Source_C/libIDT_VP3300_COM.h File Reference

VP3300 COM API.

```
#include "IDTDef.h"
```

Macros

- #define [IN](#)
- #define [OUT](#)

- `#define IN_OUT`

Typedefs

- `typedef void(* pMessageHotplug) (int, int)`
- `typedef void(* pSendDataLog) (BYTE *, int)`
- `typedef void(* pReadDataLog) (BYTE *, int)`
- `typedef void(* pEMV_callBack) (int, int, BYTE *, int, IDTTransactionData *, EMV_Callback *, int)`
- `typedef void(* pMSR_callBack) (int, IDTMSRData)`
- `typedef void(* pMSR_callBackp) (int, IDTMSRData *)`
- `typedef void(* pPIN_callBack) (int, IDTPINData *)`
- `typedef void(* pCMR_callBack) (int, IDTCMRData *)`
- `typedef void(* pCSFS_callBack) (BYTE status)`
- `typedef void(* pRKI_callBack) (int, char *)`
- `typedef void(* ftpComm_callBack) (int, int, int)`
- `typedef void(* httpComm_callBack) (BYTE *, int)`
- `typedef void(* v4Comm_callBack) (BYTE, BYTE, BYTE *, int)`
- `typedef void(* pWP_callBack) (char *, int, int)`
- `typedef void(* pWN_callBack) (char *, int, int)`

Functions

- `void registerHotplugCallBk (pMessageHotplug pMsgHotplug)`
- `void registerLogCallBk (pSendDataLog pFSend, pReadDataLog pFRead)`
- `void device_registerRKICallBk (pRKI_callBack pRKIf)`
- `void emv_registerCallBk (pEMV_callBack pEMVf)`
- `void msr_registerCallBk (pMSR_callBack pMSRf)`
- `void msr_registerCallBkp (pMSR_callBackp pMSRf)`
- `void ctls_registerCallBk (pMSR_callBack pCTLSf)`
- `void ctls_registerCallBkp (pMSR_callBackp pCTLSf)`
- `void pin_registerCallBk (pPIN_callBack pPINf)`
- `void device_registerCameraCallBk (pCMR_callBack pCMRf)`
- `void device_registerCardStatusFrontSwitchCallBk (pCSFS_callBack pCSFSf)`
- `void comm_registerHTTPCallback (httpComm_callBack cBack)`
- `void comm_registerV4Callback (v4Comm_callBack cBack)`
- `char * SDK_Version ()`
- `int setAbsoluteLibraryPath (const char *absoluteLibraryPath)`
- `int device_init ()`
- `int rs232_device_init (int deviceType, int port_number, int brate)`
- `int device_setCurrentDevice (int deviceType)`
- `int device_isAttached (int deviceType)`
- `int device_close ()`
- `void device_getIDGStatusCodeString (IN int returnCode, OUT char *despcrition)`
- `int device_isConnected ()`
- `int device_getFirmwareVersion (OUT char *firmwareVersion)`
- `int device_getFirmwareVersion_Len (OUT char *firmwareVersion, IN_OUT int *firmwareVersionLen)`
- `int device_pingDevice ()`
- `int device_controlUserInterface (IN BYTE *values)`
- `int device_getCurrentDeviceType ()`
- `int device_SendDataCommandNEO (IN int cmd, IN int subCmd, IN BYTE *data, IN int dataLen, OUT BYTE *response, IN_OUT int *respLen)`
- `int device_enablePassThrough (int enablePassThrough)`
- `int device_setBurstMode (IN BYTE mode)`
- `int device_setPollMode (IN BYTE mode)`

- int [device_setMerchantRecord](#) (int index, int enabled, char *merchantID, char *merchantURL)
- int [device_pollForToken](#) (IN int timeout, OUT BYTE *respData, IN_OUT int *respDataLen)
- int [device_getMerchantRecord](#) (IN int index, OUT BYTE *record)
- int [device_getMerchantRecord_Len](#) (IN int index, OUT BYTE *record, IN_OUT int *recordLen)
- int [device_getTransactionResults](#) (IDTMSRData *cardData)
- int [device_startTransaction](#) (IN double amount, IN double amtOther, IN int type, IN const int _timeout, IN BYTE *tags, IN int tagsLen)
- void [device_setTransactionExponent](#) (int exponent)
- int [device_activateTransaction](#) (IN const int _timeout, IN BYTE *tags, IN int tagsLen)
- int [device_cancelTransaction](#) ()
- int [device_getRTCDatetime](#) (IN BYTE *dateTime, IN_OUT int *dateTimeLen)
- int [device_setRTCDatetime](#) (IN BYTE *dateTime, IN int dateTimeLen)
- int [device_startRKI](#) (IN const char *caPath, IN int isProduction)
- void [device_setRKI_URL](#) (IN char *rkiURL, IN int rkiURLLen)
- int [device_getSDKWaitTime](#) ()
- void [device_setSDKWaitTime](#) (int waitTime)
- int [device_getThreadStackSize](#) ()
- void [device_setThreadStackSize](#) (int threadSize)
- int [config_getSerialNumber](#) (OUT char *sNumber)
- int [config_getSerialNumber_Len](#) (OUT char *sNumber, IN_OUT int *sNumberLen)
- int [ctls_startTransaction](#) (IN double amount, IN double amtOther, IN int type, IN const int _timeout, IN BYTE *tags, IN int tagsLen)
- int [ctls_activateTransaction](#) (IN const int _timeout, IN BYTE *tags, IN int tagsLen)
- int [ctls_cancelTransaction](#) ()
- int [ctls_retrieveApplicationData](#) (IN BYTE *AID, IN int AIDLen, OUT BYTE *tlv, IN_OUT int *tlvLen)
- int [ctls_setApplicationData](#) (IN BYTE *tlv, IN int tlvLen)
- int [ctls_removeApplicationData](#) (IN BYTE *AID, IN int AIDLen)
- int [ctls_removeAllApplicationData](#) ()
- int [ctls_retrieveAIDList](#) (OUT BYTE *AIDList, IN_OUT int *AIDListLen)
- int [ctls_retrieveTerminalData](#) (OUT BYTE *tlv, IN_OUT int *tlvLen)
- int [ctls_setTerminalData](#) (IN BYTE *tlv, IN int tlvLen)
- int [ctls_retrieveCAPK](#) (IN BYTE *capk, IN int capkLen, OUT BYTE *key, IN_OUT int *keyLen)
- int [ctls_setCAPK](#) (IN BYTE *capk, IN int capkLen)
- int [ctls_removeCAPK](#) (IN BYTE *capk, IN int capkLen)
- int [ctls_removeAllCAPK](#) ()
- int [ctls_retrieveCAPKList](#) (OUT BYTE *keys, IN_OUT int *keysLen)
- int [ctls_setConfigurationGroup](#) (IN BYTE *tlv, IN int tlvLen)
- int [ctls_getConfigurationGroup](#) (IN int group, OUT BYTE *tlv, OUT int *tlvLen)
- int [ctls_getAllConfigurationGroups](#) (OUT BYTE *tlv, IN_OUT int *tlvLen)
- int [ctls_removeConfigurationGroup](#) (int group)
- void [emv_allowFallback](#) (IN int allow)
- void [emv_setAutoAuthenticateTransaction](#) (IN int authenticate)
- void [emv_setAutoCompleteTransaction](#) (IN int complete)
- int [emv_getAutoAuthenticateTransaction](#) ()
- int [emv_getAutoCompleteTransaction](#) ()
- void [emv_setTransactionParameters](#) (IN double amount, IN double amtOther, IN int type, IN int timeout, IN BYTE *tags, IN int tagsLen)
- int [emv_startTransaction](#) (IN double amount, IN double amtOther, IN int exponent, IN int type, IN int timeout, IN BYTE *tags, IN int tagsLen, IN int forceOnline)
- int [emv_activateTransaction](#) (IN int timeout, IN BYTE *tags, IN int tagsLen, IN int forceOnline)
- int [emv_authenticateTransaction](#) (IN BYTE *updatedTLV, IN int updatedTLVLen)
- int [emv_authenticateTransactionWithTimeout](#) (IN int timeout, IN BYTE *updatedTLV, IN int updatedTLVLen)
- int [emv_completeTransaction](#) (IN int commError, IN BYTE *authCode, IN int authCodeLen, IN BYTE *iad, IN int iadLen, IN BYTE *tlvScripts, IN int tlvScriptsLen, IN BYTE *tlv, IN int tlvLen)
- int [emv_cancelTransaction](#) ()

- int [emv_retrieveApplicationData](#) (IN BYTE *AID, IN int AIDLen, OUT BYTE *tlv, IN_OUT int *tlvLen)
- int [emv_setApplicationData](#) (IN BYTE *name, IN int nameLen, IN BYTE *tlv, IN int tlvLen)
- int [emv_setApplicationDataTLV](#) (IN BYTE *tlv, IN int tlvLen)
- int [emv_removeApplicationData](#) (IN BYTE *AID, IN int AIDLen)
- int [emv_removeAllApplicationData](#) ()
- int [emv_retrieveAIDList](#) (OUT BYTE *AIDList, IN_OUT int *AIDListLen)
- int [emv_retrieveTerminalData](#) (OUT BYTE *tlv, IN_OUT int *tlvLen)
- int [emv_setTerminalData](#) (IN BYTE *tlv, IN int tlvLen)
- int [emv_setTerminalMajorConfiguration](#) (IN int configuration)
- int [emv_retrieveCAPK](#) (IN BYTE *capk, IN int capkLen, OUT BYTE *key, IN_OUT int *keyLen)
- int [emv_setCAPK](#) (IN BYTE *capk, IN int capkLen)
- int [emv_removeCAPK](#) (IN BYTE *capk, IN int capkLen)
- int [emv_removeAllCAPK](#) ()
- int [emv_retrieveCAPKList](#) (OUT BYTE *keys, IN_OUT int *keysLen)
- int [emv_retrieveCRL](#) (OUT BYTE *list, IN_OUT int *lssLen)
- int [emv_setCRL](#) (IN BYTE *list, IN int lssLen)
- int [emv_removeCRL](#) (IN BYTE *list, IN int lssLen)
- int [emv_removeAllCRL](#) ()
- int [icc_getICCRReaderStatus](#) (OUT BYTE *status)
- int [icc_powerOnICC](#) (OUT BYTE *ATR, IN_OUT int *inLen)
- int [icc_powerOffICC](#) ()
- int [icc_exchangeAPDU](#) (IN BYTE *c_APDU, IN int cLen, OUT BYTE *reData, IN_OUT int *reLen)
- int [msr_cancelMSRSwipe](#) ()
- int [msr_startMSRSwipe](#) (IN int _timeout)
- int [executeTransaction](#) (WorldPayData *data, [pWP_callback](#) wpCallback, int requestOnly)
- int [forwardTransaction](#) (IN [pWP_callback](#) wpCallback, IN char *forwardID, IN int forwardIDLen, IN char *password, IN int passwordLen, IN int bypassProcessing)
- int [cancelWorldPay](#) ()
- int [executeTransaction_WorldNet](#) (WorldNetData *data, [pWN_callback](#) wnCallback, int requestOnly)
- int [forwardTransaction_WorldNet](#) (IN char *apiKey, IN int apiKeyLen, IN [pWN_callback](#) wnCallback, IN char *forwardID, IN int forwardIDLen, IN char *password, IN int passwordLen, IN int bypassProcessing)
- int [cancelWorldNet](#) ()
- void [parseMSRData](#) (IN BYTE *resData, IN int resLen, IN_OUT IDTMSRData *cardData)

11.15.1 Detailed Description

VP3300 COM API.

VP3300 COM Global API methods.

11.15.2 Macro Definition Documentation

11.15.2.1 IN

```
#define IN
```

INPUT parameter.

11.15.2.2 IN_OUT

```
#define IN_OUT
```

INPUT / OUTPUT PARAMETER.

11.15.2.3 OUT

```
#define OUT
```

OUTPUT parameter.

11.15.3 Typedef Documentation

11.15.3.1 ftpComm_callBack

```
typedef void(* ftpComm_callBack) (int, int, int)
```

Define the comm callback function to get FTP file transfer status

It should be passed as a parameter in a FTP request, Signature (int, int, int) = response code, current block, total blocks
 RESPONSE CODES: 100 = FILE DOWNLOAD STARTED 101 = FILE BLOCK XX OF XX RECEIVED 102 = FILE DOWNLOAD COMPLETED 103 = FILE DOWNLOAD TERMINATED PREMATURELY

11.15.3.2 httpComm_callBack

```
typedef void(* httpComm_callBack) (BYTE *, int)
```

Define the comm callback function to get the async url data

It should be registered using the comm_registerHTTPCallback

11.15.3.3 pCMR_callBack

```
typedef void(* pCMR_callBack) (int, IDTCMRData *)
```

Define the camera callback function to get the image data

It should be registered using the device_registerCameraCallBk,

11.15.3.4 pCSFS_callBack

```
typedef void(* pCSFS_callBack) (BYTE status)
```

Define the card status and front switch callback function to get card and front switch status

It should be registered using the device_registerCardStatusFrontSwitchCallBk,

11.15.3.5 pEMV_callBack

```
typedef void(* pEMV_callBack) (int, int, BYTE *, int, IDTTransactionData *, EMV_Callback *, int)
```

Define the EMV callback function to get the transaction message/data/result.

It should be registered using the emv_registerCallBk,

11.15.3.6 pMessageHotplug

```
typedef void(* pMessageHotplug) (int, int)
```

Define the USB hot-plug callback function to monitor the info when plug in/out the reader.

It should be registered using the registerHotplugCallBk, The first integer parameter is device type, and the second integer parameter is either 0: Device Plugged Out or 1: Device Plugged In

11.15.3.7 pMSR_callBack

```
typedef void(* pMSR_callBack) (int, IDTMSRData)
```

Define the MSR callback function to get the MSR card data

It should be registered using the msr_registerCallBk, this callback function is for backward compatibility

11.15.3.8 pMSR_callBackp

```
typedef void(* pMSR_callBackp) (int, IDTMSRData *)
```

Define the MSR callback function to get pointer to the MSR card data

It should be registered using the msr_registerCallBk, this callback function is recommended instead of pMSR_callBack

11.15.3.9 pPIN_callBack

```
typedef void(* pPIN_callBack) (int, IDTPINData *)
```

Define the PINPad callback function to get the input PIN Pad data

It should be registered using the pin_registerCallBk,

11.15.3.10 pReadDataLog

```
typedef void(* pReadDataLog) (BYTE *, int)
```

Define the read response callback function to monitor the reading response from the reader.

It should be registered using the registerLogCallBk,

11.15.3.11 pRKI_callBack

```
typedef void(* pRKI_callBack) (int, char *)
```

Define the RKI callback function to get the status of the RKI

It should be registered using the device_registerRKICallBk,

11.15.3.12 pSendDataLog

```
typedef void(* pSendDataLog) (BYTE *, int)
```

Define the send command callback function to monitor the sending command into the reader.

It should be registered using the registerLogCallBk,

11.15.3.13 pWN_callBack

```
typedef void(* pWN_callBack) (char *, int, int)
```

Define the Worldnet callback function to get the transaction message/data/result.

11.15.3.14 pWP_callBack

```
typedef void(* pWP_callBack) (char *, int, int)
```

Define the Worldpay callback function to get the transaction message/data/result.

11.15.3.15 v4Comm_callBack

```
typedef void(* v4Comm_callBack) (BYTE, BYTE, BYTE *, int)
```

Define the comm callback function to receive the V4 Protocol packets received by the device from an external source (IP/USB/RS-232) It should be registered using the `comm_registerV4Callback`, Data callback will contain command, sub-command, and data from V4 packet

11.15.4 Function Documentation

11.15.4.1 cancelWorldNet()

```
int cancelWorldNet ( )
```

Cancels WorldNet transaction.

Returns

RETURN_CODE: Values can be parsed with [device_getIDGStatusCodeString\(\)](#)

11.15.4.2 cancelWorldPay()

```
int cancelWorldPay ( )
```

Cancels WorldPay transaction.

Returns

RETURN_CODE: Values can be parsed with [device_getIDGStatusCodeString\(\)](#)

11.15.4.3 comm_registerHTTPCallback()

```
void comm_registerHTTPCallback (
    httpComm_callBack cBack )
```

Register Comm HTTP Async Callback

Parameters

<i>cBack</i>	- HTTP Comm callback
--------------	----------------------

11.15.4.4 comm_registerV4Callback()

```
void comm_registerV4Callback (
    v4Comm_callBack cBack )
```

Register External V4 Protocol commands Callback

Parameters

<i>cBack</i>	- V4 Protocol Comm callback
--------------	-----------------------------

11.15.4.5 config_getSerialNumber()

```
int config_getSerialNumber (
    OUT char * sNumber )
```

DEPRECATED : please use [config_getSerialNumber_Len](#)(OUT char* sNumber, IN_OUT int *sNumberLen)

Polls device for Serial Number

Parameters

<i>sNumber</i>	Returns Serial Number; needs to have at least 64 bytes of memory
----------------	--

Returns

RETURN_CODE: Values can be parsed with [device_getIDGStatusCodeString](#)

11.15.4.6 config_getSerialNumber_Len()

```
int config_getSerialNumber_Len (
    OUT char * sNumber,
    IN_OUT int * sNumberLen )
```

Polls device for Serial Number

Parameters

<i>sNumber</i>	Returns Serial Number
<i>sNumberLen</i>	Length of Serial Number

Returns

RETURN_CODE: Values can be parsed with [device_getIDGStatusCodeString](#)

11.15.4.7 ctls_activateTransaction()

```
int ctls_activateTransaction (
    IN const int _timeout,
    IN BYTE * tags,
    IN int tagsLen )
```

Start CTLS Transaction Request

Authorizes the CTLS 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 stream. Example, tag 9F0C with amount 0x000000000100 would be 0x9F0C06000000000100
<i>tagsLen</i>	The length of tags data buffer.

>>>>>IMPORTANT: parameters for amount and amtOther MUST BE PASSED AS A DOUBLE VALUE WITH DECIMAL POINT. Example, do not pass 1, but instead pass 1.0 or 1.00. Otherwise, results will be unpredictable

Returns

RETURN_CODE: Values can be parsed with [device_getIDGStatusCodeString\(\)](#) Note: if auto poll is on, it will return the error IDG_P2_STATUS_CODE_COMMAND_NOT_ALLOWED

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) DFO1 = 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

11.15.4.8 ctls_cancelTransaction()

```
int ctls_cancelTransaction ( )
```

Cancel EMV Transaction

Cancels the currently executing EMV transaction.

Returns

RETURN_CODE: Values can be parsed with [device_getIDGStatusCodeString\(\)](#)

11.15.4.9 ctls_getAllConfigurationGroups()

```
int ctls_getAllConfigurationGroups (
    OUT BYTE * tlv,
    IN_OUT int * tlvLen )
```

Retrieve All Configuration Groups

Returns all the Configuration Groups installed on the terminal for CTLS

Parameters

<i>tlv</i>	The TLV elements data
<i>tlvLen</i>	the length of tlv data buffer.

Returns

RETURN_CODE: Values can be parsed with [device_getIDGStatusCodeString\(\)](#)

11.15.4.10 ctls_getConfigurationGroup()

```
int ctls_getConfigurationGroup (
    IN int group,
    OUT BYTE * tlv,
    OUT int * tlvLen )
```

Get Configuration Group

Retrieves the Configuration for the specified Group.

Parameters

<i>group</i>	Configuration Group
<i>tlv</i>	return data
<i>tlvLen</i>	the length of tlv data buffer

Returns

RETURN_CODE: Values can be parsed with [device_getIDGStatusCodeString\(\)](#)

11.15.4.11 ctls_registerCallBk()

```
void ctls_registerCallBk (
    pMSR_callback pCTLSf )
```

To register the ctls callback function to get the MSR card data. (Pass NULL to disable the callback.)

11.15.4.12 ctls_registerCallBkp()

```
void ctls_registerCallBkp (
    pMSR_callback pCTLSf )
```

To register the ctls callback function to get the MSR card data pointer. (Pass NULL to disable the callback.)

11.15.4.13 ctls_removeAllApplicationData()

```
int ctls_removeAllApplicationData ( )
```

Remove All Application Data

Removes all the Application Data

Returns

RETURN_CODE: Values can be parsed with [device_getIDGStatusCodeString\(\)](#)

11.15.4.14 ctls_removeAllCAPK()

```
int ctls_removeAllCAPK ( )
```

Remove All Certificate Authority Public Key

Removes all the CAPK

Returns

RETURN_CODE: Values can be parsed with [device_getIDGStatusCodeString\(\)](#)

11.15.4.15 ctls_removeApplicationData()

```
int ctls_removeApplicationData (
    IN BYTE * AID,
    IN int AIDLLen )
```

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>AIDLLen</i>	the length of AID data buffer

Returns

RETURN_CODE: Values can be parsed with [device_getIDGStatusCodeString\(\)](#)

11.15.4.16 ctls_removeCAPK()

```
int ctls_removeCAPK (
    IN BYTE * capk,
    IN int capkLen )
```

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>capkLen</i>	the length of capk data buffer

Returns

RETURN_CODE: Values can be parsed with [device_getIDGStatusCodeString\(\)](#)

11.15.4.17 ctls_removeConfigurationGroup()

```
int ctls_removeConfigurationGroup (
    int group )
```

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>	Values can be parsed with device_getIDGStatusCodeString()
--------------------	---

11.15.4.18 ctls_retrieveAIDList()

```
int ctls_retrieveAIDList (
    OUT BYTE * AIDList,
    IN_OUT int * AIDListLen )
```

Retrieve AID list

Returns all the AID names installed on the terminal for CTLS. .

Parameters

<i>AIDList</i>	array of AID name byte arrays
<i>AIDListLen</i>	the length of AIDList array buffer

Returns

RETURN_CODE: Values can be parsed with [device_getIDGStatusCodeString\(\)](#)

11.15.4.19 ctls_retrieveApplicationData()

```
int ctls_retrieveApplicationData (
    IN BYTE * AID,
    IN int AIDLen,
    OUT BYTE * tlv,
    IN_OUT int * tlvLen )
```

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>AIDLen</i>	the length of AID data buffer.
<i>tlv</i>	The TLV elements of the requested AID
<i>tlvLen</i>	the length of tlv data buffer.

Returns

RETURN_CODE: Values can be parsed with [device_getIDGStatusCodeString\(\)](#)

11.15.4.20 ctls_retrieveCAPK()

```
int ctls_retrieveCAPK (
    IN BYTE * capk,
    IN int capkLen,
    OUT BYTE * key,
    IN_OUT int * keyLen )
```

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>capkLen</i>	the length of capk data buffer

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>keyLen</i>	<p>the length of key data buffer</p> <ul style="list-style-type: none"> •

Returns

RETURN_CODE: Values can be parsed with [device_getIDGStatusCodeString\(\)](#)

11.15.4.21 `ctls_retrieveCAPKList()`

```
int ctls_retrieveCAPKList (
    OUT BYTE * keys,
    IN_OUT int * keyLen )
```

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>keyLen</i>	the length of keys data buffer

Returns

RETURN_CODE: Values can be parsed with [device_getIDGStatusCodeString\(\)](#)

11.15.4.22 `ctls_retrieveTerminalData()`

```
int ctls_retrieveTerminalData (
    OUT BYTE * tlv,
    IN_OUT int * tlvLen )
```

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>tlvLen</i>	the length of tlv data buffer

Returns

RETURN_CODE: Values can be parsed with [device_getIDGStatusCodeString\(\)](#)

11.15.4.23 `ctls_setApplicationData()`

```
int ctls_setApplicationData (  
    IN BYTE * tlv,  
    IN int tlvLen )
```

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>tlvLen</i>	the length of tlv data buffer
---------------	-------------------------------

Returns

RETURN_CODE: Values can be parsed with [device_getIDGStatusCodeString\(\)](#)

11.15.4.24 `ctls_setCAPK()`

```
int ctls_setCAPK (  
    IN BYTE * capk,  
    IN int capkLen )
```

Set Certificate Authority Public Key

Sets the CAPK as specified by the CAKey structure

Parameters

<i>capk</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>capkLen</i>	the length of capk data buffer

Returns

RETURN_CODE: Values can be parsed with [device_getIDGStatusCodeString\(\)](#)

11.15.4.25 `ctls_setConfigurationGroup()`

```
int ctls_setConfigurationGroup (
    IN BYTE * tlv,
    IN int tlvLen )
```

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>tlvLen</i>	the length of tlv data buffer

Returns

RETURN_CODE: Values can be parsed with [device_getIDGStatusCodeString\(\)](#)

11.15.4.26 `ctls_setTerminalData()`

```
int ctls_setTerminalData (
    IN BYTE * tlv,
    IN int tlvLen )
```

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 `ctls_getConfigurationGroup(int group)`, and deleted with `ctls_removeConfigurationGroup(int group)`. You cannot delete group 0.

Parameters

<i>tlv</i>	TerminalData configuration file
<i>tlvLen</i>	the length of tlv data buffer

Return values

RETURN_CODE	Values can be parsed with <code>device_getIDGStatusCodeString()</code>
--------------------	--

11.15.4.27 ctls_startTransaction()

```
int ctls_startTransaction (
    IN double amount,
    IN double amtOther,
    IN int type,
    IN const int _timeout,
    IN BYTE * tags,
    IN int tagsLen )
```

Start CTLS Transaction Request

Authorizes the CTLS transaction for an ICC card

The tags will be returned in the callback routine.

Parameters

<i>amount</i>	Transaction amount value (tag value 9F02) - SEE IMPORTANT NOTE BELOW
<i>amtOther</i>	Other amount value, if any (tag value 9F03) - SEE IMPORTANT NOTE BELOW
<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 stream. Example, tag 9F0C with amount 0x000000000100 would be 0x9F0C06000000000100 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>tagsLen</i>	The length of tags data buffer.

>>>>>IMPORTANT: parameters for amount and amtOther MUST BE PASSED AS A DOUBLE VALUE WITH DECIMAL POINT. Example, do not pass 1, but instead pass 1.0 or 1.00. Otherwise, results will be unpredictable

Returns

RETURN_CODE: Values can be parsed with `device_getIDGStatusCodeString()` Note: if auto poll is on, it will return the error IDG_P2_STATUS_CODE_COMMAND_NOT_ALLOWED

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

11.15.4.28 device_activateTransaction()

```
int device_activateTransaction (
    IN const int _timeout,
    IN BYTE * tags,
    IN int tagsLen )
```

Start Transaction Request

Authorizes the transaction for an MSR/CTLS/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. Example, tag 9F0C with amount 0x000000000100 would be 0x9F0C06000000000100 Be sure to include 9F02 (amount) and 9C (transaction type).
<i>tagsLen</i>	The length of tags data buffer.

>>>>>IMPORTANT: parameters for amount and amtOther MUST BE PASSED AS A DOUBLE VALUE WITH DECIMAL POINT. Example, do not pass 1, but instead pass 1.0 or 1.00. Otherwise, results will be unpredictable

Returns

RETURN_CODE: Values can be parsed with [device_getIDGStatusCodeString\(\)](#) Note: if auto poll is on, it will return the error IDG_P2_STATUS_CODE_COMMAND_NOT_ALLOWED

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 device_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

11.15.4.29 device_cancelTransaction()

```
int device_cancelTransaction ( )
```

Disable Transaction Cancel Transaction request.

Returns

RETURN_CODE: Values can be parsed with [device_getIDGStatusCodeString\(\)](#)

11.15.4.30 device_close()

```
int device_close ( )
```

Close the device

Returns

RETURN_CODE: 0: success, 0x0A: failed

11.15.4.31 device_controlUserInterface()

```
int device_controlUserInterface (
    IN BYTE * values )
```

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) • 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
---------------	--

Returns

RETURN_CODE: Values can be parsed with [device_getIDGStatusCodeString\(\)](#)

11.15.4.32 device_enablePassThrough()

```
int device_enablePassThrough (
    int enablePassThrough )
```

Enable Pass Through

Enables Pass Through Mode for direct communication with L1 interface (power on icc, send apdu, etc).

Parameters

<i>enablePassThrough</i>	1 = pass through ON, 0 = pass through OFF
--------------------------	---

Returns

RETURN_CODE: Values can be parsed with [device_getIDGStatusCodeString\(\)](#)

11.15.4.33 device_getCurrentDeviceType()

```
int device_getCurrentDeviceType ( )
```

Get current active device type

Returns

: return the device type defined as DEVICE_TYPE in the IDTDef.h

11.15.4.34 device_getFirmwareVersion()

```
int device_getFirmwareVersion (
    OUT char * firmwareVersion )
```

DEPRECATED : please use [device_getFirmwareVersion_Len\(\)](#)(OUT char* firmwareVersion, IN_OUT int *firmwareVersionLen)

Polls device for Firmware Version

Parameters

<i>firmwareVersion</i>	Response returned of Firmware Version; needs to have at least 128 bytes of memory
------------------------	---

Returns

RETURN_CODE: Values can be parsed with [device_getIDGStatusCodeString\(\)](#)

11.15.4.35 device_getFirmwareVersion_Len()

```
int device_getFirmwareVersion_Len (
    OUT char * firmwareVersion,
    IN_OUT int * firmwareVersionLen )
```

Polls device for Firmware Version

Parameters

<i>firmwareVersion</i>	Response returned of Firmware Version
<i>firmwareVersionLen</i>	Length of Firmware Version

Returns

RETURN_CODE: Values can be parsed with [device_getIDGStatusCodeString\(\)](#)

11.15.4.36 device_getIDGStatusCodeString()

```
void device_getIDGStatusCodeString (
    IN int  returnCode,
    OUT char * despcrition )
```

Review the return code description.

Parameters

<i>returnCode</i>	the response result.
<i>description</i>	

Return values

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

- 0: "no error, beginning task";
- 1: "no response from reader";
- 2: "invalid response data";
- 01: " Incorrect Header Tag";
- 02: " Unknown Command";
- 03: " Unknown Sub-Command";
- 04: " CRC Error in Frame";
- 05: " Incorrect Parameter";
- 06: " Parameter Not Supported";
- 07: " Mal-formatted Data";
- 08: " Timeout";
- 0A: " Failed / NACK";
- 0B: " Command not Allowed";
- 0C: " Sub-Command not Allowed";
- 0D: " Buffer Overflow (Data Length too large for reader buffer)";

- 0E: " User Interface Event";
- 10: " Need clear firmware(apply in boot loader only)";
- 11: " Communication type not supported, VT-1, burst, etc. Need encrypted firmware (apply in boot loader only)";
- 12: " Secure interface is not functional or is in an intermediate state.";
- 13: " Data field is not mod 8";
- 14: " Pad 0x80 not found where expected";
- 15: " Specified key type is invalid";
- 16: " Could not retrieve key from the SAM (InitSecureComm)";
- 17: " Hash code problem";
- 18: " Could not store the key into the SAM (InstallKey)";
- 19: " Frame is too large";
- 1A: " Unit powered up in authentication state but POS must resend the InitSecureComm command";
- 1B: " The EEPROM may not be initialized because SecCommInterface does not make sense";
- 1C: " Problem encoding APDU Module-Specific Status Codes ";
- 20: " Unsupported Index (ILM) SAM Transceiver error - problem communicating with the SAM (Key Mgr)";
- 21: " Unexpected Sequence Counter in multiple frames for single bitmap (ILM)Length error in data returned from the SAM (Key Mgr) - 22: " Improper bit map (ILM)"; - 23: " Request Online Authorization"; - 24: " ViV↔ OCard3 raw data read successful"; - 25: " Message index not available (ILM) ViVOcomm activate transaction card type (ViVOcomm)"; - 26: " Version Information Mismatch (ILM)"; - 27: " Not sending commands in correct index message index (ILM)"; - 28: " Time out or next expected message not received (ILM)"; - 29: " ILM languages not available for viewing (ILM)"; - 2A: " Other language not supported (ILM)"; - 41: " from 41 to 4F, Module-specific errors for Key Manager";
- 50: " Auto-Switch OK";
- 51: " Auto-Switch failed";
- 70: " Antenna Error 80h Use another card";
- 81: " Insert or swipe card";
- 90: " Data encryption Key does not exist";
- 91: " Data encryption Key KSN exhausted";

11.15.4.37 device_getMerchantRecord()

```
int device_getMerchantRecord (
    IN int index,
    OUT BYTE * record )
```

DEPRECATED : please use device_getMerchantRecord_Len(IN int index, OUT BYTE * record, IN_OUT int *recordLen)

Get Merchant Record

Gets the merchant record for the device.

Parameters

<i>index</i>	Merchant Record index, valid values 1-6
<i>record;</i>	needs to have at least 99 bytes of memory response data from reader. Merchant Record Index: 1 byte enabled: 1 byte Merchant ID: 32 bytes Length of Merchant URL: 1 byte Merchant URL: 64 bytes

Returns

success or error code. Values can be parsed with [device_getIDGStatusCodeString\(\)](#)

See also

[ErrorCode](#)

11.15.4.38 device_getMerchantRecord_Len()

```
int device_getMerchantRecord_Len (
    IN int index,
    OUT BYTE * record,
    IN_OUT int * recordLen )
```

Get Merchant Record

Gets the merchant record for the device.

Parameters

<i>index</i>	Merchant Record index, valid values 1-6
<i>record</i>	response data from reader. Merchant Record Index: 1 byte enabled: 1 byte Merchant ID: 32 bytes Length of Merchant URL: 1 byte Merchant URL: 64 bytes
<i>recordLen</i>	Length of record

Returns

success or error code. Values can be parsed with [device_getIDGStatusCodeString\(\)](#)

See also

[ErrorCode](#)

11.15.4.39 device_getRTCDateTime()

```
int device_getRTCDateTime (
    IN BYTE * dateTime,
    IN_OUT int * dateTimeLen )
```

get RTC date and time of the device

Parameters

<i>dateTime</i>	<dateTime data>="" is: 6 byte data for YYMMDDHHMMSS in hex. For example 0x171003102547 stands for 2017 Oct 3rd 10:25:47
<i>dateTimeLen</i>	return 6 bytes if successful

Returns

success or error code. Values can be parsed with device_getResponseCodeString

See also

ErrorCode

11.15.4.40 device_getSDKWaitTime()

```
int device_getSDKWaitTime ( )
```

Get SDK Wait Time

Get the SDK wait time for transactions

Returns

SDK wait time in seconds

11.15.4.41 device_getThreadStackSize()

```
int device_getThreadStackSize ( )
```

Get Thread Stack Size

Get the stack size setting for newly created threads

Returns

Thread Stack Size

11.15.4.42 device_getTransactionResults()

```
int device_getTransactionResults (
    IDTMSRData * cardData )
```

Get Transaction Results Gets the transaction results when the reader is functioning in "Auto Poll" mode

Parameters

<i>cardData</i>	The transaction results
-----------------	-------------------------

Returns

success or error code. Values can be parsed with device_getResponseCodeString

See also

ErrorCode

11.15.4.43 device_init()

```
int device_init ( )
```

Initial the device by USB

It will detect the device and trying connect.

The connect status can be checked by [device_isConnected\(\)](#).

Note: after the function returns success, the function [device_setCurrentDevice\(\)](#) has to be called to set the device type.

Returns

RETURN_CODE: Values can be parsed with [device_getIDGStatusCodeString\(\)](#)

11.15.4.44 device_isAttached()

```
int device_isAttached (
    int deviceType )
```

Check if the device is attached to the USB port The function [device_init\(\)](#) must be called before this function.

Parameters

<i>deviceType,the</i>	device type of the USB device
-----------------------	-------------------------------

Returns

1 if the device is attached, or 0 if the device is not attached

11.15.4.45 device_isConnected()

```
int device_isConnected ( )
```

Check the device connected status

Returns

DEVICE_DISCONNECT=0, or DEVICE_CONNECTED = 1

11.15.4.46 device_pingDevice()

```
int device_pingDevice ( )
```

Ping Device

Pings the reader. If connected, returns success. Otherwise, returns timeout.

Returns

RETURN_CODE: Values can be parsed with [device_getIDGStatusCodeString\(\)](#)

11.15.4.47 device_pollForToken()

```
int device_pollForToken (
    IN int timeout,
    OUT BYTE * respData,
    IN_OUT int * respDataLen )
```

Poll for Token

Polls for a PICC

Parameters

<i>timeout</i>	timeout in milliseconds, must be multiple of 10 milliseconds. 30, 120, 630, or 1150 for example.
<i>respData</i>	Response data will be stored in respData. 1 byte of card type, and the Serial Number (or the UID) of the PICC if available.
<i>respDataLen</i>	Length of systemCode.

Returns

RETURN_CODE: Values can be parsed with [device_getIDGStatusCodeString\(\)](#)

11.15.4.48 device_registerCameraCallBk()

```
void device_registerCameraCallBk (
    pCMR_callBack pCMRf )
```

To register the camera callback function to get the image data. (Pass NULL to disable the callback.)

11.15.4.49 device_registerCardStatusFrontSwitchCallBk()

```
void device_registerCardStatusFrontSwitchCallBk (
    pCSFS_callBack pCSFSf )
```

To register the card status and front switch callback function to get status. (Pass NULL to disable the callback.)

11.15.4.50 device_registerRKICallBk()

```
void device_registerRKICallBk (
    pRKI_callBack pRKIf )
```

To register the RKI callback function to get the RKI status. (Pass NULL to disable the callback.)

11.15.4.51 device_SendDataCommandNEO()

```
int device_SendDataCommandNEO (
    IN int cmd,
    IN int subCmd,
    IN BYTE * data,
    IN int dataLen,
    OUT BYTE * response,
    IN_OUT int * respLen )
```

Send a Command to device

Sends a command to the device .

Parameters

<i>cmd</i>	buffer of command to execute.
<i>cmdLen,the</i>	length of the buffer cmd.
<i>data</i>	buffer of IDG command data.
<i>dataLen,the</i>	length of the buffer data.
<i>response</i>	Response data
<i>respLen,the</i>	length of Response data

Returns

RETURN_CODE: Values can be parsed with [device_getIDGStatusCodeString\(\)](#) Send a Command to NEO device

Sends a command to the NEO device .

Parameters

<i>cmd</i>	command to execute.
<i>subCmd,sub</i>	command to execute.
<i>data</i>	buffer of NEO command data.
<i>dataLen,the</i>	length of the buffer data.
<i>response</i>	Response data
<i>respLen,the</i>	length of Response data

Returns

RETURN_CODE: Values can be parsed with [device_getIDGStatusCodeString\(\)](#)

11.15.4.52 device_setBurstMode()

```
int device_setBurstMode (
    IN BYTE mode )
```

Send Burst Mode

Sets the burst mode for the device.

Parameters

<i>mode</i>	0 = OFF, 1 = Always On, 2 = Auto Exit
-------------	---------------------------------------

Returns

success or error code. Values can be parsed with [device_getIDGStatusCodeString](#)

See also

ErrorCode

11.15.4.53 device_setCurrentDevice()

```
int device_setCurrentDevice (
    int deviceType )
```

Sets the current device to talk to

The connect status can be checked by [device_isConnected\(\)](#).

Parameters

<i>deviceType</i>	<p>Device to connect to</p> <pre>enum DEVICE_TYPE { IDT_DEVICE_UNKNOWN=0, IDT_DEVICE_AUGUSTA_HID, IDT_DEVICE_AUGUSTA_KB, IDT_DEVICE_AUGUSTA_S_HID, IDT_DEVICE_AUGUSTA_S_KB, IDT_DEVICE_AUGUSTA_S_TTK_HID, IDT_DEVICE_SPECTRUM_PRO, IDT_DEVICE_MINISMART_II, IDT_DEVICE_L80, IDT_DEVICE_L100, IDT_DEVICE_UNIPAY, IDT_DEVICE_UNIPAY_I_V, IDT_DEVICE_VP3300_AJ, IDT_DEVICE_KIOSK_III, IDT_DEVICE_KIOSK_III_S, IDT_DEVICE_PIP_READER, IDT_DEVICE_VENDI, IDT_DEVICE_VP3300_USB, IDT_DEVICE_UNIPAY_I_V_TTK, IDT_DEVICE_VP3300_BT, IDT_DEVICE_VP8800, IDT_DEVICE_SREDKEY2_HID, IDT_DEVICE_SREDKEY2_KB, IDT_DEVICE_NEO2, IDT_DEVICE_MINISMART_II_COM = IDT_DEVICE_NEO2+5, IDT_DEVICE_SPECTRUM_PRO_COM, IDT_DEVICE_KIOSK_III_COM, IDT_DEVICE_KIOSK_III_S_COM, IDT_DEVICE_PIP_READER_COM, IDT_DEVICE_VP3300_COM, IDT_DEVICE_NEO2_COM, IDT_DEVICE_MAX_DEVICES = IDT_DEVICE_NEO2_COM+5 };</pre>
-------------------	---

Returns

RETURN_CODE: 1: success, 0: failed

11.15.4.54 device_setMerchantRecord()

```
int device_setMerchantRecord (
    int index,
    int enabled,
    char * merchantID,
    char * merchantURL )
```

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

Returns

RETURN_CODE: Values can be parsed with [device_getIDGStatusCodeString\(\)](#)

11.15.4.55 device_setPollMode()

```
int device_setPollMode (
    IN BYTE mode )
```

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>	0 = Auto Poll, 1 = Poll On Demand
-------------	-----------------------------------

Returns

RETURN_CODE: Values can be parsed with [device_getIDGStatusCodeString\(\)](#)

11.15.4.56 device_setRKI_URL()

```
void device_setRKI_URL (
    IN char * rkiURL,
    IN int rkiURLLen )
```

Set the URL for RKI

Parameters

<i>rkiURL</i>	The URL for RKI (less than 100 characters). Pass NULL to reset to default URL
<i>rkiURLLen</i>	The length of rkiURL. Pass 0 to reset to default URL

Returns

11.15.4.57 device_setRTCDateTime()

```
int device_setRTCDateTime (
    IN BYTE * dateTime,
    IN int dateTimeLen )
```

set RTC date and time of the device

Parameters

<i>dateTime</i>	<dateTime data>=""> is: 6 byte data for YYMMDDHHMMSS in hex. For example 0x171003102547 stands for 2017 Oct 3rd 10:25:47
<i>dateTimeLen</i>	should be always 6 bytes

Returns

success or error code. Values can be parsed with [device_getResponseCodeString](#)

See also

[ErrorCode](#)

11.15.4.58 device_setSDKWaitTime()

```
void device_setSDKWaitTime (
    int waitTime )
```

Set SDK Wait Time

Set the SDK wait time for transactions

Parameters

<i>waitTime</i>	The SDK wait time for transaction in seconds. The Maximum is 2147483 seconds.
-----------------	---

11.15.4.59 device_setThreadStackSize()

```
void device_setThreadStackSize (
    int threadSize )
```

Set Thread Stack Size

Set the stack size setting for newly created threads

11.15.4.60 device_setTransactionExponent()

```
void device_setTransactionExponent (
    int exponent )
```

Sets the transaction exponent to be used with device_startTransaction. Default value is 2

Parameters

<i>exponent, The</i>	exponent to use when calling device_startTransaction
----------------------	--

11.15.4.61 device_startRKI()

```
int device_startRKI (
    IN const char * caPath,
    IN int isProduction )
```

Start remote key injection.

Parameters

<i>caPath</i>	The path to ca-certificates.crt It should be NULL, because the file is not used anymore.
<i>isProduction</i>	1: The reader is a production unit, 0: The reader is not a production unit

Returns

success or error code.

See also

ErrorCode

11.15.4.62 device_startTransaction()

```
int device_startTransaction (
    IN double amount,
    IN double amtOther,
    IN int type,
    IN const int _timeout,
    IN BYTE * tags,
    IN int tagsLen )
```

Start device Transaction Request

Authorizes the transaction for an MSR/CTLS/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>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. Example, tag 9F0C with amount 0x000000000100 would be 0x9F0C06000000000100 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 = DFEE1A079F029F36959F37

Returns

success or error code. Values can be parsed with device_getResponseCodeString

See also

ErrorCode Note: if auto poll is on, it will return the error IDG_P2_STATUS_CODE_COMMAND_NOT_ALLOWED

11.15.4.63 emv_activateTransaction()

```
int emv_activateTransaction (
    IN int timeout,
    IN BYTE * tags,
    IN int tagsLen,
    IN int forceOnline )
```

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>	Tags to be included in the request. Passed as a TLV stream. Example, tag 9F0C with amount 0x000000000100 would be 0x9F0C06000000000100
<i>tagsLen</i>	Length of tags
<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 = DFEE1A079F029F36959F37

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString >>>>IMPORTANT: parameters for amount and amtOther MUST BE PASSED AS A DOUBLE VALUE WITH DECIMAL POINT. Example, do not pass 1, but instead pass 1.0 or 1.00. Otherwise, results will be unpredictable

11.15.4.64 emv_allowFallback()

```
void emv_allowFallback (
    IN int allow )
```

Allow fallback for EMV transactions. Default is TRUE

Parameters

<i>allow</i>	TRUE = allow fallback, FALSE = don't allow fallback
--------------	---

11.15.4.65 emv_authenticateTransaction()

```
int emv_authenticateTransaction (
    IN BYTE * updatedTLV,
    IN int updatedTLVLen )
```

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 = DFEE1A079F029F36959F37
-------------------	---

Parameters

<i>updatedTLVLen</i>	
----------------------	--

Returns

RETURN_CODE: Values can be parsed with `device_getResponseCodeString`

11.15.4.66 `emv_authenticateTransactionWithTimeout()`

```
int emv_authenticateTransactionWithTimeout (
    IN int timeout,
    IN BYTE * updatedTLV,
    IN int updatedTLVLen )
```

Authenticate EMV Transaction Request with Timeout

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>timeout</i>	Timeout value in seconds.
<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 = DFEE1A079F029F36959F37
<i>updatedTLVLen</i>	

Returns

RETURN_CODE: Values can be parsed with `device_getResponseCodeString`

11.15.4.67 `emv_cancelTransaction()`

```
int emv_cancelTransaction ( )
```

Cancel EMV Transaction

Cancels the currently executing EMV transaction.

Returns

RETURN_CODE: Values can be parsed with `device_getResponseCodeString()`

11.15.4.68 emv_completeTransaction()

```
int emv_completeTransaction (
    IN int commError,
    IN BYTE * authCode,
    IN int authCodeLen,
    IN BYTE * iad,
    IN int iadLen,
    IN BYTE * tlvScripts,
    IN int tlvScriptsLen,
    IN BYTE * tlv,
    IN int tlvLen )
```

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(1) if host was unreachable, or FALSE(0) 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>authCodeLen</i>	the length of authCode
<i>iad</i>	Issuer Authentication Data, if any. Example 0x11223344556677883030 (tag value 91).
<i>iadLen</i>	the length of iadLen
<i>tlvScripts</i>	71/72 scripts, if any
<i>tlvScriptsLen</i>	the length of tlvScriptsLen
<i>tlv</i>	Additional TLV data to return with transaction results (if any)
<i>tlvLen</i>	the length of tlv

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString

11.15.4.69 emv_getAutoAuthenticateTransaction()

```
int emv_getAutoAuthenticateTransaction ( )
```

Gets auto authenticate value for EMV transactions.

Returns

RETURN_CODE: TRUE = auto authenticate, FALSE = manually authenticate

11.15.4.70 emv_getAutoCompleteTransaction()

```
int emv_getAutoCompleteTransaction ( )
```

Gets auto complete value for EMV transactions.

Returns

RETURN_CODE: TRUE = auto complete, FALSE = manually complete

11.15.4.71 emv_registerCallBk()

```
void emv_registerCallBk (
    pEMV_callback pEMVf )
```

To register the emv callback function to get the EMV processing response. (Pass NULL to disable the callback.)

11.15.4.72 emv_removeAllApplicationData()

```
int emv_removeAllApplicationData ( )
```

Remove All Application Data

Removes all the Application Data

Returns

RETURN_CODE: Values can be parsed with [device_getIDGStatusCodeString\(\)](#)

11.15.4.73 emv_removeAllCAPK()

```
int emv_removeAllCAPK ( )
```

Remove All Certificate Authority Public Key

Removes all the CAPK

Returns

RETURN_CODE: Values can be parsed with [device_getIDGStatusCodeString\(\)](#)

11.15.4.74 emv_removeAllCRL()

```
int emv_removeAllCRL ( )
```

Remove All Certificate Revocation List Entries

Removes all CRLentry entries

Returns

RETURN_CODE: Values can be parsed with [device_getResponseCodeString\(\)](#)

11.15.4.75 emv_removeApplicationData()

```
int emv_removeApplicationData (
    IN BYTE * AID,
    IN int AIDLen )
```

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>AIDLen</i>	the length of AID data buffer

Returns

RETURN_CODE: Values can be parsed with [device_getIDGStatusCodeString\(\)](#)

11.15.4.76 emv_removeCAPK()

```
int emv_removeCAPK (
    IN BYTE * capk,
    IN int capkLen )
```

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>capkLen</i>	the length of capk data buffer

Returns

RETURN_CODE: Values can be parsed with [device_getIDGStatusCodeString\(\)](#)

11.15.4.77 emv_removeCRL()

```
int emv_removeCRL (
    IN BYTE * list,
    IN int lsLen )
```

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>lsLen</i>	the length of list data buffer

Returns

RETURN_CODE: Values can be parsed with [device_getResponseCodeString\(\)](#)

11.15.4.78 emv_retrieveAIDList()

```
int emv_retrieveAIDList (
    OUT BYTE * AIDList,
    IN_OUT int * AIDListLen )
```

Retrieve AID list

Returns all the AID names installed on the terminal for CTLS. .

Parameters

<i>AIDList</i>	array of AID name byte arrays
<i>AIDListLen</i>	the length of AIDList array buffer

Returns

RETURN_CODE: Values can be parsed with [device_getIDGStatusCodeString\(\)](#)

11.15.4.79 emv_retrieveApplicationData()

```
int emv_retrieveApplicationData (
    IN BYTE * AID,
    IN int AIDLen,
    OUT BYTE * tlv,
    IN_OUT int * tlvLen )
```

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>AIDLen</i>	the length of AID data buffer.
<i>tlv</i>	The TLV elements of the requested AID
<i>tlvLen</i>	the length of tlv data buffer.

Returns

RETURN_CODE: Values can be parsed with [device_getIDGStatusCodeString\(\)](#)

11.15.4.80 emv_retrieveCAPK()

```
int emv_retrieveCAPK (
    IN BYTE * capk,
    IN int capkLen,
    OUT BYTE * key,
    IN_OUT int * keyLen )
```

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>capkLen</i>	the length of capk data buffer
<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>keyLen</i>	<p>the length of key data buffer</p> <ul style="list-style-type: none"> •

Returns

RETURN_CODE: Values can be parsed with [device_getIDGStatusCodeString\(\)](#)

11.15.4.81 emv_retrieveCAPKList()

```
int emv_retrieveCAPKList (
    OUT BYTE * keys,
    IN_OUT int * keysLen )
```

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>keysLen</i>	the length of keys data buffer

Returns

RETURN_CODE: Values can be parsed with [device_getIDGStatusCodeString\(\)](#)

11.15.4.82 emv_retrieveCRL()

```
int emv_retrieveCRL (
```

```

    OUT BYTE * list,
    IN_OUT int * lssLen )

```

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>lssLen</i>	the length of list data buffer

Returns

RETURN_CODE: Values can be parsed with [device_getResponseCodeString\(\)](#)

11.15.4.83 emv_retrieveTerminalData()

```

int emv_retrieveTerminalData (
    OUT BYTE * tlv,
    IN_OUT int * tlvLen )

```

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>tlvLen</i>	the length of tlv data buffer

Returns

RETURN_CODE: Values can be parsed with [device_getIDGStatusCodeString\(\)](#)

11.15.4.84 emv_setApplicationData()

```

int emv_setApplicationData (
    IN BYTE * name,
    IN int nameLen,
    IN BYTE * tlv,
    IN int tlvLen )

```

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>nameLen</i>	the length of name data buffer of Application name,
<i>tlv</i>	Application data in TLV format

Parameters

<i>tlvLen</i>	the length of tlv data buffer
---------------	-------------------------------

Returns

RETURN_CODE: Values can be parsed with [device_getResponseCodeString\(\)](#)

11.15.4.85 **emv_setApplicationDataTLV()**

```
int emv_setApplicationDataTLV (
    IN BYTE * tlv,
    IN int tlvLen )
```

Set Application Data by TLV

Sets the Application Data as specified by the TLV data

Parameters

<i>tlv</i>	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) Example valid TLV, for Group #2, AID a0000000035010: "dfee2d01029f0607a0000000051010ffe10101ffe50110ffe30114ffe20106"
<i>tlvLen</i>	the length of tlv data buffer

Returns

RETURN_CODE: Values can be parsed with [device_getResponseCodeString\(\)](#)

11.15.4.86 **emv_setAutoAuthenticateTransaction()**

```
void emv_setAutoAuthenticateTransaction (
    IN int authenticate )
```

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

Parameters

<i>authenticate</i>	TRUE = auto authenticate, FALSE = manually authenticate
---------------------	---

11.15.4.87 **emv_setAutoCompleteTransaction()**

```
void emv_setAutoCompleteTransaction (
    IN int complete )
```

Enables complete for EMV transactions. If a `emv_authenticateTransaction` results in code 0x0004 (go online), then `emv_completeTransaction` can automatically execute if parameter is set to TRUE

Parameters

<i>complete</i>	TRUE = auto complete, FALSE = manually complete
-----------------	---

11.15.4.88 emv_setCAPK()

```
int emv_setCAPK (
    IN BYTE * capk,
    IN int capkLen )
```

Set Certificate Authority Public Key

Sets the CAPK as specified by the CAKey structure

Parameters

<i>capk</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>capkLen</i>	the length of capk data buffer

Returns

RETURN_CODE: Values can be parsed with [device_getIDGStatusCodeString\(\)](#)

11.15.4.89 emv_setCRL()

```
int emv_setCRL (
    IN BYTE * list,
    IN int lsLen )
```

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>lsLen</i>	the length of list data buffer

Returns

RETURN_CODE: Values can be parsed with [device_getResponseCodeString\(\)](#)

11.15.4.90 emv_setTerminalData()

```
int emv_setTerminalData (
    IN BYTE * tlv,
    IN int tlvLen )
```

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 file
<i>tlvLen</i>	the length of tlv data buffer

Return values

<i>RETURN_CODE</i>	Values can be parsed with device_getIDGStatusCodeString()
--------------------	---

11.15.4.91 emv_setTerminalMajorConfiguration()

```
int emv_setTerminalMajorConfiguration (
    IN int configuration )
```

Sets the terminal major configuration in ICS .

Parameters

<i>configuration</i>	A configuration value, range 1-23 <ul style="list-style-type: none">• 1 = 1C• 2 = 2C• 3 = 3C• 4 = 4C• 5 = 5C ...• 23 = 23C
----------------------	---

Returns

RETURN_CODE: Values can be parsed with [device_getResponseCodeString\(\)](#)

11.15.4.92 emv_setTransactionParameters()

```
void emv_setTransactionParameters (
    IN double amount,
    IN double amtOther,
    IN int type,
    IN int timeout,
    IN BYTE * tags,
    IN int tagsLen )
```

Set EMV Transaction Parameters

Set the parameters to be used on EMV transactions for an ICC card when Auto Poll is on

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 (Maximum Length = 500 bytes). Passed as a string. Example, tag 9F0C with amount 0x000000000100 would be "9F0C06000000000100" 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 = DFEE1A079F029F36959F37
<i>tagsLen</i>	the length of tags

11.15.4.93 emv_startTransaction()

```
int emv_startTransaction (
    IN double amount,
    IN double amtOther,
    IN int exponent,
    IN int type,
    IN int timeout,
    IN BYTE * tags,
    IN int tagsLen,
    IN int forceOnline )
```

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) - SEE IMPORTANT NOTE BELOW
<i>amtOther</i>	Other amount value, if any (tag value 9F03) - SEE IMPORTANT NOTE BELOW
<i>exponent</i>	Number of characters after decimal point
<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 a TLV stream. Example, tag 9F0C with amount 0x000000000100 would be 0x9F0C06000000000100 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>tagsLen</i>	Length of tags
<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 = DFEE1A079F029F36959F37

Returns

RETURN_CODE: Values can be parsed with `device_getResponseCodeString` >>>>IMPORTANT: parameters for amount and amtOther MUST BE PASSED AS A DOUBLE VALUE WITH DECIMAL POINT. Example, do not pass 1, but instead pass 1.0 or 1.00. Otherwise, results will be unpredictable

11.15.4.94 executeTransaction()

```
int executeTransaction (
    WorldPayData * data,
    pWP_callback wpCallback,
    int requestOnly )
```

executeTransaction Requests 3 sets of public keys: encrypting Keys, signing/validating keys, signing/validating 3rd party apps

Parameters

<i>data</i>	WorldPay data object
<i>wpCallback</i>	WorldPay callback
<i>requestOnly</i>	<ul style="list-style-type: none"> • 0 = send transaction and return response, • 1 = send transaction and return both request and response, • 2 = do not send transaction, instead return request

Returns

RETURN_CODE: Values can be parsed with `device_getIDGStatusCodeString()`

11.15.4.95 executeTransaction_WorldNet()

```
int executeTransaction_WorldNet (
    WorldNetData * data,
    pWN_callback wnCallback,
    int requestOnly )
```

executeTransaction_WorldNet Requests 3 sets of public keys: encrypting Keys, signing/validating keys, signing/validating 3rd party apps

Parameters

<i>data</i>	WorldNet data object
<i>wnCallback</i>	WorldNet callback
<i>requestOnly</i>	<ul style="list-style-type: none"> • 0 = send transaction and return response, • 1 = send transaction and return both request and response, • 2 = do not send transaction, instead return request

Returns

RETURN_CODE: Values can be parsed with [device_getIDGStatusCodeString\(\)](#)

11.15.4.96 forwardTransaction()

```
int forwardTransaction (
    IN pWP_callBack wpCallback,
    IN char * forwardID,
    IN int forwardIDLen,
    IN char * password,
    IN int passwordLen,
    IN int bypassProcessing )
```

forwardTransaction Send the saved data to WorldPay and complete the transaction.

Parameters

<i>wpCallback</i>	WPCallback is the callback to send the results. Should be the same as executeTransaction callback.
<i>forwardID</i>	= ID, which could be either unique ID or Memo.
<i>forwardIDLen</i>	The length of forwardID.
<i>password</i>	= password. If null/blank, no password.
<i>passwordLen</i>	The length of passwordLen.
<i>bypassProcess</i>	= true means read the file from disk, but don't send to WorldPay, then delete the transaction as if you did send to WorldPay. The purpose of this is to allow them to delete transactions from the storage without sending to WorldPay.

Returns

RETURN_CODE: Values can be parsed with [device_getIDGStatusCodeString\(\)](#)

11.15.4.97 forwardTransaction_WorldNet()

```
int forwardTransaction_WorldNet (
    IN char * apiKey,
    IN int apiKeyLen,
    IN pWN_callBack wnCallback,
    IN char * forwardID,
```

```

    IN int forwardIDLen,
    IN char * password,
    IN int passwordLen,
    IN int bypassProcessing )

```

forwardTransaction_WorldNet Send the saved data to WorldNet and complete the transaction.

Parameters

<i>wnCallback</i>	WNCallback is the callback to send the results. Should be the same as executeTransaction_WorldNet callback.
<i>forwardID</i>	= ID, which could be either unique ID or Memo.
<i>forwardIDLen</i>	The length of forwardID.
<i>password</i>	= password. If null/blank, no password.
<i>passwordLen</i>	The length of passwordLen.
<i>bypassProcess</i>	= true means read the file from disk, but don't send to WorldPay, then delete the transaction as if you did send to WorldPay. The purpose of this is to allow them to delete transactions from the storage without sending to WorldPay.

Returns

RETURN_CODE: Values can be parsed with [device_getIDGStatusCodeString\(\)](#)

11.15.4.98 `icc_exchangeAPDU()`

```

int icc_exchangeAPDU (
    IN BYTE * c_APDU,
    IN int cLen,
    OUT BYTE * reData,
    IN_OUT int * reLen )

```

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>cLen</i>	APDU data packet length
<i>reData</i>	Unencrypted APDU response
<i>reLen</i>	Unencrypted APDU response data length

Returns

RETURN_CODE: Values can be parsed with [device_getIDGStatusCodeString](#)

11.15.4.99 `icc_getICReaderStatus()`

```

int icc_getICReaderStatus (
    OUT BYTE * status )

```

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
---------------	--

Returns

RETURN_CODE: Values can be parsed with [device_getIDGStatusCodeString](#)

11.15.4.100 `icc_powerOffICC()`

```
int icc_powerOffICC ( )
```

Power Off ICC

Powers down the ICC

Returns

RETURN_CODE: Values can be parsed with [device_getIDGStatusCodeString\(\)](#)

If Success, empty If Failure, ASCII encoded data of error string

11.15.4.101 `icc_powerOnICC()`

```
int icc_powerOnICC (
    OUT BYTE * ATR,
    IN_OUT int * inLen )
```

Power On ICC

Power up the currently selected microprocessor card in the ICC reader

Parameters

<i>ATR,the</i>	ATR data response when succeeded power on ICC,
<i>inLen,the</i>	length of ATR data

Returns

RETURN_CODE: Values can be parsed with [device_getIDGStatusCodeString\(\)](#)

11.15.4.102 `msr_cancelMSRSwipe()`

```
int msr_cancelMSRSwipe ( )
```

Disable MSR Swipe Cancels MSR swipe request.

Returns

RETURN_CODE: Values can be parsed with [device_getIDGStatusCodeString\(\)](#)

11.15.4.103 msr_registerCallBk()

```
void msr_registerCallBk (
    pMSR_callback pMSRf )
```

To register the msr callback function to get the MSR card data. (Pass NULL to disable the callback.)

11.15.4.104 msr_registerCallBkp()

```
void msr_registerCallBkp (
    pMSR_callbackp pMSRf )
```

To register the msr callback function to get the MSR card data pointer. (Pass NULL to disable the callback.)

11.15.4.105 msr_startMSRSwipe()

```
int msr_startMSRSwipe (
    IN int _timeout )
```

Start 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
----------------	---------------------

Returns

RETURN_CODE: Values can be parsed with [device_getIDGStatusCodeString\(\)](#) Note: if auto poll mode is on, it will return command not allowed error

11.15.4.106 parseMSRData()

```
void parseMSRData (
    IN BYTE * resData,
    IN int resLen,
    IN_OUT IDTMSRData * cardData )
```

Parser the MSR data from the buffer into IDTMSTData structure

Parameters

<i>resData</i>	MSR card data buffer
<i>resLen</i>	the length of resData
<i>cardData</i>	the parser result with IDTMSTData structure

11.15.4.107 pin_registerCallBk()

```
void pin_registerCallBk (
    pPIN_callback pPINf )
```

To register the pin callback function to get the PINPad data. (Pass NULL to disable the callback.)

11.15.4.108 registerHotplugCallBk()

```
void registerHotplugCallBk (
    pMessageHotplug pMsgHotplug )
```

To register the USB HID hot-plug callback function which implemented in the application to monitor the hotplug message from the SDK.

11.15.4.109 registerLogCallBk()

```
void registerLogCallBk (
    pSendDataLog pFSend,
    pReadDataLog pFRead )
```

To register the log callback function which implemented in the application to monitor sending/reading data between application and reader.

11.15.4.110 rs232_device_init()

```
int rs232_device_init (
    int deviceType,
    int port_number,
    int brate )
```

Initial the device by RS232

It will try to connect to the device with provided deviceType, port_number, and brate.

Parameters

<i>deviceType</i>	Device to connect to
<i>port_number</i>	Port number of the device

Port nr. | Linux | Windows

```
| 0 | ttyS0 | COM1 | | 1 | ttyS1 | COM2 | | 2 | ttyS2 | COM3 | | 3 | ttyS3 | COM4 | | 4 | ttyS4 | COM5 | | 5 | ttyS5 |
COM6 | | 6 | ttyS6 | COM7 | | 7 | ttyS7 | COM8 | | 8 | ttyS8 | COM9 | | 9 | ttyS9 | COM10 | | 10 | ttyS10 | COM11
| | 11 | ttyS11 | COM12 | | 12 | ttyS12 | COM13 | | 13 | ttyS13 | COM14 | | 14 | ttyS14 | COM15 | | 15 | ttyS15 |
COM16 | | 16 | ttyUSB0 | n.a. | | 17 | ttyUSB1 | n.a. | | 18 | ttyUSB2 | n.a. | | 19 | ttyUSB3 | n.a. | | 20 | ttyUSB4 |
n.a. | | 21 | ttyUSB5 | n.a. | | 22 | ttyAMA0 | n.a. | | 23 | ttyAMA1 | n.a. | | 24 | ttyACM0 | n.a. | | 25 | ttyACM1 | n.a.
| | 26 | rfcomm0 | n.a. | | 27 | rfcomm1 | n.a. | | 28 | ircomm0 | n.a. | | 29 | ircomm1 | n.a. | | 30 | cuau0 | n.a. | | 31
| cuau1 | n.a. | | 32 | cuau2 | n.a. | | 33 | cuau3 | n.a. | | 34 | cuaU0 | n.a. | | 35 | cuaU1 | n.a. | | 36 | cuaU2 | n.a. |
| 37 | cuaU3 | n.a. |
```

Parameters

<i>brate</i>	Bitrate of the device
--------------	-----------------------

Returns

RETURN_CODE: Values can be parsed with [device_getIDGStatusCodeString\(\)](#)

11.15.4.111 SDK_Version()

```
char* SDK_Version ( )
```

To Get SDK version

Returns

return the SDK version string

11.15.4.112 setAbsoluteLibraryPath()

```
int setAbsoluteLibraryPath (
    const char * absoluteLibraryPath )
```

Set the path to use when searching for ID TECH's libraries. If this is not set, the libraries will be searched for with the system's default procedures.

Parameters

<i>absoluteLibraryPath</i>	The absolute path to ID TECH's libraries.
----------------------------	---

Returns

RETURN_CODE: Values can be parsed with [device_getIDGStatusCodeString\(\)](#)

11.16 Source_C/libIDT_VP3300_USB.h File Reference

VP3300 USB API.

```
#include "IDTDef.h"
```

Macros

- `#define IN`
- `#define OUT`
- `#define IN_OUT`

Typedefs

- `typedef void(* pMessageHotplug) (int, int)`
- `typedef void(* pSendDataLog) (BYTE *, int)`
- `typedef void(* pReadDataLog) (BYTE *, int)`
- `typedef void(* pEMV_callback) (int, int, BYTE *, int, IDTTransactionData *, EMV_Callback *, int)`
- `typedef void(* pMSR_callback) (int, IDTMSRData)`
- `typedef void(* pMSR_callbackp) (int, IDTMSRData *)`
- `typedef void(* pPIN_callback) (int, IDTPINData *)`
- `typedef void(* pCMR_callback) (int, IDTCMRData *)`
- `typedef void(* pCSFS_callback) (BYTE status)`
- `typedef void(* pRKI_callback) (int, char *)`
- `typedef void(* ftpComm_callback) (int, int, int)`

- typedef void(* [httpComm_callback](#)) (BYTE *, int)
- typedef void(* [v4Comm_callback](#)) (BYTE, BYTE, BYTE *, int)
- typedef void(* [pWP_callback](#)) (char *, int, int)
- typedef void(* [pWN_callback](#)) (char *, int, int)

Functions

- void [registerHotplugCallBk](#) (pMessageHotplug pMsgHotplug)
- void [registerLogCallBk](#) (pSendDataLog pFSend, [pReadDataLog](#) pFRead)
- void [device_registerRKICallBk](#) (pRKI_callback pRKIf)
- void [emv_registerCallBk](#) (pEMV_callback pEMVf)
- void [msr_registerCallBk](#) (pMSR_callback pMSRf)
- void [msr_registerCallBkp](#) (pMSR_callback pMSRf)
- void [ctls_registerCallBk](#) (pMSR_callback pCTLSf)
- void [ctls_registerCallBkp](#) (pMSR_callback pCTLSf)
- void [pin_registerCallBk](#) (pPIN_callback pPINf)
- void [device_registerCameraCallBk](#) (pCMR_callback pCMRf)
- void [device_registerCardStatusFrontSwitchCallBk](#) (pCSFS_callback pCSFSf)
- void [comm_registerHTTPCallback](#) ([httpComm_callback](#) cBack)
- void [comm_registerV4Callback](#) ([v4Comm_callback](#) cBack)
- char * [SDK_Version](#) ()
- int [setAbsoluteLibraryPath](#) (const char *absoluteLibraryPath)
- int [device_init](#) ()
- int [device_setCurrentDevice](#) (int deviceType)
- int [device_isAttached](#) (int deviceType)
- int [device_close](#) ()
- void [device_getIDGStatusCodeString](#) (IN int returnCode, OUT char *despcriton)
- int [device_isConnected](#) ()
- int [device_getFirmwareVersion](#) (OUT char *firmwareVersion)
- int [device_getFirmwareVersion_Len](#) (OUT char *firmwareVersion, IN_OUT int *firmwareVersionLen)
- int [device_pingDevice](#) ()
- int [device_controlUserInterface](#) (IN BYTE *values)
- int [device_getCurrentDeviceType](#) ()
- int [device_SendDataCommandNEO](#) (IN int cmd, IN int subCmd, IN BYTE *data, IN int dataLen, OUT BYTE *response, IN_OUT int *respLen)
- int [device_enablePassThrough](#) (int enablePassThrough)
- int [device_setBurstMode](#) (IN BYTE mode)
- int [device_setPollMode](#) (IN BYTE mode)
- int [device_setMerchantRecord](#) (int index, int enabled, char *merchantID, char *merchantURL)
- int [device_pollForToken](#) (IN int timeout, OUT BYTE *respData, IN_OUT int *respDataLen)
- int [device_getMerchantRecord](#) (IN int index, OUT BYTE *record)
- int [device_getMerchantRecord_Len](#) (IN int index, OUT BYTE *record, IN_OUT int *recordLen)
- int [device_getTransactionResults](#) (IDTMSRData *cardData)
- int [device_startTransaction](#) (IN double amount, IN double amtOther, IN int type, IN const int _timeout, IN BYTE *tags, IN int tagsLen)
- void [device_setTransactionExponent](#) (int exponent)
- int [device_activateTransaction](#) (IN const int _timeout, IN BYTE *tags, IN int tagsLen)
- int [device_cancelTransaction](#) ()
- int [device_getRTCDateTime](#) (IN BYTE *dateTime, IN_OUT int *dateTimeLen)
- int [device_setRTCDateTime](#) (IN BYTE *dateTime, IN int dateTimeLen)
- int [device_startRKI](#) (IN const char *caPath, IN int isProduction)
- void [device_setRKI_URL](#) (IN char *rkiURL, IN int rkiURLLen)
- int [device_getSDKWaitTime](#) ()
- void [device_setSDKWaitTime](#) (int waitTime)

- int [device_getThreadStackSize](#) ()
- void [device_setThreadStackSize](#) (int threadSize)
- int [config_getSerialNumber](#) (OUT char *sNumber)
- int [config_getSerialNumber_Len](#) (OUT char *sNumber, IN_OUT int *sNumberLen)
- int [ctls_startTransaction](#) (IN double amount, IN double amtOther, IN int type, IN const int _timeout, IN BYTE *tags, IN int tagsLen)
- int [ctls_activateTransaction](#) (IN const int _timeout, IN BYTE *tags, IN int tagsLen)
- int [ctls_cancelTransaction](#) ()
- int [ctls_retrieveApplicationData](#) (IN BYTE *AID, IN int AIDLen, OUT BYTE *tlv, IN_OUT int *tlvLen)
- int [ctls_setApplicationData](#) (IN BYTE *tlv, IN int tlvLen)
- int [ctls_removeApplicationData](#) (IN BYTE *AID, IN int AIDLen)
- int [ctls_removeAllApplicationData](#) ()
- int [ctls_retrieveAIDList](#) (OUT BYTE *AIDList, IN_OUT int *AIDListLen)
- int [ctls_retrieveTerminalData](#) (OUT BYTE *tlv, IN_OUT int *tlvLen)
- int [ctls_setTerminalData](#) (IN BYTE *tlv, IN int tlvLen)
- int [ctls_retrieveCAPK](#) (IN BYTE *capk, IN int capkLen, OUT BYTE *key, IN_OUT int *keyLen)
- int [ctls_setCAPK](#) (IN BYTE *capk, IN int capkLen)
- int [ctls_removeCAPK](#) (IN BYTE *capk, IN int capkLen)
- int [ctls_removeAllCAPK](#) ()
- int [ctls_retrieveCAPKList](#) (OUT BYTE *keys, IN_OUT int *keysLen)
- int [ctls_setConfigurationGroup](#) (IN BYTE *tlv, IN int tlvLen)
- int [ctls_getConfigurationGroup](#) (IN int group, OUT BYTE *tlv, OUT int *tlvLen)
- int [ctls_getAllConfigurationGroups](#) (OUT BYTE *tlv, IN_OUT int *tlvLen)
- int [ctls_removeConfigurationGroup](#) (int group)
- void [emv_allowFallback](#) (IN int allow)
- void [emv_setAutoAuthenticateTransaction](#) (IN int authenticate)
- void [emv_setAutoCompleteTransaction](#) (IN int complete)
- int [emv_getAutoAuthenticateTransaction](#) ()
- int [emv_getAutoCompleteTransaction](#) ()
- void [emv_setTransactionParameters](#) (IN double amount, IN double amtOther, IN int type, IN int timeout, IN BYTE *tags, IN int tagsLen)
- int [emv_startTransaction](#) (IN double amount, IN double amtOther, IN int exponent, IN int type, IN int timeout, IN BYTE *tags, IN int tagsLen, IN int forceOnline)
- int [emv_activateTransaction](#) (IN int timeout, IN BYTE *tags, IN int tagsLen, IN int forceOnline)
- int [emv_authenticateTransaction](#) (IN BYTE *updatedTLV, IN int updatedTLVLen)
- int [emv_authenticateTransactionWithTimeout](#) (IN int timeout, IN BYTE *updatedTLV, IN int updatedTLVLen)
- int [emv_completeTransaction](#) (IN int commError, IN BYTE *authCode, IN int authCodeLen, IN BYTE *iad, IN int iadLen, IN BYTE *tlvScripts, IN int tlvScriptsLen, IN BYTE *tlv, IN int tlvLen)
- int [emv_cancelTransaction](#) ()
- int [emv_retrieveApplicationData](#) (IN BYTE *AID, IN int AIDLen, OUT BYTE *tlv, IN_OUT int *tlvLen)
- int [emv_setApplicationData](#) (IN BYTE *name, IN int nameLen, IN BYTE *tlv, IN int tlvLen)
- int [emv_setApplicationDataTLV](#) (IN BYTE *tlv, IN int tlvLen)
- int [emv_removeApplicationData](#) (IN BYTE *AID, IN int AIDLen)
- int [emv_removeAllApplicationData](#) ()
- int [emv_retrieveAIDList](#) (OUT BYTE *AIDList, IN_OUT int *AIDListLen)
- int [emv_retrieveTerminalData](#) (OUT BYTE *tlv, IN_OUT int *tlvLen)
- int [emv_setTerminalData](#) (IN BYTE *tlv, IN int tlvLen)
- int [emv_setTerminalMajorConfiguration](#) (IN int configuration)
- int [emv_retrieveCAPK](#) (IN BYTE *capk, IN int capkLen, OUT BYTE *key, IN_OUT int *keyLen)
- int [emv_setCAPK](#) (IN BYTE *capk, IN int capkLen)
- int [emv_removeCAPK](#) (IN BYTE *capk, IN int capkLen)
- int [emv_removeAllCAPK](#) ()
- int [emv_retrieveCAPKList](#) (OUT BYTE *keys, IN_OUT int *keysLen)
- int [emv_retrieveCRL](#) (OUT BYTE *list, IN_OUT int *lssLen)
- int [emv_setCRL](#) (IN BYTE *list, IN int lssLen)

- int [emv_removeCRL](#) (IN BYTE *list, IN int lsLen)
- int [emv_removeAllCRL](#) ()
- int [icc_getICCRReaderStatus](#) (OUT BYTE *status)
- int [icc_powerOnICC](#) (OUT BYTE *ATR, IN_OUT int *inLen)
- int [icc_powerOffICC](#) ()
- int [icc_exchangeAPDU](#) (IN BYTE *c_APDU, IN int cLen, OUT BYTE *reData, IN_OUT int *reLen)
- int [msr_cancelMSRSwipe](#) ()
- int [msr_startMSRSwipe](#) (IN int _timeout)
- int [executeTransaction](#) (WorldPayData *data, [pWP_callback](#) wpCallback, int requestOnly)
- int [forwardTransaction](#) (IN [pWP_callback](#) wpCallback, IN char *forwardID, IN int forwardIDLen, IN char *password, IN int passwordLen, IN int bypassProcessing)
- int [cancelWorldPay](#) ()
- int [executeTransaction_WorldNet](#) (WorldNetData *data, [pWN_callback](#) wnCallback, int requestOnly)
- int [forwardTransaction_WorldNet](#) (IN char *apiKey, IN int apiKeyLen, IN [pWN_callback](#) wnCallback, IN char *forwardID, IN int forwardIDLen, IN char *password, IN int passwordLen, IN int bypassProcessing)
- int [cancelWorldNet](#) ()
- void [parseMSRData](#) (IN BYTE *resData, IN int resLen, IN_OUT IDTMSRData *cardData)

11.16.1 Detailed Description

VP3300 USB API.

VP3300 USB Global API methods.

11.16.2 Macro Definition Documentation

11.16.2.1 IN

```
#define IN
```

INPUT parameter.

11.16.2.2 IN_OUT

```
#define IN_OUT
```

INPUT / OUTPUT PARAMETER.

11.16.2.3 OUT

```
#define OUT
```

OUTPUT parameter.

11.16.3 Typedef Documentation

11.16.3.1 ftpComm_callBack

```
typedef void(* ftpComm_callBack) (int, int, int)
```

Define the comm callback function to get FTP file transfer status

It should be passed as a parameter in a FTP request, Signature (int, int, int) = response code, current block, total blocks
 RESPONSE CODES: 100 = FILE DOWNLOAD STARTED 101 = FILE BLOCK XX OF XX RECEIVED 102 = FILE DOWNLOAD COMPLETED 103 = FILE DOWNLOAD TERMINATED PREMATURELY

11.16.3.2 httpComm_callBack

```
typedef void(* httpComm_callBack) (BYTE *, int)
```

Define the comm callback function to get the async url data

It should be registered using the comm_registerHTTPCallback

11.16.3.3 pCMR_callBack

```
typedef void(* pCMR_callBack) (int, IDTCMRData *)
```

Define the camera callback function to get the image data

It should be registered using the device_registerCameraCallBk,

11.16.3.4 pCSFS_callBack

```
typedef void(* pCSFS_callBack) (BYTE status)
```

Define the card status and front switch callback function to get card and front switch status

It should be registered using the device_registerCardStatusFrontSwitchCallBk,

11.16.3.5 pEMV_callBack

```
typedef void(* pEMV_callBack) (int, int, BYTE *, int, IDTTransactionData *, EMV_Callback *, int)
```

Define the EMV callback function to get the transaction message/data/result.

It should be registered using the emv_registerCallBk,

11.16.3.6 pMessageHotplug

```
typedef void(* pMessageHotplug) (int, int)
```

Define the USB hot-plug callback function to monitor the info when plug in/out the reader.

It should be registered using the registerHotplugCallBk, The first integer parameter is device type, and the second integer parameter is either 0: Device Plugged Out or 1: Device Plugged In

11.16.3.7 pMSR_callBack

```
typedef void(* pMSR_callBack) (int, IDTMSRData)
```

Define the MSR callback function to get the MSR card data

It should be registered using the msr_registerCallBk, this callback function is for backward compatibility

11.16.3.8 pMSR_callBackp

```
typedef void(* pMSR_callBackp) (int, IDTMSRData *)
```

Define the MSR callback function to get pointer to the MSR card data

It should be registered using the `msr_registerCallBk`, this callback function is recommended instead of `pMSR_callBack`

11.16.3.9 pPIN_callBack

```
typedef void(* pPIN_callBack) (int, IDTPINData *)
```

Define the PINPad callback function to get the input PIN Pad data

It should be registered using the `pin_registerCallBk`,

11.16.3.10 pReadDataLog

```
typedef void(* pReadDataLog) (BYTE *, int)
```

Define the read response callback function to monitor the reading response from the reader.

It should be registered using the `registerLogCallBk`,

11.16.3.11 pRKI_callBack

```
typedef void(* pRKI_callBack) (int, char *)
```

Define the RKI callback function to get the status of the RKI

It should be registered using the `device_registerRKICallBk`,

11.16.3.12 pSendDataLog

```
typedef void(* pSendDataLog) (BYTE *, int)
```

Define the send command callback function to monitor the sending command into the reader.

It should be registered using the `registerLogCallBk`,

11.16.3.13 pWN_callBack

```
typedef void(* pWN_callBack) (char *, int, int)
```

Define the Worldnet callback function to get the transaction message/data/result.

11.16.3.14 pWP_callBack

```
typedef void(* pWP_callBack) (char *, int, int)
```

Define the Worldpay callback function to get the transaction message/data/result.

11.16.3.15 v4Comm_callBack

```
typedef void(* v4Comm_callBack) (BYTE, BYTE, BYTE *, int)
```

Define the comm callback function to receive the V4 Protocol packets received by the device from an external source (IP/USB/RS-232) It should be registered using the `comm_registerV4Callback`, Data callback will contain command, sub-command, and data from V4 packet

11.16.4 Function Documentation

11.16.4.1 `cancelWorldNet()`

```
int cancelWorldNet ( )
```

Cancels WorldNet transaction.

Returns

RETURN_CODE: Values can be parsed with [device_getIDGStatusCodeString\(\)](#)

11.16.4.2 `cancelWorldPay()`

```
int cancelWorldPay ( )
```

Cancels WorldPay transaction.

Returns

RETURN_CODE: Values can be parsed with [device_getIDGStatusCodeString\(\)](#)

11.16.4.3 `comm_registerHTTPCallback()`

```
void comm_registerHTTPCallback (
    httpComm_callBack cBack )
```

Register Comm HTTP Async Callback

Parameters

<i>cBack</i>	- HTTP Comm callback
--------------	----------------------

11.16.4.4 `comm_registerV4Callback()`

```
void comm_registerV4Callback (
    v4Comm_callBack cBack )
```

Register External V4 Protocol commands Callback

Parameters

<i>cBack</i>	- V4 Protocol Comm callback
--------------	-----------------------------

11.16.4.5 config_getSerialNumber()

```
int config_getSerialNumber (
    OUT char * sNumber )
```

DEPRECATED : please use [config_getSerialNumber_Len\(OUT char* sNumber, IN_OUT int *sNumberLen\)](#)

Polls device for Serial Number

Parameters

<i>sNumber</i>	Returns Serial Number; needs to have at least 64 bytes of memory
----------------	--

Returns

RETURN_CODE: Values can be parsed with [device_getIDGStatusCodeString](#)

11.16.4.6 config_getSerialNumber_Len()

```
int config_getSerialNumber_Len (
    OUT char * sNumber,
    IN_OUT int * sNumberLen )
```

Polls device for Serial Number

Parameters

<i>sNumber</i>	Returns Serial Number
<i>sNumberLen</i>	Length of Serial Number

Returns

RETURN_CODE: Values can be parsed with [device_getIDGStatusCodeString](#)

11.16.4.7 ctls_activateTransaction()

```
int ctls_activateTransaction (
    IN const int _timeout,
    IN BYTE * tags,
    IN int tagsLen )
```

Start CTLS Transaction Request

Authorizes the CTLS 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 stream. Example, tag 9F0C with amount 0x000000000100 would be 0x9F0C06000000000100
<i>tagsLen</i>	The length of tags data buffer.

>>>>>IMPORTANT: parameters for amount and amtOther MUST BE PASSED AS A DOUBLE VALUE WITH DECIMAL POINT. Example, do not pass 1, but instead pass 1.0 or 1.00. Otherwise, results will be unpredictable

Returns

RETURN_CODE: Values can be parsed with [device_getIDGStatusCodeString\(\)](#) Note: if auto poll is on, it will return the error IDG_P2_STATUS_CODE_COMMAND_NOT_ALLOWED

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) DFO1 = 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

11.16.4.8 ctls_cancelTransaction()

```
int ctls_cancelTransaction ( )
```

Cancel EMV Transaction

Cancels the currently executing EMV transaction.

Returns

RETURN_CODE: Values can be parsed with [device_getIDGStatusCodeString\(\)](#)

11.16.4.9 ctls_getAllConfigurationGroups()

```
int ctls_getAllConfigurationGroups (
    OUT BYTE * tlv,
    IN_OUT int * tlvLen )
```

Retrieve All Configuration Groups

Returns all the Configuration Groups installed on the terminal for CTLS

Parameters

<i>tlv</i>	The TLV elements data
<i>tlvLen</i>	the length of tlv data buffer.

Returns

RETURN_CODE: Values can be parsed with [device_getIDGStatusCodeString\(\)](#)

11.16.4.10 ctls_getConfigurationGroup()

```
int ctls_getConfigurationGroup (
    IN int group,
    OUT BYTE * tlv,
    OUT int * tlvLen )
```

Get Configuration Group

Retrieves the Configuration for the specified Group.

Parameters

<i>group</i>	Configuration Group
<i>tlv</i>	return data
<i>tlvLen</i>	the length of tlv data buffer

Returns

RETURN_CODE: Values can be parsed with [device_getIDGStatusCodeString\(\)](#)

11.16.4.11 ctls_registerCallBk()

```
void ctls_registerCallBk (
    pMSR_callback pCTLSf )
```

To register the ctls callback function to get the MSR card data. (Pass NULL to disable the callback.)

11.16.4.12 ctls_registerCallBkp()

```
void ctls_registerCallBkp (
    pMSR_callback pCTLSf )
```

To register the ctls callback function to get the MSR card data pointer. (Pass NULL to disable the callback.)

11.16.4.13 ctls_removeAllApplicationData()

```
int ctls_removeAllApplicationData ( )
```

Remove All Application Data

Removes all the Application Data

Returns

RETURN_CODE: Values can be parsed with [device_getIDGStatusCodeString\(\)](#)

11.16.4.14 ctls_removeAllCAPK()

```
int ctls_removeAllCAPK ( )
```

Remove All Certificate Authority Public Key

Removes all the CAPK

Returns

RETURN_CODE: Values can be parsed with [device_getIDGStatusCodeString\(\)](#)

11.16.4.15 ctls_removeApplicationData()

```
int ctls_removeApplicationData (
    IN BYTE * AID,
    IN int AIDLLen )
```

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>AIDLLen</i>	the length of AID data buffer

Returns

RETURN_CODE: Values can be parsed with [device_getIDGStatusCodeString\(\)](#)

11.16.4.16 ctls_removeCAPK()

```
int ctls_removeCAPK (
    IN BYTE * capk,
    IN int capkLen )
```

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>capkLen</i>	the length of capk data buffer

Returns

RETURN_CODE: Values can be parsed with [device_getIDGStatusCodeString\(\)](#)

11.16.4.17 ctls_removeConfigurationGroup()

```
int ctls_removeConfigurationGroup (
    int group )
```

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>	Values can be parsed with device_getIDGStatusCodeString()
--------------------	---

11.16.4.18 ctls_retrieveAIDList()

```
int ctls_retrieveAIDList (
    OUT BYTE * AIDList,
    IN_OUT int * AIDListLen )
```

Retrieve AID list

Returns all the AID names installed on the terminal for CTLS. .

Parameters

<i>AIDList</i>	array of AID name byte arrays
<i>AIDListLen</i>	the length of AIDList array buffer

Returns

RETURN_CODE: Values can be parsed with [device_getIDGStatusCodeString\(\)](#)

11.16.4.19 ctls_retrieveApplicationData()

```
int ctls_retrieveApplicationData (
    IN BYTE * AID,
    IN int AIDLen,
    OUT BYTE * tlv,
    IN_OUT int * tlvLen )
```

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>AIDLen</i>	the length of AID data buffer.
<i>tlv</i>	The TLV elements of the requested AID
<i>tlvLen</i>	the length of tlv data buffer.

Returns

RETURN_CODE: Values can be parsed with [device_getIDGStatusCodeString\(\)](#)

11.16.4.20 ctls_retrieveCAPK()

```
int ctls_retrieveCAPK (
    IN BYTE * capk,
    IN int capkLen,
    OUT BYTE * key,
    IN_OUT int * keyLen )
```

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>capkLen</i>	the length of capk data buffer

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>keyLen</i>	<p>the length of key data buffer</p> <ul style="list-style-type: none"> •

Returns

RETURN_CODE: Values can be parsed with [device_getIDGStatusCodeString\(\)](#)

11.16.4.21 `ctls_retrieveCAPKList()`

```
int ctls_retrieveCAPKList (
    OUT BYTE * keys,
    IN_OUT int * keyLen )
```

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>keyLen</i>	the length of keys data buffer

Returns

RETURN_CODE: Values can be parsed with [device_getIDGStatusCodeString\(\)](#)

11.16.4.22 `ctls_retrieveTerminalData()`

```
int ctls_retrieveTerminalData (
    OUT BYTE * tlv,
    IN_OUT int * tlvLen )
```

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>tlvLen</i>	the length of tlv data buffer

Returns

RETURN_CODE: Values can be parsed with [device_getIDGStatusCodeString\(\)](#)

11.16.4.23 `ctls_setApplicationData()`

```
int ctls_setApplicationData (  
    IN BYTE * tlv,  
    IN int tlvLen )
```

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>tlvLen</i>	the length of tlv data buffer
---------------	-------------------------------

Returns

RETURN_CODE: Values can be parsed with [device_getIDGStatusCodeString\(\)](#)

11.16.4.24 `ctls_setCAPK()`

```
int ctls_setCAPK (  
    IN BYTE * capk,  
    IN int capkLen )
```

Set Certificate Authority Public Key

Sets the CAPK as specified by the CAKey structure

Parameters

<i>capk</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>capkLen</i>	the length of capk data buffer

Returns

RETURN_CODE: Values can be parsed with [device_getIDGStatusCodeString\(\)](#)

11.16.4.25 ctls_setConfigurationGroup()

```
int ctls_setConfigurationGroup (
    IN BYTE * tlv,
    IN int tlvLen )
```

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>tlvLen</i>	the length of tlv data buffer

Returns

RETURN_CODE: Values can be parsed with [device_getIDGStatusCodeString\(\)](#)

11.16.4.26 ctls_setTerminalData()

```
int ctls_setTerminalData (
    IN BYTE * tlv,
    IN int tlvLen )
```

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 `ctls_getConfigurationGroup(int group)`, and deleted with `ctls_removeConfigurationGroup(int group)`. You cannot delete group 0.

Parameters

<i>tlv</i>	TerminalData configuration file
<i>tlvLen</i>	the length of tlv data buffer

Return values

RETURN_CODE	Values can be parsed with <code>device_getIDGStatusCodeString()</code>
--------------------	--

11.16.4.27 ctls_startTransaction()

```
int ctls_startTransaction (
    IN double amount,
    IN double amtOther,
    IN int type,
    IN const int _timeout,
    IN BYTE * tags,
    IN int tagsLen )
```

Start CTLS Transaction Request

Authorizes the CTLS transaction for an ICC card

The tags will be returned in the callback routine.

Parameters

<i>amount</i>	Transaction amount value (tag value 9F02) - SEE IMPORTANT NOTE BELOW
<i>amtOther</i>	Other amount value, if any (tag value 9F03) - SEE IMPORTANT NOTE BELOW
<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 stream. Example, tag 9F0C with amount 0x000000000100 would be 0x9F0C06000000000100 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>tagsLen</i>	The length of tags data buffer.

>>>>>IMPORTANT: parameters for amount and amtOther MUST BE PASSED AS A DOUBLE VALUE WITH DECIMAL POINT. Example, do not pass 1, but instead pass 1.0 or 1.00. Otherwise, results will be unpredictable

Returns

RETURN_CODE: Values can be parsed with `device_getIDGStatusCodeString()` Note: if auto poll is on, it will return the error IDG_P2_STATUS_CODE_COMMAND_NOT_ALLOWED

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

11.16.4.28 device_activateTransaction()

```
int device_activateTransaction (
    IN const int _timeout,
    IN BYTE * tags,
    IN int tagsLen )
```

Start Transaction Request

Authorizes the transaction for an MSR/CTLS/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. Example, tag 9F0C with amount 0x000000000100 would be 0x9F0C06000000000100 Be sure to include 9F02 (amount) and 9C (transaction type).
<i>tagsLen</i>	The length of tags data buffer.

>>>>>IMPORTANT: parameters for amount and amtOther MUST BE PASSED AS A DOUBLE VALUE WITH DECIMAL POINT. Example, do not pass 1, but instead pass 1.0 or 1.00. Otherwise, results will be unpredictable

Returns

RETURN_CODE: Values can be parsed with [device_getIDGStatusCodeString\(\)](#) Note: if auto poll is on, it will return the error IDG_P2_STATUS_CODE_COMMAND_NOT_ALLOWED

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 `device_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

11.16.4.29 device_cancelTransaction()

```
int device_cancelTransaction ( )
```

Disable Transaction Cancel Transaction request.

Returns

RETURN_CODE: Values can be parsed with [device_getIDGStatusCodeString\(\)](#)

11.16.4.30 device_close()

```
int device_close ( )
```

Close the device

Returns

RETURN_CODE: 0: success, 0x0A: failed

11.16.4.31 device_controlUserInterface()

```
int device_controlUserInterface (
    IN BYTE * values )
```

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) • 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
---------------	--

Returns

RETURN_CODE: Values can be parsed with [device_getIDGStatusCodeString\(\)](#)

11.16.4.32 device_enablePassThrough()

```
int device_enablePassThrough (
    int enablePassThrough )
```

Enable Pass Through

Enables Pass Through Mode for direct communication with L1 interface (power on icc, send apdu, etc).

Parameters

<i>enablePassThrough</i>	1 = pass through ON, 0 = pass through OFF
--------------------------	---

Returns

RETURN_CODE: Values can be parsed with [device_getIDGStatusCodeString\(\)](#)

11.16.4.33 device_getCurrentDeviceType()

```
int device_getCurrentDeviceType ( )
```

Get current active device type

Returns

: return the device type defined as DEVICE_TYPE in the IDTDef.h

11.16.4.34 device_getFirmwareVersion()

```
int device_getFirmwareVersion (
    OUT char * firmwareVersion )
```

DEPRECATED : please use [device_getFirmwareVersion_Len\(\)](#)(OUT char* firmwareVersion, IN_OUT int *firmwareVersionLen)

Polls device for Firmware Version

Parameters

<i>firmwareVersion</i>	Response returned of Firmware Version; needs to have at least 128 bytes of memory
------------------------	---

Returns

RETURN_CODE: Values can be parsed with [device_getIDGStatusCodeString\(\)](#)

11.16.4.35 device_getFirmwareVersion_Len()

```
int device_getFirmwareVersion_Len (
    OUT char * firmwareVersion,
    IN_OUT int * firmwareVersionLen )
```

Polls device for Firmware Version

Parameters

<i>firmwareVersion</i>	Response returned of Firmware Version
<i>firmwareVersionLen</i>	Length of Firmware Version

Returns

RETURN_CODE: Values can be parsed with [device_getIDGStatusCodeString\(\)](#)

11.16.4.36 device_getIDGStatusCodeString()

```
void device_getIDGStatusCodeString (
    IN int  returnCode,
    OUT char * despcrition )
```

Review the return code description.

Parameters

<i>returnCode</i>	the response result.
<i>description</i>	

Return values

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

- 0: "no error, beginning task";
- 1: "no response from reader";
- 2: "invalid response data";
- 01: " Incorrect Header Tag";
- 02: " Unknown Command";
- 03: " Unknown Sub-Command";
- 04: " CRC Error in Frame";
- 05: " Incorrect Parameter";
- 06: " Parameter Not Supported";
- 07: " Mal-formatted Data";
- 08: " Timeout";
- 0A: " Failed / NACK";
- 0B: " Command not Allowed";
- 0C: " Sub-Command not Allowed";
- 0D: " Buffer Overflow (Data Length too large for reader buffer)";

- 0E: " User Interface Event";
- 10: " Need clear firmware(apply in boot loader only)";
- 11: " Communication type not supported, VT-1, burst, etc. Need encrypted firmware (apply in boot loader only)";
- 12: " Secure interface is not functional or is in an intermediate state.";
- 13: " Data field is not mod 8";
- 14: " Pad 0x80 not found where expected";
- 15: " Specified key type is invalid";
- 16: " Could not retrieve key from the SAM (InitSecureComm)";
- 17: " Hash code problem";
- 18: " Could not store the key into the SAM (InstallKey)";
- 19: " Frame is too large";
- 1A: " Unit powered up in authentication state but POS must resend the InitSecureComm command";
- 1B: " The EEPROM may not be initialized because SecCommInterface does not make sense";
- 1C: " Problem encoding APDU Module-Specific Status Codes ";
- 20: " Unsupported Index (ILM) SAM Transceiver error - problem communicating with the SAM (Key Mgr)";
- 21: " Unexpected Sequence Counter in multiple frames for single bitmap (ILM)Length error in data returned from the SAM (Key Mgr) - 22: " Improper bit map (ILM)"; - 23: " Request Online Authorization"; - 24: " ViV \leftrightarrow OCard3 raw data read successful"; - 25: " Message index not available (ILM) ViVOcomm activate transaction card type (ViVOcomm)"; - 26: " Version Information Mismatch (ILM)"; - 27: " Not sending commands in correct index message index (ILM)"; - 28: " Time out or next expected message not received (ILM)"; - 29: " ILM languages not available for viewing (ILM)"; - 2A: " Other language not supported (ILM)"; - 41: " from 41 to 4F, Module-specific errors for Key Manager";
- 50: " Auto-Switch OK";
- 51: " Auto-Switch failed";
- 70: " Antenna Error 80h Use another card";
- 81: " Insert or swipe card";
- 90: " Data encryption Key does not exist";
- 91: " Data encryption Key KSN exhausted";

11.16.4.37 device_getMerchantRecord()

```
int device_getMerchantRecord (
    IN int index,
    OUT BYTE * record )
```

DEPRECATED : please use device_getMerchantRecord_Len(IN int index, OUT BYTE * record, IN_OUT int *recordLen)

Get Merchant Record

Gets the merchant record for the device.

Parameters

<i>index</i>	Merchant Record index, valid values 1-6
<i>record;</i>	needs to have at least 99 bytes of memory response data from reader. Merchant Record Index: 1 byte enabled: 1 byte Merchant ID: 32 bytes Length of Merchant URL: 1 byte Merchant URL: 64 bytes

Returns

success or error code. Values can be parsed with [device_getIDGStatusCodeString\(\)](#)

See also

[ErrorCode](#)

11.16.4.38 [device_getMerchantRecord_Len\(\)](#)

```
int device_getMerchantRecord_Len (
    IN int index,
    OUT BYTE * record,
    IN_OUT int * recordLen )
```

Get Merchant Record

Gets the merchant record for the device.

Parameters

<i>index</i>	Merchant Record index, valid values 1-6
<i>record</i>	response data from reader. Merchant Record Index: 1 byte enabled: 1 byte Merchant ID: 32 bytes Length of Merchant URL: 1 byte Merchant URL: 64 bytes
<i>recordLen</i>	Length of record

Returns

success or error code. Values can be parsed with [device_getIDGStatusCodeString\(\)](#)

See also

[ErrorCode](#)

11.16.4.39 [device_getRTCDateTime\(\)](#)

```
int device_getRTCDateTime (
    IN BYTE * dateTime,
    IN_OUT int * dateTimeLen )
```

get RTC date and time of the device

Parameters

<i>dateTime</i>	<dateTime data>="" is: 6 byte data for YYMMDDHHMMSS in hex. For example 0x171003102547 stands for 2017 Oct 3rd 10:25:47
<i>dateTimeLen</i>	return 6 bytes if successful

Returns

success or error code. Values can be parsed with device_getResponseCodeString

See also

ErrorCode

11.16.4.40 device_getSDKWaitTime()

```
int device_getSDKWaitTime ( )
```

Get SDK Wait Time

Get the SDK wait time for transactions

Returns

SDK wait time in seconds

11.16.4.41 device_getThreadStackSize()

```
int device_getThreadStackSize ( )
```

Get Thread Stack Size

Get the stack size setting for newly created threads

Returns

Thread Stack Size

11.16.4.42 device_getTransactionResults()

```
int device_getTransactionResults (
    IDTMSRData * cardData )
```

Get Transaction Results Gets the transaction results when the reader is functioning in "Auto Poll" mode

Parameters

<i>cardData</i>	The transaction results
-----------------	-------------------------

Returns

success or error code. Values can be parsed with device_getResponseCodeString

See also

ErrorCode

11.16.4.43 device_init()

```
int device_init ( )
```

Initial the device by USB

It will detect the device and trying connect.

The connect status can be checked by [device_isConnected\(\)](#).

Note: after the function returns success, the function [device_setCurrentDevice\(\)](#) has to be called to set the device type.

Returns

RETURN_CODE: Values can be parsed with [device_getIDGStatusCodeString\(\)](#)

11.16.4.44 device_isAttached()

```
int device_isAttached (
    int deviceType )
```

Check if the device is attached to the USB port The function [device_init\(\)](#) must be called before this function.

Parameters

<i>deviceType,the</i>	device type of the USB device
-----------------------	-------------------------------

Returns

1 if the device is attached, or 0 if the device is not attached

11.16.4.45 device_isConnected()

```
int device_isConnected ( )
```

Check the device connected status

Returns

DEVICE_DISCONNECT=0, or DEVICE_CONNECTED = 1

11.16.4.46 device_pingDevice()

```
int device_pingDevice ( )
```

Ping Device

Pings the reader. If connected, returns success. Otherwise, returns timeout.

Returns

RETURN_CODE: Values can be parsed with [device_getIDGStatusCodeString\(\)](#)

11.16.4.47 device_pollForToken()

```
int device_pollForToken (
    IN int timeout,
    OUT BYTE * respData,
    IN_OUT int * respDataLen )
```

Poll for Token

Polls for a PICC

Parameters

<i>timeout</i>	timeout in milliseconds, must be multiple of 10 milliseconds. 30, 120, 630, or 1150 for example.
<i>respData</i>	Response data will be stored in respData. 1 byte of card type, and the Serial Number (or the UID) of the PICC if available.
<i>respDataLen</i>	Length of systemCode.

Returns

RETURN_CODE: Values can be parsed with [device_getIDGStatusCodeString\(\)](#)

11.16.4.48 device_registerCameraCallBk()

```
void device_registerCameraCallBk (
    pCMR_callBack pCMRf )
```

To register the camera callback function to get the image data. (Pass NULL to disable the callback.)

11.16.4.49 device_registerCardStatusFrontSwitchCallBk()

```
void device_registerCardStatusFrontSwitchCallBk (
    pCSFS_callBack pCSFSf )
```

To register the card status and front switch callback function to get status. (Pass NULL to disable the callback.)

11.16.4.50 device_registerRKICallBk()

```
void device_registerRKICallBk (
    pRKI_callBack pRKIf )
```

To register the RKI callback function to get the RKI status. (Pass NULL to disable the callback.)

11.16.4.51 device_SendDataCommandNEO()

```
int device_SendDataCommandNEO (
    IN int cmd,
    IN int subCmd,
    IN BYTE * data,
    IN int dataLen,
    OUT BYTE * response,
    IN_OUT int * respLen )
```

Send a Command to device

Sends a command to the device .

Parameters

<i>cmd</i>	buffer of command to execute.
<i>cmdLen,the</i>	length of the buffer cmd.
<i>data</i>	buffer of IDG command data.
<i>dataLen,the</i>	length of the buffer data.
<i>response</i>	Response data
<i>respLen,the</i>	length of Response data

Returns

RETURN_CODE: Values can be parsed with [device_getIDGStatusCodeString\(\)](#) Send a Command to NEO device

Sends a command to the NEO device .

Parameters

<i>cmd</i>	command to execute.
<i>subCmd,sub</i>	command to execute.
<i>data</i>	buffer of NEO command data.
<i>dataLen,the</i>	length of the buffer data.
<i>response</i>	Response data
<i>respLen,the</i>	length of Response data

Returns

RETURN_CODE: Values can be parsed with [device_getIDGStatusCodeString\(\)](#)

11.16.4.52 device_setBurstMode()

```
int device_setBurstMode (
    IN BYTE mode )
```

Send Burst Mode

Sets the burst mode for the device.

Parameters

<i>mode</i>	0 = OFF, 1 = Always On, 2 = Auto Exit
-------------	---------------------------------------

Returns

success or error code. Values can be parsed with [device_getIDGStatusCodeString](#)

See also

ErrorCode

11.16.4.53 device_setCurrentDevice()

```
int device_setCurrentDevice (
    int deviceType )
```

Sets the current device to talk to

The connect status can be checked by [device_isConnected\(\)](#).

Parameters

<i>deviceType</i>	<p>Device to connect to</p> <pre>enum DEVICE_TYPE { IDT_DEVICE_UNKNOWN=0, IDT_DEVICE_AUGUSTA_HID, IDT_DEVICE_AUGUSTA_KB, IDT_DEVICE_AUGUSTA_S_HID, IDT_DEVICE_AUGUSTA_S_KB, IDT_DEVICE_AUGUSTA_S_TTK_HID, IDT_DEVICE_SPECTRUM_PRO, IDT_DEVICE_MINISMART_II, IDT_DEVICE_L80, IDT_DEVICE_L100, IDT_DEVICE_UNIPAY, IDT_DEVICE_UNIPAY_I_V, IDT_DEVICE_VP3300_AJ, IDT_DEVICE_KIOSK_III, IDT_DEVICE_KIOSK_III_S, IDT_DEVICE_PIP_READER, IDT_DEVICE_VENDI, IDT_DEVICE_VP3300_USB, IDT_DEVICE_UNIPAY_I_V_TTK, IDT_DEVICE_VP3300_BT, IDT_DEVICE_VP8800, IDT_DEVICE_SREDKEY2_HID, IDT_DEVICE_SREDKEY2_KB, IDT_DEVICE_NEO2, IDT_DEVICE_MINISMART_II_COM = IDT_DEVICE_NEO2+5, IDT_DEVICE_SPECTRUM_PRO_COM, IDT_DEVICE_KIOSK_III_COM, IDT_DEVICE_KIOSK_III_S_COM, IDT_DEVICE_PIP_READER_COM, IDT_DEVICE_VP3300_COM, IDT_DEVICE_NEO2_COM, IDT_DEVICE_MAX_DEVICES = IDT_DEVICE_NEO2_COM+5 };</pre>
-------------------	---

Returns

RETURN_CODE: 1: success, 0: failed

11.16.4.54 device_setMerchantRecord()

```
int device_setMerchantRecord (
    int index,
    int enabled,
    char * merchantID,
    char * merchantURL )
```

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

Returns

RETURN_CODE: Values can be parsed with [device_getIDGStatusCodeString\(\)](#)

11.16.4.55 device_setPollMode()

```
int device_setPollMode (
    IN BYTE mode )
```

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>	0 = Auto Poll, 1 = Poll On Demand
-------------	-----------------------------------

Returns

RETURN_CODE: Values can be parsed with [device_getIDGStatusCodeString\(\)](#)

11.16.4.56 device_setRKI_URL()

```
void device_setRKI_URL (
    IN char * rkiURL,
    IN int rkiURLLen )
```

Set the URL for RKI

Parameters

<i>rkiURL</i>	The URL for RKI (less than 100 characters). Pass NULL to reset to default URL
<i>rkiURLLen</i>	The length of rkiURL. Pass 0 to reset to default URL

Returns

11.16.4.57 device_setRTCDateTime()

```
int device_setRTCDateTime (
    IN BYTE * dateTime,
    IN int dateTimeLen )
```

set RTC date and time of the device

Parameters

<i>dateTime</i>	<dateTime data>=""> is: 6 byte data for YYMMDDHHMMSS in hex. For example 0x171003102547 stands for 2017 Oct 3rd 10:25:47
<i>dateTimeLen</i>	should be always 6 bytes

Returns

success or error code. Values can be parsed with [device_getResponseCodeString](#)

See also

[ErrorCode](#)

11.16.4.58 device_setSDKWaitTime()

```
void device_setSDKWaitTime (
    int waitTime )
```

Set SDK Wait Time

Set the SDK wait time for transactions

Parameters

<i>waitTime</i>	The SDK wait time for transaction in seconds. The Maximum is 2147483 seconds.
-----------------	---

11.16.4.59 device_setThreadStackSize()

```
void device_setThreadStackSize (
    int threadSize )
```

Set Thread Stack Size

Set the stack size setting for newly created threads

11.16.4.60 device_setTransactionExponent()

```
void device_setTransactionExponent (
    int exponent )
```

Sets the transaction exponent to be used with device_startTransaction. Default value is 2

Parameters

<i>exponent, The</i>	exponent to use when calling device_startTransaction
----------------------	--

11.16.4.61 device_startRKI()

```
int device_startRKI (
    IN const char * caPath,
    IN int isProduction )
```

Start remote key injection.

Parameters

<i>caPath</i>	The path to ca-certificates.crt It should be NULL, because the file is not used anymore.
<i>isProduction</i>	1: The reader is a production unit, 0: The reader is not a production unit

Returns

success or error code.

See also

ErrorCode

11.16.4.62 device_startTransaction()

```
int device_startTransaction (
    IN double amount,
    IN double amtOther,
    IN int type,
    IN const int _timeout,
    IN BYTE * tags,
    IN int tagsLen )
```

Start device Transaction Request

Authorizes the transaction for an MSR/CTLS/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>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. Example, tag 9F0C with amount 0x000000000100 would be 0x9F0C06000000000100 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 = DFEE1A079F029F36959F37

Returns

success or error code. Values can be parsed with device_getResponseCodeString

See also

ErrorCode Note: if auto poll is on, it will return the error IDG_P2_STATUS_CODE_COMMAND_NOT_ALLOWED

11.16.4.63 emv_activateTransaction()

```
int emv_activateTransaction (
    IN int timeout,
    IN BYTE * tags,
    IN int tagsLen,
    IN int forceOnline )
```

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>	Tags to be included in the request. Passed as a TLV stream. Example, tag 9F0C with amount 0x000000000100 would be 0x9F0C06000000000100
<i>tagsLen</i>	Length of tags
<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 = DFEE1A079F029F36959F37

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString >>>>IMPORTANT: parameters for amount and amtOther MUST BE PASSED AS A DOUBLE VALUE WITH DECIMAL POINT. Example, do not pass 1, but instead pass 1.0 or 1.00. Otherwise, results will be unpredictable

11.16.4.64 emv_allowFallback()

```
void emv_allowFallback (
    IN int allow )
```

Allow fallback for EMV transactions. Default is TRUE

Parameters

<i>allow</i>	TRUE = allow fallback, FALSE = don't allow fallback
--------------	---

11.16.4.65 emv_authenticateTransaction()

```
int emv_authenticateTransaction (
    IN BYTE * updatedTLV,
    IN int updatedTLVLen )
```

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 = DFEE1A079F029F36959F37
-------------------	---

Parameters

<i>updatedTLVLen</i>	
----------------------	--

Returns

RETURN_CODE: Values can be parsed with `device_getResponseCodeString`

11.16.4.66 `emv_authenticateTransactionWithTimeout()`

```
int emv_authenticateTransactionWithTimeout (
    IN int timeout,
    IN BYTE * updatedTLV,
    IN int updatedTLVLen )
```

Authenticate EMV Transaction Request with Timeout

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>timeout</i>	Timeout value in seconds.
<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 = DFEE1A079F029F36959F37
<i>updatedTLVLen</i>	

Returns

RETURN_CODE: Values can be parsed with `device_getResponseCodeString`

11.16.4.67 `emv_cancelTransaction()`

```
int emv_cancelTransaction ( )
```

Cancel EMV Transaction

Cancels the currently executing EMV transaction.

Returns

RETURN_CODE: Values can be parsed with `device_getResponseCodeString()`

11.16.4.68 emv_completeTransaction()

```
int emv_completeTransaction (
    IN int commError,
    IN BYTE * authCode,
    IN int authCodeLen,
    IN BYTE * iad,
    IN int iadLen,
    IN BYTE * tlvScripts,
    IN int tlvScriptsLen,
    IN BYTE * tlv,
    IN int tlvLen )
```

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(1) if host was unreachable, or FALSE(0) 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>authCodeLen</i>	the length of authCode
<i>iad</i>	Issuer Authentication Data, if any. Example 0x11223344556677883030 (tag value 91).
<i>iadLen</i>	the length of iadLen
<i>tlvScripts</i>	71/72 scripts, if any
<i>tlvScriptsLen</i>	the length of tlvScriptsLen
<i>tlv</i>	Additional TLV data to return with transaction results (if any)
<i>tlvLen</i>	the length of tlv

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString

11.16.4.69 emv_getAutoAuthenticateTransaction()

```
int emv_getAutoAuthenticateTransaction ( )
```

Gets auto authenticate value for EMV transactions.

Returns

RETURN_CODE: TRUE = auto authenticate, FALSE = manually authenticate

11.16.4.70 emv_getAutoCompleteTransaction()

```
int emv_getAutoCompleteTransaction ( )
```

Gets auto complete value for EMV transactions.

Returns

RETURN_CODE: TRUE = auto complete, FALSE = manually complete

11.16.4.71 emv_registerCallBk()

```
void emv_registerCallBk (
    pEMV_callback pEMVf )
```

To register the emv callback function to get the EMV processing response. (Pass NULL to disable the callback.)

11.16.4.72 emv_removeAllApplicationData()

```
int emv_removeAllApplicationData ( )
```

Remove All Application Data

Removes all the Application Data

Returns

RETURN_CODE: Values can be parsed with [device_getIDGStatusCodeString\(\)](#)

11.16.4.73 emv_removeAllCAPK()

```
int emv_removeAllCAPK ( )
```

Remove All Certificate Authority Public Key

Removes all the CAPK

Returns

RETURN_CODE: Values can be parsed with [device_getIDGStatusCodeString\(\)](#)

11.16.4.74 emv_removeAllCRL()

```
int emv_removeAllCRL ( )
```

Remove All Certificate Revocation List Entries

Removes all CRLentry entries

Returns

RETURN_CODE: Values can be parsed with [device_getResponseCodeString\(\)](#)

11.16.4.75 emv_removeApplicationData()

```
int emv_removeApplicationData (
    IN BYTE * AID,
    IN int AIDLen )
```

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>AIDLen</i>	the length of AID data buffer

Returns

RETURN_CODE: Values can be parsed with [device_getIDGStatusCodeString\(\)](#)

11.16.4.76 emv_removeCAPK()

```
int emv_removeCAPK (
    IN BYTE * capk,
    IN int capkLen )
```

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>capkLen</i>	the length of capk data buffer

Returns

RETURN_CODE: Values can be parsed with [device_getIDGStatusCodeString\(\)](#)

11.16.4.77 emv_removeCRL()

```
int emv_removeCRL (
    IN BYTE * list,
    IN int lsLen )
```

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>lsLen</i>	the length of list data buffer

Returns

RETURN_CODE: Values can be parsed with [device_getResponseCodeString\(\)](#)

11.16.4.78 emv_retrieveAIDList()

```
int emv_retrieveAIDList (
    OUT BYTE * AIDList,
    IN_OUT int * AIDListLen )
```

Retrieve AID list

Returns all the AID names installed on the terminal for CTLS. .

Parameters

<i>AIDList</i>	array of AID name byte arrays
<i>AIDListLen</i>	the length of AIDList array buffer

Returns

RETURN_CODE: Values can be parsed with [device_getIDGStatusCodeString\(\)](#)

11.16.4.79 emv_retrieveApplicationData()

```
int emv_retrieveApplicationData (
    IN BYTE * AID,
    IN int AIDLen,
    OUT BYTE * tlv,
    IN_OUT int * tlvLen )
```

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>AIDLen</i>	the length of AID data buffer.
<i>tlv</i>	The TLV elements of the requested AID
<i>tlvLen</i>	the length of tlv data buffer.

Returns

RETURN_CODE: Values can be parsed with [device_getIDGStatusCodeString\(\)](#)

11.16.4.80 emv_retrieveCAPK()

```
int emv_retrieveCAPK (
    IN BYTE * capk,
    IN int capkLen,
    OUT BYTE * key,
    IN_OUT int * keyLen )
```

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>capkLen</i>	the length of capk data buffer
<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>keyLen</i>	<p>the length of key data buffer</p> <ul style="list-style-type: none"> •

Returns

RETURN_CODE: Values can be parsed with [device_getIDGStatusCodeString\(\)](#)

11.16.4.81 emv_retrieveCAPKList()

```
int emv_retrieveCAPKList (
    OUT BYTE * keys,
    IN_OUT int * keysLen )
```

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>keysLen</i>	the length of keys data buffer

Returns

RETURN_CODE: Values can be parsed with [device_getIDGStatusCodeString\(\)](#)

11.16.4.82 emv_retrieveCRL()

```
int emv_retrieveCRL (
```

```

    OUT BYTE * list,
    IN_OUT int * lssLen )

```

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>lssLen</i>	the length of list data buffer

Returns

RETURN_CODE: Values can be parsed with [device_getResponseCodeString\(\)](#)

11.16.4.83 emv_retrieveTerminalData()

```

int emv_retrieveTerminalData (
    OUT BYTE * tlv,
    IN_OUT int * tlvLen )

```

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>tlvLen</i>	the length of tlv data buffer

Returns

RETURN_CODE: Values can be parsed with [device_getIDGStatusCodeString\(\)](#)

11.16.4.84 emv_setApplicationData()

```

int emv_setApplicationData (
    IN BYTE * name,
    IN int nameLen,
    IN BYTE * tlv,
    IN int tlvLen )

```

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>nameLen</i>	the length of name data buffer of Application name,
<i>tlv</i>	Application data in TLV format

Parameters

<i>tlvLen</i>	the length of tlv data buffer
---------------	-------------------------------

Returns

RETURN_CODE: Values can be parsed with [device_getResponseCodeString\(\)](#)

11.16.4.85 emv_setApplicationDataTLV()

```
int emv_setApplicationDataTLV (
    IN BYTE * tlv,
    IN int tlvLen )
```

Set Application Data by TLV

Sets the Application Data as specified by the TLV data

Parameters

<i>tlv</i>	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) Example valid TLV, for Group #2, AID a0000000035010: "dfee2d01029f0607a0000000051010ffe10101ffe50110ffe30114ffe20106"
<i>tlvLen</i>	the length of tlv data buffer

Returns

RETURN_CODE: Values can be parsed with [device_getResponseCodeString\(\)](#)

11.16.4.86 emv_setAutoAuthenticateTransaction()

```
void emv_setAutoAuthenticateTransaction (
    IN int authenticate )
```

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

Parameters

<i>authenticate</i>	TRUE = auto authenticate, FALSE = manually authenticate
---------------------	---

11.16.4.87 emv_setAutoCompleteTransaction()

```
void emv_setAutoCompleteTransaction (
    IN int complete )
```

Enables complete for EMV transactions. If a `emv_authenticateTransaction` results in code 0x0004 (go online), then `emv_completeTransaction` can automatically execute if parameter is set to TRUE

Parameters

<i>complete</i>	TRUE = auto complete, FALSE = manually complete
-----------------	---

11.16.4.88 emv_setCAPK()

```
int emv_setCAPK (
    IN BYTE * capk,
    IN int capkLen )
```

Set Certificate Authority Public Key

Sets the CAPK as specified by the CAKey structure

Parameters

<i>capk</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>capkLen</i>	the length of capk data buffer

Returns

RETURN_CODE: Values can be parsed with [device_getIDGStatusCodeString\(\)](#)

11.16.4.89 emv_setCRL()

```
int emv_setCRL (
    IN BYTE * list,
    IN int lsLen )
```

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>lsLen</i>	the length of list data buffer

Returns

RETURN_CODE: Values can be parsed with [device_getResponseCodeString\(\)](#)

11.16.4.90 emv_setTerminalData()

```
int emv_setTerminalData (
    IN BYTE * tlv,
    IN int tlvLen )
```

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 file
<i>tlvLen</i>	the length of tlv data buffer

Return values

<i>RETURN_CODE</i>	Values can be parsed with device_getIDGStatusCodeString()
--------------------	---

11.16.4.91 emv_setTerminalMajorConfiguration()

```
int emv_setTerminalMajorConfiguration (
    IN int configuration )
```

Sets the terminal major configuration in ICS .

Parameters

<i>configuration</i>	<p>A configuration value, range 1-23</p> <ul style="list-style-type: none"> • 1 = 1C • 2 = 2C • 3 = 3C • 4 = 4C • 5 = 5C ... • 23 = 23C
----------------------	---

Returns

RETURN_CODE: Values can be parsed with [device_getResponseCodeString\(\)](#)

11.16.4.92 `emv_setTransactionParameters()`

```
void emv_setTransactionParameters (
    IN double amount,
    IN double amtOther,
    IN int type,
    IN int timeout,
    IN BYTE * tags,
    IN int tagsLen )
```

Set EMV Transaction Parameters

Set the parameters to be used on EMV transactions for an ICC card when Auto Poll is on

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 (Maximum Length = 500 bytes). Passed as a string. Example, tag 9F0C with amount 0x000000000100 would be "9F0C06000000000100" 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 = DFEE1A079F029F36959F37
<i>tagsLen</i>	the length of tags

11.16.4.93 `emv_startTransaction()`

```
int emv_startTransaction (
    IN double amount,
    IN double amtOther,
    IN int exponent,
    IN int type,
    IN int timeout,
    IN BYTE * tags,
    IN int tagsLen,
    IN int forceOnline )
```

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) - SEE IMPORTANT NOTE BELOW
<i>amtOther</i>	Other amount value, if any (tag value 9F03) - SEE IMPORTANT NOTE BELOW
<i>exponent</i>	Number of characters after decimal point
<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 a TLV stream. Example, tag 9F0C with amount 0x000000000100 would be 0x9F0C06000000000100 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>tagsLen</i>	Length of tags
<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 = DFEE1A079F029F36959F37

Returns

RETURN_CODE: Values can be parsed with `device_getResponseCodeString` >>>>IMPORTANT: parameters for amount and amtOther MUST BE PASSED AS A DOUBLE VALUE WITH DECIMAL POINT. Example, do not pass 1, but instead pass 1.0 or 1.00. Otherwise, results will be unpredictable

11.16.4.94 executeTransaction()

```
int executeTransaction (
    WorldPayData * data,
    pWP_callback wpCallback,
    int requestOnly )
```

executeTransaction Requests 3 sets of public keys: encrypting Keys, signing/validating keys, signing/validating 3rd party apps

Parameters

<i>data</i>	WorldPay data object
<i>wpCallback</i>	WorldPay callback
<i>requestOnly</i>	<ul style="list-style-type: none"> • 0 = send transaction and return response, • 1 = send transaction and return both request and response, • 2 = do not send transaction, instead return request

Returns

RETURN_CODE: Values can be parsed with `device_getIDGStatusCodeString()`

11.16.4.95 executeTransaction_WorldNet()

```
int executeTransaction_WorldNet (
    WorldNetData * data,
    pWN_callback wnCallback,
    int requestOnly )
```

executeTransaction_WorldNet Requests 3 sets of public keys: encrypting Keys, signing/validating keys, signing/validating 3rd party apps

Parameters

<i>data</i>	WorldNet data object
<i>wnCallback</i>	WorldNet callback
<i>requestOnly</i>	<ul style="list-style-type: none"> • 0 = send transaction and return response, • 1 = send transaction and return both request and response, • 2 = do not send transaction, instead return request

Returns

RETURN_CODE: Values can be parsed with [device_getIDGStatusCodeString\(\)](#)

11.16.4.96 forwardTransaction()

```
int forwardTransaction (
    IN pWP_callBack wpCallback,
    IN char * forwardID,
    IN int forwardIDLen,
    IN char * password,
    IN int passwordLen,
    IN int bypassProcessing )
```

forwardTransaction Send the saved data to WorldPay and complete the transaction.

Parameters

<i>wpCallback</i>	WPCallback is the callback to send the results. Should be the same as executeTransaction callback.
<i>forwardID</i>	= ID, which could be either unique ID or Memo.
<i>forwardIDLen</i>	The length of forwardID.
<i>password</i>	= password. If null/blank, no password.
<i>passwordLen</i>	The length of passwordLen.
<i>bypassProcess</i>	= true means read the file from disk, but don't send to WorldPay, then delete the transaction as if you did send to WorldPay. The purpose of this is to allow them to delete transactions from the storage without sending to WorldPay.

Returns

RETURN_CODE: Values can be parsed with [device_getIDGStatusCodeString\(\)](#)

11.16.4.97 forwardTransaction_WorldNet()

```
int forwardTransaction_WorldNet (
    IN char * apiKey,
    IN int apiKeyLen,
    IN pWN_callBack wnCallback,
    IN char * forwardID,
```

```

    IN int forwardIDLen,
    IN char * password,
    IN int passwordLen,
    IN int bypassProcessing )

```

forwardTransaction_WorldNet Send the saved data to WorldNet and complete the transaction.

Parameters

<i>wnCallback</i>	WNCallback is the callback to send the results. Should be the same as executeTransaction_WorldNet callback.
<i>forwardID</i>	= ID, which could be either unique ID or Memo.
<i>forwardIDLen</i>	The length of forwardID.
<i>password</i>	= password. If null/blank, no password.
<i>passwordLen</i>	The length of passwordLen.
<i>bypassProcess</i>	= true means read the file from disk, but don't send to WorldPay, then delete the transaction as if you did send to WorldPay. The purpose of this is to allow them to delete transactions from the storage without sending to WorldPay.

Returns

RETURN_CODE: Values can be parsed with [device_getIDGStatusCodeString\(\)](#)

11.16.4.98 icc_exchangeAPDU()

```

int icc_exchangeAPDU (
    IN BYTE * c_APDU,
    IN int cLen,
    OUT BYTE * reData,
    IN_OUT int * reLen )

```

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>cLen</i>	APDU data packet length
<i>reData</i>	Unencrypted APDU response
<i>reLen</i>	Unencrypted APDU response data length

Returns

RETURN_CODE: Values can be parsed with [device_getIDGStatusCodeString](#)

11.16.4.99 icc_getICCRaderStatus()

```

int icc_getICCRaderStatus (
    OUT BYTE * status )

```

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
---------------	--

Returns

RETURN_CODE: Values can be parsed with `device_getIDGStatusCodeString`

11.16.4.100 `icc_powerOffICC()`

```
int icc_powerOffICC ( )
```

Power Off ICC

Powers down the ICC

Returns

RETURN_CODE: Values can be parsed with `device_getIDGStatusCodeString()`

If Success, empty If Failure, ASCII encoded data of error string

11.16.4.101 `icc_powerOnICC()`

```
int icc_powerOnICC (
    OUT BYTE * ATR,
    IN_OUT int * inLen )
```

Power On ICC

Power up the currently selected microprocessor card in the ICC reader

Parameters

<i>ATR,the</i>	ATR data response when succeeded power on ICC,
<i>inLen,the</i>	length of ATR data

Returns

RETURN_CODE: Values can be parsed with `device_getIDGStatusCodeString()`

11.16.4.102 `msr_cancelMSRSwipe()`

```
int msr_cancelMSRSwipe ( )
```

Disable MSR Swipe Cancels MSR swipe request.

Returns

RETURN_CODE: Values can be parsed with `device_getIDGStatusCodeString()`

11.16.4.103 msr_registerCallBk()

```
void msr_registerCallBk (
    pMSR_callback pMSRf )
```

To register the msr callback function to get the MSR card data. (Pass NULL to disable the callback.)

11.16.4.104 msr_registerCallBkp()

```
void msr_registerCallBkp (
    pMSR_callbackp pMSRf )
```

To register the msr callback function to get the MSR card data pointer. (Pass NULL to disable the callback.)

11.16.4.105 msr_startMSRSwipe()

```
int msr_startMSRSwipe (
    IN int _timeout )
```

Start 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
----------------	---------------------

Returns

RETURN_CODE: Values can be parsed with [device_getIDGStatusCodeString\(\)](#) Note: if auto poll mode is on, it will return command not allowed error

11.16.4.106 parseMSRData()

```
void parseMSRData (
    IN BYTE * resData,
    IN int resLen,
    IN_OUT IDTMSRData * cardData )
```

Parser the MSR data from the buffer into IDTMSTData structure

Parameters

<i>resData</i>	MSR card data buffer
<i>resLen</i>	the length of resData
<i>cardData</i>	the parser result with IDTMSTData structure

11.16.4.107 pin_registerCallBk()

```
void pin_registerCallBk (
    pPIN_callback pPINf )
```

To register the pin callback function to get the PINPad data. (Pass NULL to disable the callback.)

11.16.4.108 registerHotplugCallBk()

```
void registerHotplugCallBk (
    pMessageHotplug pMsgHotplug )
```

To register the USB HID hot-plug callback function which implemented in the application to monitor the hotplug message from the SDK.

11.16.4.109 registerLogCallBk()

```
void registerLogCallBk (
    pSendDataLog pFSend,
    pReadDataLog pFRead )
```

To register the log callback function which implemented in the application to monitor sending/reading data between application and reader.

11.16.4.110 SDK_Version()

```
char* SDK_Version ( )
```

To Get SDK version

Returns

return the SDK version string

11.16.4.111 setAbsoluteLibraryPath()

```
int setAbsoluteLibraryPath (
    const char * absoluteLibraryPath )
```

Set the path to use when searching for ID TECH's libraries. If this is not set, the libraries will be searched for with the system's default procedures.

Parameters

<i>absoluteLibraryPath</i>	The absolute path to ID TECH's libraries.
----------------------------	---

Returns

RETURN_CODE: Values can be parsed with [device_getIDGStatusCodeString\(\)](#)

11.17 Source_C/libIDT_VP8800.h File Reference

VP8800 API.

```
#include <stdarg.h>
#include "IDTDef.h"
```

Macros

- `#define IN`

- `#define OUT`
- `#define IN_OUT`

Typedefs

- `typedef void(* pMessageHotplug) (int, int)`
- `typedef void(* pSendDataLog) (BYTE *, int)`
- `typedef void(* pReadDataLog) (BYTE *, int)`
- `typedef void(* pEMV_callBack) (int, int, BYTE *, int, IDTTransactionData *, EMV_Callback *, int)`
- `typedef void(* pMSR_callBack) (int, IDTMSRData)`
- `typedef void(* pMSR_callBackp) (int, IDTMSRData *)`
- `typedef void(* pPIN_callBack) (int, IDTPINData *)`
- `typedef void(* pCMR_callBack) (int, IDTCMRData *)`
- `typedef void(* pCSFS_callBack) (BYTE status)`
- `typedef void(* ftpComm_callBack) (int, int, int)`
- `typedef void(* httpComm_callBack) (BYTE *, int)`
- `typedef void(* v4Comm_callBack) (BYTE, BYTE, BYTE *, int)`
- `typedef void(* pLog_callback) (BYTE, char *)`

Functions

- `void registerHotplugCallBk (pMessageHotplug pMsgHotplug)`
- `void registerLogCallBk (pSendDataLog pFSend, pReadDataLog pFRead)`
- `void emv_registerCallBk (pEMV_callBack pEMVf)`
- `void msr_registerCallBk (pMSR_callBack pMSRf)`
- `void msr_registerCallBkp (pMSR_callBackp pMSRf)`
- `void ctls_registerCallBk (pMSR_callBack pCTLSf)`
- `void ctls_registerCallBkp (pMSR_callBackp pCTLSf)`
- `void pin_registerCallBk (pPIN_callBack pPINf)`
- `void device_registerCameraCallBk (pCMR_callBack pCMRf)`
- `void device_registerCardStatusFrontSwitchCallBk (pCSFS_callBack pCSFSf)`
- `void comm_registerHTTPCallback (httpComm_callBack cBack)`
- `void comm_registerV4Callback (v4Comm_callBack cBack)`
- `char * SDK_Version ()`
- `int setAbsoluteLibraryPath (const char *absoluteLibraryPath)`
- `int device_init ()`
- `int device_setCurrentDevice (int deviceType)`
- `int device_close ()`
- `void device_getIDGStatusCodeString (IN int returnCode, OUT char *despcrition)`
- `int device_isConnected ()`
- `int device_isAttached (int deviceType)`
- `int device_startTransaction (IN double amount, IN double amtOther, IN int type, IN const int _timeout, IN BYTE *tags, IN int tagsLen)`
- `void device_setTransactionExponent (int exponent)`
- `int device_activateTransaction (IN const int _timeout, IN BYTE *tags, IN int tagsLen)`
- `int device_cancelTransaction ()`
- `int device_getDriveFreeSpace (OUT int *free, OUT int *used)`
- `int device_listDirectory (IN char *directoryName, IN int directoryNameLen, IN int recursive, IN int onSD, OUT char *directory, IN_OUT int *directoryLen)`
- `int device_createDirectory (IN char *directoryName, IN int directoryNameLen)`
- `int device_deleteDirectory (IN char *dirName, IN int dirNameLen)`
- `int device_transferFile (IN char *fileName, IN int fileNameLen, IN BYTE *file, IN int fileLen)`
- `int device_deleteFile (IN char *fileName, IN int fileNameLen)`
- `int device_getFirmwareVersion (OUT char *firmwareVersion)`

- int [device_getFirmwareVersion_Len](#) (OUT char *firmwareVersion, IN_OUT int *firmwareVersionLen)
- int [device_pingDevice](#) ()
- int [device_controlUserInterface](#) (IN BYTE *values)
- int [device_controlIndicator](#) (IN int indicator, IN int enable)
- int [device_getCurrentDeviceType](#) ()
- int [device_SendDataCommandNEO](#) (IN int cmd, IN int subCmd, IN BYTE *data, IN int dataLen, OUT BYTE *response, IN_OUT int *respLen)
- int [device_enablePassThrough](#) (int enablePassThrough)
- int [device_enhancedPassthrough](#) (IN BYTE *data, IN int dataLen)
- int [device_setMerchantRecord](#) (int index, int enabled, char *merchantID, char *merchantURL)
- int [device_getMerchantRecord](#) (IN int index, OUT BYTE *record)
- int [device_getMerchantRecord_Len](#) (IN int index, OUT BYTE *record, IN_OUT int *recordLen)
- int [device_getTransactionResults](#) (IDTMSRData *cardData)
- int [device_calibrateParameters](#) (BYTE delta)
- int [config_getSerialNumber](#) (OUT char *sNumber)
- int [config_getSerialNumber_Len](#) (OUT char *sNumber, IN_OUT int *sNumberLen)
- int [device_getSDKWaitTime](#) ()
- void [device_setSDKWaitTime](#) (int waitTime)
- int [device_getThreadStackSize](#) ()
- void [device_setThreadStackSize](#) (int threadSize)
- int [ctls_startTransaction](#) (IN double amount, IN double amtOther, IN int type, IN const int _timeout, IN BYTE *tags, IN int tagsLen)
- int [ctls_activateTransaction](#) (IN const int _timeout, IN BYTE *tags, IN int tagsLen)
- int [ctls_cancelTransaction](#) ()
- int [ctls_retrieveApplicationData](#) (IN BYTE *AID, IN int AIDLen, OUT BYTE *tlv, IN_OUT int *tlvLen)
- int [ctls_setApplicationData](#) (IN BYTE *tlv, IN int tlvLen)
- int [ctls_removeApplicationData](#) (IN BYTE *AID, IN int AIDLen)
- int [ctls_removeAllApplicationData](#) ()
- int [ctls_retrieveAIDList](#) (OUT BYTE *AIDList, IN_OUT int *AIDListLen)
- int [ctls_retrieveTerminalData](#) (OUT BYTE *tlv, IN_OUT int *tlvLen)
- int [ctls_setTerminalData](#) (IN BYTE *tlv, IN int tlvLen)
- int [ctls_retrieveCAPK](#) (IN BYTE *capk, IN int capkLen, OUT BYTE *key, IN_OUT int *keyLen)
- int [ctls_setCAPK](#) (IN BYTE *capk, IN int capkLen)
- int [ctls_removeCAPK](#) (IN BYTE *capk, IN int capkLen)
- int [ctls_removeAllCAPK](#) ()
- int [ctls_retrieveCAPKList](#) (OUT BYTE *keys, IN_OUT int *keysLen)
- int [ctls_setConfigurationGroup](#) (IN BYTE *tlv, IN int tlvLen)
- int [ctls_getConfigurationGroup](#) (IN int group, OUT BYTE *tlv, OUT int *tlvLen)
- int [ctls_getAllConfigurationGroups](#) (OUT BYTE *tlv, IN_OUT int *tlvLen)
- int [ctls_removeConfigurationGroup](#) (int group)
- int [ctls_displayOnlineAuthResult](#) (IN int statusCode, IN BYTE *TLV, IN int TLVLen)
- void [emv_allowFallback](#) (IN int allow)
- void [emv_setAutoAuthenticateTransaction](#) (IN int authenticate)
- void [emv_setAutoCompleteTransaction](#) (IN int complete)
- int [emv_getAutoAuthenticateTransaction](#) ()
- int [emv_getAutoCompleteTransaction](#) ()
- int [emv_startTransaction](#) (IN double amount, IN double amtOther, IN int exponent, IN int type, IN int timeout, IN BYTE *tags, IN int tagsLen, IN int forceOnline)
- int [emv_activateTransaction](#) (IN int timeout, IN BYTE *tags, IN int tagsLen, IN int forceOnline)
- int [emv_authenticateTransaction](#) (IN BYTE *updatedTLV, IN int updatedTLVLen)
- int [emv_authenticateTransactionWithTimeout](#) (IN int timeout, IN BYTE *updatedTLV, IN int updatedTLVLen)
- int [emv_completeTransaction](#) (IN int commError, IN BYTE *authCode, IN int authCodeLen, IN BYTE *iad, IN int iadLen, IN BYTE *tlvScripts, IN int tlvScriptsLen, IN BYTE *tlv, IN int tlvLen)
- int [emv_cancelTransaction](#) ()
- int [emv_retrieveApplicationData](#) (IN BYTE *AID, IN int AIDLen, OUT BYTE *tlv, IN_OUT int *tlvLen)

- int [emv_setApplicationData](#) (IN BYTE *name, IN int nameLen, IN BYTE *tlv, IN int tlvLen)
- int [emv_setApplicationDataTLV](#) (IN BYTE *tlv, IN int tlvLen)
- int [emv_removeApplicationData](#) (IN BYTE *AID, IN int AIDLen)
- int [emv_removeAllApplicationData](#) ()
- int [emv_retrieveAIDList](#) (OUT BYTE *AIDList, IN_OUT int *AIDListLen)
- int [emv_retrieveTerminalData](#) (OUT BYTE *tlv, IN_OUT int *tlvLen)
- int [emv_setTerminalData](#) (IN BYTE *tlv, IN int tlvLen)
- int [emv_retrieveCAPK](#) (IN BYTE *capk, IN int capkLen, OUT BYTE *key, IN_OUT int *keyLen)
- int [emv_setCAPK](#) (IN BYTE *capk, IN int capkLen)
- int [emv_removeCAPK](#) (IN BYTE *capk, IN int capkLen)
- int [emv_removeAllCAPK](#) ()
- int [emv_retrieveCAPKList](#) (OUT BYTE *keys, IN_OUT int *keysLen)
- int [emv_retrieveExceptionList](#) (OUT BYTE *exceptionList, IN_OUT int *exceptionListLen)
- int [emv_setException](#) (IN BYTE *exception, IN int exceptionLen)
- int [emv_removeException](#) (IN BYTE *exception, IN int exceptionLen)
- int [emv_removeAllExceptions](#) ()
- int [emv_retrieveExceptionLogStatus](#) (OUT BYTE *exceptionLogStatus, IN_OUT int *exceptionLogStatusLen)
- int [emv_removeTransactionLog](#) ()
- int [emv_retrieveTransactionLogStatus](#) (OUT BYTE *transactionLogStatus, IN_OUT int *transactionLogStatusLen)
- int [emv_retrieveTransactionLog](#) (OUT BYTE *transactionLog, IN_OUT int *transactionLogLen, IN_OUT int *remainingTransactionLogLen)
- int [emv_getEMVKernelVersion](#) (OUT char *version)
- int [emv_getEMVKernelVersion_Len](#) (OUT char *version, IN_OUT int *versionLen)
- int [emv_getEMVKernelCheckValue](#) (OUT BYTE *checkValue, IN_OUT int *checkValueLen)
- int [emv_getEMVConfigurationCheckValue](#) (OUT BYTE *checkValue, IN_OUT int *checkValueLen)
- int [emv_retrieveCRL](#) (OUT BYTE *list, IN_OUT int *listLen)
- int [emv_setCRL](#) (IN BYTE *list, IN int listLen)
- int [emv_removeCRL](#) (IN BYTE *list, IN int listLen)
- int [emv_removeAllCRL](#) ()
- int [lcd_resetInitialState](#) ()
- int [lcd_customDisplayMode](#) (IN int enable)
- int [lcd_setForeBackColor](#) (IN BYTE *foreRGB, IN int foreRGBLen, IN BYTE *backRGB, IN int backRGBLen)
- int [lcd_clearDisplay](#) (IN BYTE control)
- int [lcd_captureSignature](#) (IN int timeout)
- int [lcd_startSlideShow](#) (IN char *files, IN int filesLen, IN int posX, IN int posY, IN int posMode, IN int touchEnable, IN int recursion, IN int touchTerminate, IN int delay, IN int loops, IN int clearScreen)
- int [lcd_cancelSlideShow](#) (OUT BYTE *statusCode, IN_OUT int *statusCodeLen)
- int [lcd_setDisplayImage](#) (IN char *file, IN int fileLen, IN int posX, IN int posY, IN int posMode, IN int touchEnable, IN int clearScreen)
- int [lcd_setBackgroundImage](#) (IN char *file, IN int fileLen, IN int enable)
- int [lcd_displayText](#) (IN int posX, IN int posY, IN int displayWidth, IN int displayHeight, IN int fontDesignation, IN int fontID, IN int screenPosition, IN char *displayText, OUT BYTE *graphicsID)
- int [lcd_displayText_Len](#) (IN int posX, IN int posY, IN int displayWidth, IN int displayHeight, IN int fontDesignation, IN int fontID, IN int screenPosition, IN char *displayText, OUT BYTE *graphicsID, IN_OUT int *graphicsIDLen)
- int [lcd_displayParagraph](#) (IN int posX, IN int posY, IN int displayWidth, IN int displayHeight, IN int fontDesignation, IN int fontID, IN int displayProperties, IN char *displayText)
- int [lcd_displayButton](#) (IN int posX, IN int posY, IN int buttonWidth, IN int buttonHeight, IN int fontDesignation, IN int fontID, IN int displayPosition, IN char *buttonLabel, IN int buttonTextColorR, IN int buttonTextColorG, IN int buttonTextColorB, IN int buttonBackgroundColorR, IN int buttonBackgroundColorG, IN int buttonBackgroundColorB, OUT BYTE *graphicsID)

- int [lcd_displayButton_Len](#) (IN int posX, IN int posY, IN int buttonWidth, IN int buttonHeight, IN int font↵ Designation, IN int fontID, IN int displayPosition, IN char *buttonLabel, IN int buttonTextColorR, IN int button↵ TextColorG, IN int buttonTextColorB, IN int buttonBackgroundColorR, IN int buttonBackgroundColorG, IN int buttonBackgroundColorB, OUT BYTE *graphicsID, IN_OUT int *graphicsIDLen)
- int [lcd_createList](#) (IN int posX, IN int posY, IN int numOfColumns, IN int numOfRows, IN int fontDesignation, IN int fontID, IN int verticalScrollArrowsVisible, IN int borderedListItems, IN int borderedScrollArrows, IN int touchSensitive, IN int automaticScrolling, OUT BYTE *graphicsID)
- int [lcd_createList_Len](#) (IN int posX, IN int posY, IN int numOfColumns, IN int numOfRows, IN int font↵ Designation, IN int fontID, IN int verticalScrollArrowsVisible, IN int borderedListItems, IN int borderedScroll↵ Arrows, IN int touchSensitive, IN int automaticScrolling, OUT BYTE *graphicsID, IN_OUT int *graphicsIDLen)
- int [lcd_addItemToList](#) (IN BYTE *listGraphicsID, IN char *itemName, IN char *itemID, IN int selected)
- int [lcd_getSelectedItem](#) (IN BYTE *listGraphicsID, OUT char *itemID)
- int [lcd_getSelectedItem_Len](#) (IN BYTE *listGraphicsID, OUT char *itemID, IN_OUT int *itemIDLen)
- int [lcd_clearEventQueue](#) ()
- int [lcd_getInputEvent](#) (IN int timeout, OUT int *dataReceived, OUT BYTE *eventType, OUT BYTE *graphics↵ ID, OUT BYTE *eventData)
- int [lcd_getInputEvent_Len](#) (IN int timeout, OUT int *dataReceived, OUT BYTE *eventType, IN_OUT int ↵ *eventTypeLen, OUT BYTE *graphicsID, IN_OUT int *graphicsIDLen, OUT BYTE *eventData, IN_OUT int ↵ *eventDataLen)
- int [lcd_createInputField](#) (IN BYTE *specs, IN int specsLen, OUT BYTE *graphicId)
- int [lcd_createInputField_Len](#) (IN BYTE *specs, IN int specsLen, OUT BYTE *graphicId, IN_OUT int ↵ *graphicIdLen)
- int [lcd_getInputFieldValue](#) (IN BYTE *graphicId, OUT BYTE *retData, IN_OUT int *retDataLen)
- int [msr_cancelMSRSwipe](#) ()
- int [msr_startMSRSwipe](#) (IN int _timeout)
- int [msr_flushTrackData](#) ()
- void [parseMSRData](#) (IN BYTE *resData, IN int resLen, IN_OUT IDTMSRData *cardData)
- int [pin_getEncryptedOnlinePIN](#) (IN int keyType, IN int timeout)
- int [pin_getPAN](#) (IN int getCSC, IN int timeout)
- int [pin_promptCreditDebit](#) (IN char *currencySymbol, IN int currencySymbolLen, IN char *displayAmount, IN ↵ int displayAmountLen, IN int timeout, OUT BYTE *retData, IN_OUT int *retDataLen)
- int [ws_requestCSR](#) (OUT RequestCSR *csr)
- int [ws_loadSSLCert](#) (IN char *name, IN int nameLen, IN char *dataDER, IN int dataDERLen)
- int [ws_revokeSSLCert](#) (IN char *name, IN int nameLen)
- int [ws_deleteSSLCert](#) (IN char *name, IN int nameLen)
- int [ws_getCertChainType](#) (OUT int *type)
- int [ws_updateRootCertificate](#) (IN char *name, IN int nameLen, IN char *dataDER, IN int dataDERLen, IN ↵ char *signature, IN int signatureLen)

11.17.1 Detailed Description

VP8800 API.

VP8800 Global API methods.

11.17.2 Macro Definition Documentation

11.17.2.1 IN

```
#define IN
```

INPUT parameter.

11.17.2.2 IN_OUT

```
#define IN_OUT
```

INPUT / OUTPUT PARAMETER.

11.17.2.3 OUT

```
#define OUT
```

OUTPUT parameter.

11.17.3 Typedef Documentation

11.17.3.1 ftpComm_callBack

```
typedef void(* ftpComm_callBack) (int, int, int)
```

Define the comm callback function to get FTP file transfer status

It should be passed as a parameter in a FTP request, Signature (int, int, int) = response code, current block, total blocks
 RESPONSE CODES: 100 = FILE DOWNLOAD STARTED 101 = FILE BLOCK XX OF XX RECEIVED 102 = FILE DOWNLOAD COMPLETED 103 = FILE DOWNLOAD TERMINATED PREMATURELY

11.17.3.2 httpComm_callBack

```
typedef void(* httpComm_callBack) (BYTE *, int)
```

Define the comm callback function to get the async url data

It should be registered using the comm_registerHTTPCallback

11.17.3.3 pCMR_callBack

```
typedef void(* pCMR_callBack) (int, IDTCMRData *)
```

Define the camera callback function to get the image data

It should be registered using the device_registerCameraCallBk,

11.17.3.4 pCSFS_callBack

```
typedef void(* pCSFS_callBack) (BYTE status)
```

Define the card status and front switch callback function to get card and front switch status

It should be registered using the device_registerCardStatusFrontSwitchCallBk,

11.17.3.5 pEMV_callBack

```
typedef void(* pEMV_callBack) (int, int, BYTE *, int, IDTTransactionData *, EMV_Callback *, int)
```

Define the EMV callback function to get the transaction message/data/result.

It should be registered using the emv_registerCallBk,

11.17.3.6 pLog_callback

```
typedef void(* pLog_callback) (BYTE, char *)
```

Define the log callback function to receive log messages.

11.17.3.7 pMessageHotplug

```
typedef void(* pMessageHotplug) (int, int)
```

Define the USB hot-plug callback function to monitor the info when plug in/out the reader.

It should be registered using the registerHotplugCallBk, The first integer parameter is device type, and the second integer parameter is either 0: Device Plugged Out or 1: Device Plugged In

11.17.3.8 pMSR_callBack

```
typedef void(* pMSR_callBack) (int, IDTMSRData)
```

Define the MSR callback function to get the MSR card data

It should be registered using the msr_registerCallBk, this callback function is for backward compatibility

11.17.3.9 pMSR_callBackp

```
typedef void(* pMSR_callBackp) (int, IDTMSRData *)
```

Define the MSR callback function to get pointer to the MSR card data

It should be registered using the msr_registerCallBk, this callback function is recommended instead of pMSR_callBack

11.17.3.10 pPIN_callBack

```
typedef void(* pPIN_callBack) (int, IDTPINData *)
```

Define the PINPad callback function to get the input PIN Pad data

It should be registered using the pin_registerCallBk,

11.17.3.11 pReadDataLog

```
typedef void(* pReadDataLog) (BYTE *, int)
```

Define the read response callback function to monitor the reading response from the reader.

It should be registered using the registerLogCallBk,

11.17.3.12 pSendDataLog

```
typedef void(* pSendDataLog) (BYTE *, int)
```

Define the send command callback function to monitor the sending command into the reader.

It should be registered using the registerLogCallBk,

11.17.3.13 v4Comm_callBack

```
typedef void(* v4Comm_callBack) (BYTE, BYTE, BYTE *, int)
```

Define the comm callback function to receive the V4 Protocol packets received by the device from an external source (IP/USB/RS-232) It should be registered using the comm_registerV4Callback, Data callback will contain command, sub-command, and data from V4 packet

11.17.4 Function Documentation

11.17.4.1 comm_registerHTTPCallback()

```
void comm_registerHTTPCallback (
    httpComm_callBack cBack )
```

Register Comm HTTP Async Callback

Parameters

<i>cBack</i>	? HTTP Comm callback
--------------	----------------------

11.17.4.2 comm_registerV4Callback()

```
void comm_registerV4Callback (
    v4Comm_callBack cBack )
```

Register External V4 Protocol commands Callback

Parameters

<i>cBack</i>	? V4 Protocol Comm callback
--------------	-----------------------------

11.17.4.3 config_getSerialNumber()

```
int config_getSerialNumber (
    OUT char * sNumber )
```

DEPRECATED : please use [config_getSerialNumber_Len](#)(OUT char* sNumber, IN_OUT int *sNumberLen)

Polls device for Serial Number

Parameters

<i>sNumber</i>	Returns Serial Number; needs to have at least 64 bytes of memory
----------------	--

Returns

RETURN_CODE: Values can be parsed with [device_getIDGStatusCodeString](#)

11.17.4.4 config_getSerialNumber_Len()

```
int config_getSerialNumber_Len (
    OUT char * sNumber,
    IN_OUT int * sNumberLen )
```

Polls device for Serial Number

Parameters

<i>sNumber</i>	Returns Serial Number
<i>sNumberLen</i>	Length of Serial Number

Returns

RETURN_CODE: Values can be parsed with `device_getIDGStatusCodeString`

11.17.4.5 `ctls_activateTransaction()`

```
int ctls_activateTransaction (
    IN const int _timeout,
    IN BYTE * tags,
    IN int tagsLen )
```

Start CTLS Transaction Request

Authorizes the CTLS 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 stream. Example, tag 9F0C with amount 0x000000000100 would be 0x9F0C06000000000100
<i>tagsLen</i>	The length of tags data buffer.

>>>>>IMPORTANT: parameters for amount and amtOther MUST BE PASSED AS A DOUBLE VALUE WITH DECIMAL POINT. Example, do not pass 1, but instead pass 1.0 or 1.00. Otherwise, results will be unpredictable

Returns

RETURN_CODE: Values can be parsed with [device_getIDGStatusCodeString\(\)](#) Note: if auto poll is on, it will return the error IDG_P2_STATUS_CODE_COMMAND_NOT_ALLOWED

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

11.17.4.6 `ctls_cancelTransaction()`

```
int ctls_cancelTransaction ( )
```

Cancel EMV Transaction

Cancels the currently executing EMV transaction.

Returns

RETURN_CODE: Values can be parsed with [device_getIDGStatusCodeString\(\)](#)

11.17.4.7 `ctls_displayOnlineAuthResult()`

```
int ctls_displayOnlineAuthResult (
    IN int statusCode,
    IN BYTE * TLV,
    IN int TLVLen )
```

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. The SDK timeout of the command is set to 7 seconds.

Parameters

<i>statusCode</i>	1 = OK, 0 = NOT OK, 2 = ARC response 89 for Interac
<i>TLV</i>	Optional TLV for AOSA
<i>TLVLen</i>	TLV Length

Returns

RETURN_CODE: Values can be parsed with [device_getIDGStatusCodeString\(\)](#)

11.17.4.8 ctls_getAllConfigurationGroups()

```
int ctls_getAllConfigurationGroups (
    OUT BYTE * tlv,
    IN_OUT int * tlvLen )
```

Retrieve All Configuration Groups

Returns all the Configuration Groups installed on the terminal for CTLS

Parameters

<i>tlv</i>	The TLV elements data
<i>tlvLen</i>	the length of tlv data buffer.

Returns

RETURN_CODE: Values can be parsed with [device_getIDGStatusCodeString\(\)](#)

11.17.4.9 ctls_getConfigurationGroup()

```
int ctls_getConfigurationGroup (
    IN int group,
    OUT BYTE * tlv,
    OUT int * tlvLen )
```

Get Configuration Group

Retrieves the Configuration for the specified Group.

Parameters

<i>group</i>	Configuration Group
<i>tlv</i>	return data
<i>tlvLen</i>	the length of tlv data buffer

Returns

RETURN_CODE: Values can be parsed with [device_getIDGStatusCodeString\(\)](#)

11.17.4.10 ctls_registerCallBk()

```
void ctls_registerCallBk (
    pMSR_callback pCTLSf )
```

To register the ctls callback function to get the MSR card data. (Pass NULL to disable the callback.)

11.17.4.11 ctls_registerCallBkp()

```
void ctls_registerCallBkp (
    pMSR_callback pCTLSf )
```

To register the ctls callback function to get the MSR card data pointer. (Pass NULL to disable the callback.)

11.17.4.12 ctls_removeAllApplicationData()

```
int ctls_removeAllApplicationData ( )
```

Remove All Application Data

Removes all the Application Data

Returns

RETURN_CODE: Values can be parsed with [device_getIDGStatusCodeString\(\)](#)

11.17.4.13 ctls_removeAllCAPK()

```
int ctls_removeAllCAPK ( )
```

Remove All Certificate Authority Public Key

Removes all the CAPK

Returns

RETURN_CODE: Values can be parsed with [device_getIDGStatusCodeString\(\)](#)

11.17.4.14 ctls_removeApplicationData()

```
int ctls_removeApplicationData (
    IN BYTE * AID,
    IN int AIDLen )
```

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>AIDLen</i>	the length of AID data buffer

Returns

RETURN_CODE: Values can be parsed with [device_getIDGStatusCodeString\(\)](#)

11.17.4.15 ctls_removeCAPK()

```
int ctls_removeCAPK (
```

```
IN BYTE * capk,  
IN int capkLen )
```

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>capkLen</i>	the length of capk data buffer

Returns

RETURN_CODE: Values can be parsed with [device_getIDGStatusCodeString\(\)](#)

11.17.4.16 ctls_removeConfigurationGroup()

```
int ctls_removeConfigurationGroup (  
    int group )
```

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>	Values can be parsed with device_getIDGStatusCodeString()
--------------------	---

11.17.4.17 ctls_retrieveAIDList()

```
int ctls_retrieveAIDList (  
    OUT BYTE * AIDList,  
    IN_OUT int * AIDListLen )
```

Retrieve AID list

Returns all the AID names installed on the terminal for CTLS. .

Parameters

<i>AIDList</i>	array of AID name byte arrays
<i>AIDListLen</i>	the length of AIDList array buffer

Returns

RETURN_CODE: Values can be parsed with [device_getIDGStatusCodeString\(\)](#)

11.17.4.18 ctls_retrieveApplicationData()

```
int ctls_retrieveApplicationData (
    IN BYTE * AID,
    IN int AIDLen,
    OUT BYTE * tlv,
    IN_OUT int * tlvLen )
```

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>AIDLen</i>	the length of AID data buffer.
<i>tlv</i>	The TLV elements of the requested AID
<i>tlvLen</i>	the length of tlv data buffer.

Returns

RETURN_CODE: Values can be parsed with [device_getIDGStatusCodeString\(\)](#)

11.17.4.19 ctls_retrieveCAPK()

```
int ctls_retrieveCAPK (
    IN BYTE * capk,
    IN int capkLen,
    OUT BYTE * key,
    IN_OUT int * keyLen )
```

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>capkLen</i>	the length of capk data buffer

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>keyLen</i>	<p>the length of key data buffer</p> <ul style="list-style-type: none"> •

Returns

RETURN_CODE: Values can be parsed with [device_getIDGStatusCodeString\(\)](#)

11.17.4.20 `ctls_retrieveCAPKList()`

```
int ctls_retrieveCAPKList (
    OUT BYTE * keys,
    IN_OUT int * keyLen )
```

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>keyLen</i>	the length of keys data buffer

Returns

RETURN_CODE: Values can be parsed with [device_getIDGStatusCodeString\(\)](#)

11.17.4.21 `ctls_retrieveTerminalData()`

```
int ctls_retrieveTerminalData (
    OUT BYTE * tlv,
    IN_OUT int * tlvLen )
```

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>tlvLen</i>	the length of tlv data buffer

Returns

RETURN_CODE: Values can be parsed with [device_getIDGStatusCodeString\(\)](#)

11.17.4.22 `ctls_setApplicationData()`

```
int ctls_setApplicationData (  
    IN BYTE * tlv,  
    IN int tlvLen )
```

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>tlvLen</i>	the length of tlv data buffer
---------------	-------------------------------

Returns

RETURN_CODE: Values can be parsed with [device_getIDGStatusCodeString\(\)](#)

11.17.4.23 `ctls_setCAPK()`

```
int ctls_setCAPK (  
    IN BYTE * capk,  
    IN int capkLen )
```

Set Certificate Authority Public Key

Sets the CAPK as specified by the CAKey structure

Parameters

<i>capk</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>capkLen</i>	the length of capk data buffer

Returns

RETURN_CODE: Values can be parsed with [device_getIDGStatusCodeString\(\)](#)

11.17.4.24 **ctls_setConfigurationGroup()**

```
int ctls_setConfigurationGroup (
    IN BYTE * tlv,
    IN int tlvLen )
```

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>tlvLen</i>	the length of tlv data buffer

Returns

RETURN_CODE: Values can be parsed with [device_getIDGStatusCodeString\(\)](#)

11.17.4.25 **ctls_setTerminalData()**

```
int ctls_setTerminalData (
    IN BYTE * tlv,
    IN int tlvLen )
```

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 `ctls_getConfigurationGroup(int group)`, and deleted with `ctls_removeConfigurationGroup(int group)`. You cannot delete group 0.

Parameters

<i>tlv</i>	TerminalData configuration file
<i>tlvLen</i>	the length of tlv data buffer

Return values

<i>RETURN_CODE</i>	Values can be parsed with <code>device_getIDGStatusCodeString()</code>
--------------------	--

11.17.4.26 ctls_startTransaction()

```
int ctls_startTransaction (
    IN double amount,
    IN double amtOther,
    IN int type,
    IN const int _timeout,
    IN BYTE * tags,
    IN int tagsLen )
```

Start CTLS Transaction Request

Authorizes the CTLS transaction for an ICC card

The tags will be returned in the callback routine.

Parameters

<i>amount</i>	Transaction amount value (tag value 9F02) - SEE IMPORTANT NOTE BELOW
<i>amtOther</i>	Other amount value, if any (tag value 9F03) - SEE IMPORTANT NOTE BELOW
<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 stream. Example, tag 9F0C with amount 0x000000000100 would be 0x9F0C06000000000100 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>tagsLen</i>	The length of tags data buffer.

>>>>>IMPORTANT: parameters for amount and amtOther MUST BE PASSED AS A DOUBLE VALUE WITH DECIMAL POINT. Example, do not pass 1, but instead pass 1.0 or 1.00. Otherwise, results will be unpredictable

Returns

RETURN_CODE: Values can be parsed with `device_getIDGStatusCodeString()` Note: if auto poll is on, it will return the error IDG_P2_STATUS_CODE_COMMAND_NOT_ALLOWED

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

11.17.4.27 device_activateTransaction()

```
int device_activateTransaction (
    IN const int _timeout,
    IN BYTE * tags,
    IN int tagsLen )
```

Start Transaction Request

Authorizes the transaction for an MSR/CTLS/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. Example, tag 9F0C with amount 0x000000000100 would be 0x9F0C06000000000100 Be sure to include 9F02 (amount) and 9C (transaction type).
<i>tagsLen</i>	The length of tags data buffer.

>>>>>IMPORTANT: parameters for amount and amtOther MUST BE PASSED AS A DOUBLE VALUE WITH DECIMAL POINT. Example, do not pass 1, but instead pass 1.0 or 1.00. Otherwise, results will be unpredictable

Returns

RETURN_CODE: Values can be parsed with [device_getIDGStatusCodeString\(\)](#) Note: if auto poll is on, it will return the error IDG_P2_STATUS_CODE_COMMAND_NOT_ALLOWED

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 `device_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

11.17.4.28 device_calibrateParameters()

```
int device_calibrateParameters (
    BYTE delta )
```

Calibrate reference parameters

Parameters

<i>delta</i>	Delta value (0x02 standard default value)
--------------	---

Returns

RETURN_CODE: Values can be parsed with [device_getIDGStatusCodeString\(\)](#)

11.17.4.29 device_cancelTransaction()

```
int device_cancelTransaction ( )
```

Cancel Transaction

Cancels the currently executing transaction.

Returns

RETURN_CODE: Values can be parsed with [device_getIDGStatusCodeString\(\)](#)

11.17.4.30 device_close()

```
int device_close ( )
```

Close the device

Returns

RETURN_CODE: 0: success, 0x0A: failed

11.17.4.31 device_controlIndicator()

```
int device_controlIndicator (
    IN int indicator,
    IN int enable )
```

Control Indicators

Control the reader. If connected, returns success. Otherwise, returns timeout.

Parameters

<i>indicator</i>	description as follows: <ul style="list-style-type: none">• 00h: ICC LED• 01h: Blue MSR• 02h: Red MSR• 03h: Green MSR
<i>enable</i>	TRUE = ON, FALSE = OFF

Returns

success or error code. Values can be parsed with `device_getResponseCodeString`

See also

`ErrorCode`

11.17.4.32 device_controlUserInterface()

```
int device_controlUserInterface (  
    IN BYTE * values )
```

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) • 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
---------------	--

Returns

RETURN_CODE: Values can be parsed with [device_getIDGStatusCodeString\(\)](#)

11.17.4.33 device_createDirectory()

```
int device_createDirectory (
    IN char * directoryName,
    IN int directoryNameLen )
```

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>directoryNameLen</i>	Directory Name Length.

Returns

RETURN_CODE: Values can be parsed with [device_getIDGStatusCodeString\(\)](#)

11.17.4.34 device_deleteDirectory()

```
int device_deleteDirectory (
    IN char * dirName,
    IN int dirNameLen )
```

Delete Directory This command deletes an empty directory. For NEO 2 devices, it will delete the directory even the directory is not empty.

Parameters

<i>dirName</i>	Complete path of the directory you want to delete. You do not need to specify the root directory. Indicate subdirectories with a forward slash (/). For NEO 2 devices, to delete the root directory, simply pass "" with 0 for dirNameLen.
<i>dirNameLen</i>	Directory Name Length.

Returns

RETURN_CODE: Values can be parsed with [device_getIDGStatusCodeString\(\)](#)

11.17.4.35 device_deleteFile()

```
int device_deleteFile (
    IN char * fileName,
    IN int fileNameLen )
```

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>filenameLen</i>	File Name Length.

Returns

RETURN_CODE: Values can be parsed with [device_getIDGStatusCodeString\(\)](#)

11.17.4.36 device_enablePassThrough()

```
int device_enablePassThrough (
    int enablePassThrough )
```

Enable Pass Through

Enables Pass Through Mode for direct communication with L1 interface (power on icc, send apdu, etc).

Parameters

<i>enablePassThrough</i>	1 = pass through ON, 0 = pass through OFF
--------------------------	---

Returns

RETURN_CODE: Values can be parsed with [device_getIDGStatusCodeString\(\)](#)

11.17.4.37 device_enhancedPassthrough()

```
int device_enhancedPassthrough (
    IN BYTE * data,
    IN int dataLen )
```

Enables pass through mode for ICC. Required when direct ICC commands are required (power on/off ICC, exchange APDU)

Parameters

<i>data</i>	The data includes Poll Timeout, Flags, Contact Interface to Use, Beep Indicator, LED Status, and Display Strings.
<i>dataLen</i>	length of data

Returns

success or error code. Values can be parsed with [device_getIDGStatusCodeString](#)

See also

ErrorCode

11.17.4.38 device_getCurrentDeviceType()

```
int device_getCurrentDeviceType ( )
```

Get current active device type

Returns

: return the device type defined as DEVICE_TYPE in the IDTDef.h

11.17.4.39 device_getDriveFreeSpace()

```
int device_getDriveFreeSpace (
    OUT int * free,
    OUT int * used )
```

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

Returns

RETURN_CODE: Values can be parsed with [device_getIDGStatusCodeString\(\)](#)

11.17.4.40 device_getFirmwareVersion()

```
int device_getFirmwareVersion (
    OUT char * firmwareVersion )
```

DEPRECATED : please use [device_getFirmwareVersion_Len\(\)](#)(OUT char* firmwareVersion, IN_OUT int *firmwareVersionLen)

Polls device for Firmware Version

Parameters

<i>firmwareVersion</i>	Response returned of Firmware Version; needs to have at least 128 bytes of memory
------------------------	---

Returns

RETURN_CODE: Values can be parsed with [device_getIDGStatusCodeString\(\)](#)

11.17.4.41 device_getFirmwareVersion_Len()

```
int device_getFirmwareVersion_Len (
    OUT char * firmwareVersion,
    IN_OUT int * firmwareVersionLen )
```

Polls device for Firmware Version

Parameters

<i>firmwareVersion</i>	Response returned of Firmware Version
<i>firmwareVersionLen</i>	Length of Firmware Version

Returns

RETURN_CODE: Values can be parsed with [device_getIDGStatusCodeString\(\)](#)

11.17.4.42 device_getIDGStatusCodeString()

```
void device_getIDGStatusCodeString (
    IN int  returnCode,
    OUT char * despcrition )
```

Review the return code description.

Parameters

<i>returnCode</i>	the response result.
<i>description</i>	

Return values

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

- 0: "no error, beginning task";
- 1: "no response from reader";
- 2: "invalid response data";
- 01: " Incorrect Header Tag";
- 02: " Unknown Command";
- 03: " Unknown Sub-Command";
- 04: " CRC Error in Frame";
- 05: " Incorrect Parameter";
- 06: " Parameter Not Supported";
- 07: " Mal-formatted Data";
- 08: " Timeout";
- 0A: " Failed / NACK";
- 0B: " Command not Allowed";
- 0C: " Sub-Command not Allowed";
- 0D: " Buffer Overflow (Data Length too large for reader buffer)";
- 0E: " User Interface Event";
- 10: " Need clear firmware(apply in boot loader only)";
- 11: " Communication type not supported, VT-1, burst, etc. Need encrypted firmware (apply in boot loader only)";
- 12: " Secure interface is not functional or is in an intermediate state.";

- 13: " Data field is not mod 8";
- 14: " Pad 0x80 not found where expected";
- 15: " Specified key type is invalid";
- 16: " Could not retrieve key from the SAM (InitSecureComm)";
- 17: " Hash code problem";
- 18: " Could not store the key into the SAM (InstallKey)";
- 19: " Frame is too large";
- 1A: " Unit powered up in authentication state but POS must resend the InitSecureComm command";
- 1B: " The EEPROM may not be initialized because SecCommInterface does not make sense";
- 1C: " Problem encoding APDU Module-Specific Status Codes ";
- 20: " Unsupported Index (ILM) SAM Transceiver error - problem communicating with the SAM (Key Mgr)";
- 21: " Unexpected Sequence Counter in multiple frames for single bitmap (ILM)Length error in data returned from the SAM (Key Mgr) - 22: " Improper bit map (ILM)"; - 23: " Request Online Authorization"; - 24: " ViV← OCard3 raw data read successful"; - 25: " Message index not available (ILM) ViVOcomm activate transaction card type (ViVOcomm)"; - 26: " Version Information Mismatch (ILM)"; - 27: " Not sending commands in correct index message index (ILM)"; - 28: " Time out or next expected message not received (ILM)"; - 29: " ILM languages not available for viewing (ILM)"; - 2A: " Other language not supported (ILM)"; - 41: " from 41 to 4F, Module-specific errors for Key Manager";
- 50: " Auto-Switch OK";
- 51: " Auto-Switch failed";
- 70: " Antenna Error 80h Use another card";
- 81: " Insert or swipe card";
- 90: " Data encryption Key does not exist";
- 91: " Data encryption Key KSN exhausted";

11.17.4.43 device_getMerchantRecord()

```
int device_getMerchantRecord (
    IN int index,
    OUT BYTE * record )
```

DEPRECATED : please use device_getMerchantRecord_Len(IN int index, OUT BYTE * record, IN_OUT int *recordLen)

Get Merchant Record

Gets the merchant record for the device.

Parameters

<i>index</i>	Merchant Record index, valid values 1-6
<i>record;</i>	needs to have at least 99 bytes of memory response data from reader. Merchant Record Index: 1 byte enabled: 1 byte Merchant ID: 32 bytes Length of Merchant URL: 1 byte Merchant URL: 64 bytes

Returns

success or error code. Values can be parsed with [device_getIDGStatusCodeString\(\)](#)

See also

[ErrorCode](#)

11.17.4.44 device_getMerchantRecord_Len()

```
int device_getMerchantRecord_Len (
    IN int index,
    OUT BYTE * record,
    IN_OUT int * recordLen )
```

Get Merchant Record

Gets the merchant record for the device.

Parameters

<i>index</i>	Merchant Record index, valid values 1-6
<i>record</i>	response data from reader. Merchant Record Index: 1 byte enabled: 1 byte Merchant ID: 32 bytes Length of Merchant URL: 1 byte Merchant URL: 64 bytes
<i>recordLen</i>	Length of record

Returns

success or error code. Values can be parsed with [device_getIDGStatusCodeString\(\)](#)

See also

[ErrorCode](#)

11.17.4.45 device_getSDKWaitTime()

```
int device_getSDKWaitTime ( )
```

Get SDK Wait Time

Get the SDK wait time for transactions

Returns

SDK wait time in seconds

11.17.4.46 device_getThreadStackSize()

```
int device_getThreadStackSize ( )
```

Get Thread Stack Size

Get the stack size setting for newly created threads

Returns

Thread Stack Size

11.17.4.47 device_getTransactionResults()

```
int device_getTransactionResults (
    IDTMSRData * cardData )
```

Get Transaction Results Gets the transaction results when the reader is functioning in "Auto Poll" mode

Parameters

<i>cardData</i>	The transaction results
-----------------	-------------------------

Returns

success or error code. Values can be parsed with [device_getResponseCodeString](#)

See also

[ErrorCode](#)

11.17.4.48 device_init()

```
int device_init ( )
```

Initial the device by USB

It will detect the device and trying connect.

The connect status can be checked by [device_isConnected\(\)](#).

Note: after the function returns success, the function [device_setCurrentDevice\(\)](#) has to be called to set the device type.

Returns

RETURN_CODE: Values can be parsed with [device_getIDGStatusCodeString\(\)](#)

11.17.4.49 device_isAttached()

```
int device_isAttached (
    int deviceType )
```

Check if the device is attached to the USB port The function [device_init\(\)](#) must be called before this function.

Parameters

<i>deviceType,the</i>	device type of the USB device
-----------------------	-------------------------------

Returns

1 if the device is attached, or 0 if the device is not attached

11.17.4.50 device_isConnected()

```
int device_isConnected ( )
```

Check the device connected status

Returns

DEVICE_DISCONNECT=0, or DEVICE_CONNECTED = 1

11.17.4.51 device_listDirectory()

```
int device_listDirectory (
    IN char * directoryName,
    IN int directoryNameLen,
    IN int recursive,
    IN int onSD,
    OUT char * directory,
    IN_OUT int * directoryLen )
```

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>directoryNameLen</i>	Directory Name Length. If null, root directory is listed
<i>recursive</i>	Included sub-directories
<i>onSD</i>	0: use internal memory, 1: use SD card The returned directory information The returned directory information length

Returns

RETURN_CODE: Values can be parsed with [device_getIDGStatusCodeString\(\)](#)

11.17.4.52 device_pingDevice()

```
int device_pingDevice ( )
```

Ping Device

Pings the reader. If connected, returns success. Otherwise, returns timeout.

Returns

RETURN_CODE: Values can be parsed with [device_getIDGStatusCodeString\(\)](#)

11.17.4.53 device_registerCameraCallBk()

```
void device_registerCameraCallBk (
    pCMR_callback pCMRf )
```

To register the camera callback function to get the image data. (Pass NULL to disable the callback.)

11.17.4.54 device_registerCardStatusFrontSwitchCallBk()

```
void device_registerCardStatusFrontSwitchCallBk (
    pCSFS_callback pCSFSf )
```

To register the card status and front switch callback function to get status. (Pass NULL to disable the callback.)

11.17.4.55 device_SendDataCommandNEO()

```
int device_SendDataCommandNEO (
    IN int cmd,
    IN int subCmd,
    IN BYTE * data,
    IN int dataLen,
    OUT BYTE * response,
    IN_OUT int * respLen )
```

Send a Command to device

Sends a command to the device .

Parameters

<i>cmd</i>	buffer of command to execute.
<i>cmdLen,the</i>	length of the buffer cmd.
<i>data</i>	buffer of IDG command data.
<i>dataLen,the</i>	length of the buffer data.
<i>response</i>	Response data
<i>respLen,the</i>	length of Response data

Returns

RETURN_CODE: Values can be parsed with [device_getIDGStatusCodeString\(\)](#) Send a Command to NEO device

Sends a command to the NEO device .

Parameters

<i>cmd</i>	command to execute.
<i>subCmd,sub</i>	command to execute.
<i>data</i>	buffer of NEO command data.
<i>dataLen,the</i>	length of the buffer data.
<i>response</i>	Response data
<i>respLen,the</i>	length of Response data

Returns

RETURN_CODE: Values can be parsed with [device_getIDGStatusCodeString\(\)](#)

11.17.4.56 device_setCurrentDevice()

```
int device_setCurrentDevice (
    int deviceType )
```

Sets the current device to talk to

The connect status can be checked by [device_isConnected\(\)](#).

Parameters

<i>deviceType</i>	Device to connect to
	<pre>enum DEVICE_TYPE { IDT_DEVICE_UNKNOWN=0, IDT_DEVICE_AUGUSTA_HID, IDT_DEVICE_AUGUSTA_KB, IDT_DEVICE_AUGUSTA_S_HID, IDT_DEVICE_AUGUSTA_S_KB, IDT_DEVICE_AUGUSTA_S_TTK_HID, IDT_DEVICE_SPECTRUM_PRO, IDT_DEVICE_MINISMART_II, IDT_DEVICE_L80, IDT_DEVICE_L100, IDT_DEVICE_UNIPAY, IDT_DEVICE_UNIPAY_I_V, IDT_DEVICE_VP3300_AJ, IDT_DEVICE_KIOSK_III, IDT_DEVICE_KIOSK_III_S, IDT_DEVICE_PIP_READER, IDT_DEVICE_VENDI, IDT_DEVICE_VP3300_USB, IDT_DEVICE_UNIPAY_I_V_TTK, IDT_DEVICE_VP3300_BT, IDT_DEVICE_VP8800, IDT_DEVICE_SREDKEY2_HID, IDT_DEVICE_SREDKEY2_KB, IDT_DEVICE_NEO2, IDT_DEVICE_MINISMART_II_COM = IDT_DEVICE_NEO2+5, IDT_DEVICE_SPECTRUM_PRO_COM, IDT_DEVICE_KIOSK_III_COM, IDT_DEVICE_KIOSK_III_S_COM, IDT_DEVICE_PIP_READER_COM, IDT_DEVICE_VP3300_COM, IDT_DEVICE_NEO2_COM, IDT_DEVICE_MAX_DEVICES = IDT_DEVICE_NEO2_COM+5 };</pre>

Returns

RETURN_CODE: 1: success, 0: failed

11.17.4.57 device_setMerchantRecord()

```
int device_setMerchantRecord (
    int index,
    int enabled,
    char * merchantID,
    char * merchantURL )
```

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

Returns

RETURN_CODE: Values can be parsed with [device_getIDGStatusCodeString\(\)](#)

11.17.4.58 device_setSDKWaitTime()

```
void device_setSDKWaitTime (
    int waitTime )
```

Set SDK Wait Time

Set the SDK wait time for transactions

Parameters

<i>waitTime</i>	The SDK wait time for transaction in seconds. The Maximum is 2147483 seconds.
-----------------	---

11.17.4.59 device_setThreadStackSize()

```
void device_setThreadStackSize (
    int threadSize )
```

Set Thread Stack Size

Set the stack size setting for newly created threads

11.17.4.60 device_setTransactionExponent()

```
void device_setTransactionExponent (
    int exponent )
```

Sets the transaction exponent to be used with device_startTransaction. Default value is 2

Parameters

<i>exponent, The</i>	exponent to use when calling device_startTransaction
----------------------	--

11.17.4.61 device_startTransaction()

```
int device_startTransaction (
    IN double amount,
    IN double amtOther,
    IN int type,
    IN const int _timeout,
    IN BYTE * tags,
    IN int tagsLen )
```

Start Transaction Request

Authorizes the transaction for an MSR/CTLS/ICC card

The tags will be returned in the callback routine.

Parameters

<i>amount</i>	Transaction amount value (tag value 9F02) - SEE IMPORTANT NOTE BELOW
<i>amtOther</i>	Other amount value, if any (tag value 9F03) - SEE IMPORTANT NOTE BELOW
<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. Example, tag 9F0C with amount 0x000000000100 would be 0x9F0C06000000000100 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>tagsLen</i>	The length of tags data buffer.

>>>>>IMPORTANT: parameters for amount and amtOther MUST BE PASSED AS A DOUBLE VALUE WITH DECIMAL POINT. Example, do not pass 1, but instead pass 1.0 or 1.00. Otherwise, results will be unpredictable

Returns

RETURN_CODE: Values can be parsed with [device_getIDGStatusCodeString\(\)](#) Note: if auto poll is on, it will return the error IDG_P2_STATUS_CODE_COMMAND_NOT_ALLOWED

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 `device_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

11.17.4.62 device_transferFile()

```
int device_transferFile (
    IN char * fileName,
    IN int fileNameLen,
    IN BYTE * file,
    IN int fileLen )
```

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>filenameLen</i>	File Name Length.
<i>file</i>	The data file.
<i>fileLen</i>	File Length.

Returns

RETURN_CODE: Values can be parsed with [device_getIDGStatusCodeString\(\)](#)

11.17.4.63 emv_activateTransaction()

```
int emv_activateTransaction (
    IN int timeout,
    IN BYTE * tags,
    IN int tagsLen,
    IN int forceOnline )
```

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>	Tags to be included in the request. Passed as a TLV stream. Example, tag 9F0C with amount 0x000000000100 would be 0x9F0C06000000000100
<i>tagsLen</i>	Length of tags
<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 = DFEE1A079F029F36959F37

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString >>>>IMPORTANT: parameters for amount and amtOther MUST BE PASSED AS A DOUBLE VALUE WITH DECIMAL POINT. Example, do not pass 1, but instead pass 1.0 or 1.00. Otherwise, results will be unpredictable

11.17.4.64 emv_allowFallback()

```
void emv_allowFallback (
    IN int allow )
```

Allow fallback for EMV transactions. Default is TRUE

Parameters

<i>allow</i>	TRUE = allow fallback, FALSE = don't allow fallback
--------------	---

11.17.4.65 emv_authenticateTransaction()

```
int emv_authenticateTransaction (
    IN BYTE * updatedTLV,
    IN int updatedTLVLen )
```

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 = DFEE1A079F029F36959F37
<i>updatedTLVLen</i>	

Returns

RETURN_CODE: Values can be parsed with device_getResponseCodeString

11.17.4.66 emv_authenticateTransactionWithTimeout()

```
int emv_authenticateTransactionWithTimeout (
```

```

    IN int timeout,
    IN BYTE * updatedTLV,
    IN int updatedTLVLen )

```

Authenticate EMV Transaction Request with Timeout

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>timeout</i>	Timeout value in seconds.
<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 = DFEE1A079F029F36959F37
<i>updatedTLVLen</i>	

Returns

RETURN_CODE: Values can be parsed with `device_getResponseCodeString`

11.17.4.67 `emv_cancelTransaction()`

```
int emv_cancelTransaction ( )
```

Cancel EMV Transaction

Cancels the currently executing EMV transaction.

Returns

RETURN_CODE: Values can be parsed with [device_getResponseCodeString\(\)](#)

11.17.4.68 `emv_completeTransaction()`

```

int emv_completeTransaction (
    IN int commError,
    IN BYTE * authCode,
    IN int authCodeLen,
    IN BYTE * iad,
    IN int iadLen,
    IN BYTE * tlvScripts,
    IN int tlvScriptsLen,
    IN BYTE * tlv,
    IN int tlvLen )

```

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(1) if host was unreachable, or FALSE(0) 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>authCodeLen</i>	the length of <i>authCode</i>
<i>iad</i>	Issuer Authentication Data, if any. Example 0x11223344556677883030 (tag value 91).
<i>iadLen</i>	the length of <i>iad</i>
<i>tlvScripts</i>	71/72 scripts, if any
<i>tlvScriptsLen</i>	the length of <i>tlvScripts</i>
<i>tlv</i>	Additional TLV data to return with transaction results (if any)
<i>tlvLen</i>	the length of <i>tlv</i>

Returns

RETURN_CODE: Values can be parsed with `device_getResponseCodeString`

11.17.4.69 `emv_getAutoAuthenticateTransaction()`

```
int emv_getAutoAuthenticateTransaction ( )
```

Gets auto authenticate value for EMV transactions.

Returns

RETURN_CODE: TRUE = auto authenticate, FALSE = manually authenticate

11.17.4.70 `emv_getAutoCompleteTransaction()`

```
int emv_getAutoCompleteTransaction ( )
```

Gets auto complete value for EMV transactions.

Returns

RETURN_CODE: TRUE = auto complete, FALSE = manually complete

11.17.4.71 `emv_getEMVConfigurationCheckValue()`

```
int emv_getEMVConfigurationCheckValue (
    OUT BYTE * checkValue,
    IN_OUT int * checkValueLen )
```

Get EMV Kernel configuration check value info

Parameters

<i>checkValue</i>	Response returned of Kernel configuration check value info
<i>checkValueLen</i>	the length of checkValue

Returns

RETURN_CODE: Values can be parsed with [device_getIDGStatusCodeString\(\)](#)

11.17.4.72 emv_getEMVKernelCheckValue()

```
int emv_getEMVKernelCheckValue (
    OUT BYTE * checkValue,
    IN_OUT int * checkValueLen )
```

Get EMV Kernel check value info

Parameters

<i>checkValue</i>	Response returned of Kernel check value info
<i>checkValueLen</i>	the length of checkValue

Returns

RETURN_CODE: Values can be parsed with [device_getIDGStatusCodeString\(\)](#)

11.17.4.73 emv_getEMVKernelVersion()

```
int emv_getEMVKernelVersion (
    OUT char * version )
```

DEPRECATED : please use [emv_getEMVKernelVersion_Len\(\)](#)(OUT char* version, IN_OUT int *versionLen)

Polls device for EMV Kernel Version

Parameters

<i>version</i>	Response returned of Kernel Version; needs to have at least 128 bytes of memory.
----------------	--

Returns

RETURN_CODE: Values can be parsed with [device_getIDGStatusCodeString\(\)](#)

11.17.4.74 emv_getEMVKernelVersion_Len()

```
int emv_getEMVKernelVersion_Len (
    OUT char * version,
    IN_OUT int * versionLen )
```


Polls device for EMV Kernel Version

Parameters

<i>version</i>	Response returned of Kernel Version
<i>versionLen</i>	Length of version

Returns

RETURN_CODE: Values can be parsed with [device_getIDGStatusCodeString\(\)](#)

11.17.4.75 emv_registerCallBk()

```
void emv_registerCallBk (
    pEMV_callback pEMVf )
```

To register the emv callback function to get the EMV processing response. (Pass NULL to disable the callback.)

11.17.4.76 emv_removeAllApplicationData()

```
int emv_removeAllApplicationData ( )
```

Remove All Application Data

Removes all the Application Data

Returns

RETURN_CODE: Values can be parsed with [device_getIDGStatusCodeString\(\)](#)

11.17.4.77 emv_removeAllCAPK()

```
int emv_removeAllCAPK ( )
```

Remove All Certificate Authority Public Key

Removes all the CAPK

Returns

RETURN_CODE: Values can be parsed with [device_getIDGStatusCodeString\(\)](#)

11.17.4.78 emv_removeAllCRL()

```
int emv_removeAllCRL ( )
```

Remove All Certificate Revocation List Entries

Removes all CRLentry entries

Returns

RETURN_CODE: Values can be parsed with [device_getResponseCodeString\(\)](#)

11.17.4.79 emv_removeAllExceptions()

```
int emv_removeAllExceptions ( )
```

Remove All EMV Exceptions

Removes all entries from the EMV Exception List

Returns

RETURN_CODE: Values can be parsed with [device_getIDGStatusCodeString\(\)](#)

11.17.4.80 emv_removeApplicationData()

```
int emv_removeApplicationData (
    IN BYTE * AID,
    IN int AIDLen )
```

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>AIDLen</i>	the length of AID data buffer

Returns

RETURN_CODE: Values can be parsed with [device_getIDGStatusCodeString\(\)](#)

11.17.4.81 emv_removeCAPK()

```
int emv_removeCAPK (
    IN BYTE * capk,
    IN int capkLen )
```

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>capkLen</i>	the length of capk data buffer

Returns

RETURN_CODE: Values can be parsed with [device_getIDGStatusCodeString\(\)](#)

11.17.4.82 emv_removeCRL()

```
int emv_removeCRL (
    IN BYTE * list,
    IN int lsLen )
```

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>lsLen</i>	the length of list data buffer

Returns

RETURN_CODE: Values can be parsed with [device_getResponseCodeString\(\)](#)

11.17.4.83 emv_removeException()

```
int emv_removeException (
    IN BYTE * exception,
    IN int exceptionLen )
```

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)
<i>exceptionLen</i>	The length of the exception.

Returns

RETURN_CODE: Values can be parsed with [device_getIDGStatusCodeString\(\)](#)

11.17.4.84 emv_removeTransactionLog()

```
int emv_removeTransactionLog ( )
```

Clear Transaction Log

Clears the transaction log.

Returns

RETURN_CODE: Values can be parsed with [device_getIDGStatusCodeString\(\)](#)

11.17.4.85 emv_retrieveAIDList()

```
int emv_retrieveAIDList (
    OUT BYTE * AIDList,
    IN_OUT int * AIDListLen )
```

Retrieve AID list

Returns all the AID names installed on the terminal for CTLS. .

Parameters

<i>AIDList</i>	array of AID name byte arrays
<i>AIDListLen</i>	the length of AIDList array buffer

Returns

RETURN_CODE: Values can be parsed with [device_getIDGStatusCodeString\(\)](#)

11.17.4.86 emv_retrieveApplicationData()

```
int emv_retrieveApplicationData (
    IN BYTE * AID,
    IN int AIDLen,
    OUT BYTE * tlv,
    IN_OUT int * tlvLen )
```

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>AIDLen</i>	the length of AID data buffer.
<i>tlv</i>	The TLV elements of the requested AID
<i>tlvLen</i>	the length of tlv data buffer.

Returns

RETURN_CODE: Values can be parsed with [device_getIDGStatusCodeString\(\)](#)

11.17.4.87 emv_retrieveCAPK()

```
int emv_retrieveCAPK (
    IN BYTE * capk,
    IN int capkLen,
    OUT BYTE * key,
    IN_OUT int * keyLen )
```

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>capkLen</i>	the length of capk data buffer
<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>keyLen</i>	<p>the length of key data buffer</p> <ul style="list-style-type: none"> •

Returns

RETURN_CODE: Values can be parsed with [device_getIDGStatusCodeString\(\)](#)

11.17.4.88 emv_retrieveCAPKList()

```
int emv_retrieveCAPKList (
    OUT BYTE * keys,
    IN_OUT int * keysLen )
```

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>keysLen</i>	the length of keys data buffer

Returns

RETURN_CODE: Values can be parsed with [device_getIDGStatusCodeString\(\)](#)

11.17.4.89 emv_retrieveCRL()

```
int emv_retrieveCRL (
```

```

    OUT BYTE * list,
    IN_OUT int * lssLen )

```

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>lssLen</i>	the length of list data buffer

Returns

RETURN_CODE: Values can be parsed with [device_getResponseCodeString\(\)](#)

11.17.4.90 emv_retrieveExceptionList()

```

int emv_retrieveExceptionList (
    OUT BYTE * exceptionList,
    IN_OUT int * exceptionListLen )

```

Retrieve the EMV Exception List

Returns the EMV Exception entries on the terminal.

Parameters

<i>exceptionList</i>	[Exception1][Exception2]...[Exceptionnn], where [Exception] is 12 bytes: [1 byte Len][10 bytes PAN][1 byte Sequence Number]
<i>exceptionListLen</i>	The length of the exception list.

Returns

RETURN_CODE: Values can be parsed with [device_getIDGStatusCodeString\(\)](#)

11.17.4.91 emv_retrieveExceptionLogStatus()

```

int emv_retrieveExceptionLogStatus (
    OUT BYTE * exceptionLogStatus,
    IN_OUT int * exceptionLogStatusLen )

```

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>exceptionLogStatus</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>exceptionLogStatusLen</i>	The length of the exception log status.

Returns

RETURN_CODE: Values can be parsed with [device_getIDGStatusCodeString\(\)](#)

11.17.4.92 **emv_retrieveTerminalData()**

```
int emv_retrieveTerminalData (
    OUT BYTE * tlv,
    IN_OUT int * tlvLen )
```

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>tlvLen</i>	the length of tlv data buffer

Returns

RETURN_CODE: Values can be parsed with [device_getIDGStatusCodeString\(\)](#)

11.17.4.93 **emv_retrieveTransactionLog()**

```
int emv_retrieveTransactionLog (
    OUT BYTE * transactionLog,
    IN_OUT int * transactionLogLen,
    IN_OUT int * remainingTransactionLogLen )
```

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>transactionLog</i>	Transaction Record		
<i>transactionLogLen</i>	The length of the transaction log.		
<i>remainingTransactionLogLen</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, ONLI↔
NEADVICE = 2, REVERSAL = 3 |

| 4 | AppExpDate | BYTE [4] |

| 3 | AuthRespCode | BYTE [3] |

| 3 | MerchantCategoryCode | BYTE [3] |

| 16 | MerchantID | BYTE [16] |

| 2 | PosEntryMode | BYTE [2] |

| 9 | TermID | BYTE [9] |

| 3 | AIP | BYTE [3] |

| 3 | ATC | BYTE [3] |

| 33 | IssuerAppData | BYTE [33] |

| 6 | TVR | BYTE [6] |

| 3 | TSI | BYTE [3] |

| 11 | Pan | BYTE [11] |

| 2 | PanSQNCNum | BYTE [2] |

| 3 | TermCountryCode | BYTE [3] |

| 7 | TranAmount | BYTE [7] |

| 3 | TranCurCode | BYTE [3] |

| 4 | TranDate | BYTE [4] |

| 2 | TranType | BYTE [2] |

| 9 | IFDSerialNum | BYTE [9] |

| 12 | AcquirerID | BYTE [12] |

| 2 | CID | BYTE [2] |

| 9 | AppCryptogram | BYTE [9] |

| 5 | UnpNum | BYTE [5] |

| 7 | AmountAuth | BYTE [7] |

| 4 | AppEffDate | BYTE [4] |

| 4 | CVMResults | BYTE [4] |

| 129 | IssScriptResults | BYTE [129] |

| 4 | TermCap | BYTE [4] |

| 2 | TermType | BYTE [2] |

| 20 | Track2 | BYTE [20] |

| 4 | TranTime | BYTE [4] |

| 7 | AmountOther | BYTE [7] |

| 1 | Unused | BYTE [1] |

Returns

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

11.17.4.94 emv_retrieveTransactionLogStatus()

```
int emv_retrieveTransactionLogStatus (
    OUT BYTE * transactionLogStatus,
    IN_OUT int * transactionLogStatusLen )
```

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>transactionLogStatus</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>transactionLogStatusLen</i>	The length of the transaction log status.

Returns

RETURN_CODE: Values can be parsed with [device_getIDGStatusCodeString\(\)](#)

11.17.4.95 emv_setApplicationData()

```
int emv_setApplicationData (
    IN BYTE * name,
    IN int nameLen,
    IN BYTE * tlv,
    IN int tlvLen )
```

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>nameLen</i>	the length of name data buffer of Application name,
<i>tlv</i>	Application data in TLV format
<i>tlvLen</i>	the length of tlv data buffer

Returns

RETURN_CODE: Values can be parsed with [device_getResponseCodeString\(\)](#)

11.17.4.96 emv_setApplicationDataTLV()

```
int emv_setApplicationDataTLV (
    IN BYTE * tlv,
```

```
IN int tlvLen )
```

Set Application Data by TLV

Sets the Application Data as specified by the TLV data

Parameters

<i>tlv</i>	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) Example valid TLV, for Group #2, AID a0000000035010: "dfee2d01029f0607a0000000051010ffe10101ffe50110ffe30114ffe20106"
<i>tlvLen</i>	the length of tlv data buffer

Returns

RETURN_CODE: Values can be parsed with [device_getResponseCodeString\(\)](#)

11.17.4.97 emv_setAutoAuthenticateTransaction()

```
void emv_setAutoAuthenticateTransaction (
    IN int authenticate )
```

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

Parameters

<i>authenticate</i>	TRUE = auto authenticate, FALSE = manually authenticate
---------------------	---

11.17.4.98 emv_setAutoCompleteTransaction()

```
void emv_setAutoCompleteTransaction (
    IN int complete )
```

Enables complete for EMV transactions. If a `emv_authenticateTransaction` results in code 0x0004 (go online), then `emv_completeTransaction` can automatically execute if parameter is set to TRUE

Parameters

<i>complete</i>	TRUE = auto complete, FALSE = manually complete
-----------------	---

11.17.4.99 emv_setCAPK()

```
int emv_setCAPK (
    IN BYTE * capk,
    IN int capkLen )
```

Set Certificate Authority Public Key

Sets the CAPK as specified by the CAKey structure

Parameters

<i>capk</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>capkLen</i>	the length of capk data buffer

Returns

RETURN_CODE: Values can be parsed with [device_getIDGStatusCodeString\(\)](#)

11.17.4.100 **emv_setCRL()**

```
int emv_setCRL (
    IN BYTE * list,
    IN int lsLen )
```

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>lsLen</i>	the length of list data buffer

Returns

RETURN_CODE: Values can be parsed with [device_getResponseCodeString\(\)](#)

11.17.4.101 **emv_setException()**

```
int emv_setException (
    IN BYTE * exception,
    IN int exceptionLen )
```

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)
<i>exceptionLen</i>	The length of the exception.

Returns

RETURN_CODE: Values can be parsed with [device_getIDGStatusCodeString\(\)](#)

11.17.4.102 **emv_setTerminalData()**

```
int emv_setTerminalData (
    IN BYTE * tlv,
    IN int tlvLen )
```

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 file. The first byte is the Terminal Configuration: 0x01 for 1C, 0x02 for 2C, 0x03 for 3C, and 0x04 for 4C.
<i>tlvLen</i>	the length of tlv data buffer

Return values

<i>RETURN_CODE</i>	Values can be parsed with device_getIDGStatusCodeString()
--------------------	---

11.17.4.103 **emv_startTransaction()**

```
int emv_startTransaction (
    IN double amount,
    IN double amtOther,
    IN int exponent,
    IN int type,
    IN int timeout,
    IN BYTE * tags,
    IN int tagsLen,
    IN int forceOnline )
```

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) - SEE IMPORTANT NOTE BELOW
<i>amtOther</i>	Other amount value, if any (tag value 9F03) - SEE IMPORTANT NOTE BELOW
<i>exponent</i>	Number of characters after decimal 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 stream. Example, tag 9F0C with amount 0x000000000100 would be 0x9F0C06000000000100 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>tagsLen</i>	The length of tags
<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 = DFEE1A079F029F36959F37

Returns

RETURN_CODE: Values can be parsed with `device_getResponseCodeString`

>>>>IMPORTANT: parameters for amount and amtOther MUST BE PASSED AS A DOUBLE VALUE WITH DECIMAL POINT. Example, do not pass 1, but instead pass 1.0 or 1.00. Otherwise, results will be unpredictable

11.17.4.104 `lcd_addItemToList()`

```
int lcd_addItemToList (
    IN BYTE * listGraphicsID,
    IN char * itemName,
    IN char * itemID,
    IN int selected )
```

Adds an item to an existing list.

Custom Display Mode must be enabled for custom text.

Parameters

<i>listGraphicsID</i>	Existing list's graphics ID (4 byte array) that is provided during creation
<i>itemName</i>	Item name (Maximum: 127 characters)
<i>itemID</i>	Identifier for the item (Maximum: 31 characters)
<i>selected</i>	If the item should be selected

Returns

RETURN_CODE: Values can be parsed with `device_getIDGStatusCodeString()`

11.17.4.105 `lcd_cancelSlideShow()`

```
int lcd_cancelSlideShow (
    OUT BYTE * statusCode,
    IN_OUT int * statusCodeLen )
```

Cancel slide show Cancel the slide show currently running

Parameters

<i>statusCode</i>	If the return code is not Success (0), the kernel may return a four-byte Extended Status Code
<i>statusCodeLen</i>	the length of the Extended Status Code (should be 4 bytes)

Returns

RETURN_CODE: Values can be parsed with [device_getIDGStatusCodeString\(\)](#)

11.17.4.106 lcd_captureSignature()

```
int lcd_captureSignature (
    IN int timeout )
```

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
----------------	---

Returns

RETURN_CODE: Values can be parsed with [device_getIDGStatusCodeString\(\)](#)

11.17.4.107 lcd_clearDisplay()

```
int lcd_clearDisplay (
    IN BYTE control )
```

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>control</i>	for L80 and L100 only. 0:First Line 1:Second Line 2:Third Line 3:Fourth Line 0xFF: All Screen
----------------	---

Returns

RETURN_CODE: Values can be parsed with [device_getIDGStatusCodeString\(\)](#)

11.17.4.108 lcd_clearEventQueue()

```
int lcd_clearEventQueue ( )
```

Removes all entries from the event queue.

Returns

RETURN_CODE: Values can be parsed with [device_getIDGStatusCodeString\(\)](#)

11.17.4.109 lcd_createInputField()

```
int lcd_createInputField (
    IN BYTE * specs,
    IN int specsLen,
    OUT BYTE * graphicId )
```

DEPRECATED : please use [lcd_createInputField_Len\(IN BYTE *specs, IN int specsLen, OUT BYTE *graphicId, IN_OUT int *graphicIdLen\)](#)

Create an input field on the screen.

Parameters

<i>specs</i>	The specs of the input field: Length (bytes) Description
--------------	--

-----	-----
2 - 4	X coordinate in pixels, zero terminated ASCII
-----	-----
2 - 4	Y coordinate in pixels, zero terminated ASCII
-----	-----
2 - 4	Width in pixels, zero terminated ASCII. Set to 0 (30h) for calculated width.
-----	-----
2 - 4	Height in pixels, zero terminated ASCII. Set to 0 (30h) for calculated height.
-----	-----
2	Font designation. Default font = 1, zero terminated ASCII
-----	-----
2 - 3	Zero terminated ASCII Font ID
-----	-----
3	Zero terminated ASCII hexadecimal display option flag
Bit 0	0 = No Border
1	= Show Border
Bit 1	0 = Characters are first displayed on the leftmost area of the screen.
1	= The first character entered is displayed on the rightmost area of
	the screen, and, as further digits are entered, characters scroll
	from the right to the left.
Bit 2 - 15	Reserved
-----	-----
1 or 9	Foreground color, zero terminated ASCII hexadecimal
-----	-----
1 or 9	Background color, zero terminated ASCII hexadecimal

-----	-----
1 or 9	Border color, zero terminated ASCII hexadecimal
-----	-----
1 - 65	Prefill String, zero terminated ASCII
-----	-----
1 - 65	Format String, zero terminated ASCII

Parameters

<i>specsLen</i>	The length of specs
<i>graphicsID</i>	The graphicID of the event (required to be 4 bytes)

Returns

RETURN_CODE: Values can be parsed with [device_getIDGStatusCodeString\(\)](#)

11.17.4.110 lcd_createInputField_Len()

```
int lcd_createInputField_Len (
    IN BYTE * specs,
    IN int specsLen,
    OUT BYTE * graphicId,
    IN_OUT int * graphicIdLen )
```

Create an input field on the screen.

Parameters

<i>specs</i>	The specs of the input field: Length (bytes) Description
--------------	--

-----	-----
2 - 4	X coordinate in pixels, zero terminated ASCII
-----	-----
2 - 4	Y coordinate in pixels, zero terminated ASCII
-----	-----
2 - 4	Width in pixels, zero terminated ASCII. Set to 0 (30h) for calculated width.
-----	-----
2 - 4	Height in pixels, zero terminated ASCII. Set to 0 (30h) for calculated height.
-----	-----
2	Font designation. Default font = 1, zero terminated ASCII
-----	-----
2 - 3	Zero terminated ASCII Font ID
-----	-----
3	Zero terminated ASCII hexadecimal display option flag
	Bit 0 0 = No Border

```

| | 1 = Show Border |
| | Bit 1 0 = Characters are first displayed on the leftmost area of the screen. |
| | 1 = The first character entered is displayed on the rightmost area of |
| | the screen, and, as further digits are entered, characters scroll |
| | from the right to the left. |
| | Bit 2 - 15 Reserved |
| ----- | ----- |
| 1 or 9 | Foreground color, zero terminated ASCII hexadecimal |
| ----- | ----- |
| 1 or 9 | Background color, zero terminated ASCII hexadecimal |
| ----- | ----- |
| 1 or 9 | Border color, zero terminated ASCII hexadecimal |
| ----- | ----- |
| 1 - 65 | Prefill String, zero terminated ASCII |
| ----- | ----- |
| 1 - 65 | Format String, zero terminated ASCII |

```

Parameters

<i>specsLen</i>	The length of specs
<i>graphicsID</i>	The graphicID of the event (required to be 4 bytes)
<i>graphicsIDLen</i>	Length of graphicsID

Returns

RETURN_CODE: Values can be parsed with [device_getIDGStatusCodeString\(\)](#)

11.17.4.111 lcd_createList()

```

int lcd_createList (
    IN int posX,
    IN int posY,
    IN int numOfColumns,
    IN int numOfRows,
    IN int fontDesignation,
    IN int fontID,
    IN int verticalScrollArrowsVisible,
    IN int borderedListItems,
    IN int borderdScrollArrows,
    IN int touchSensitive,
    IN int automaticScrolling,
    OUT BYTE * graphicsID )

```

DEPRECATED : please use lcd_createList_Len(IN int posX, IN int posY, IN int numOfColumns, IN int numOfRows, IN int fontDesignation, IN int fontID, IN int verticalScrollArrowsVisible, IN int borderedListItems, IN int borderdScrollArrows, IN int touchSensitive, IN int automaticScrolling, OUT BYTE *graphicsID, IN_OUT int *graphicsIDLen)

Creates a display list.

Parameters

<i>posX</i>	X coordinate in pixels
<i>posY</i>	Y coordinate in pixels
<i>numOfColumns</i>	Number of columns to display
<i>numOfRows</i>	Number of rows to display
<i>fontDesignation</i>	Font Designation 1 - Default font
<i>fontID</i>	Font styling Font ID Height in pixels Font Properties

| ----- | ----- | ----- |

| 1 | 13 | Regular |

| 2 | 17 | Regular |

| 3 | 17 | Bold |

| 4 | 22 | Regular |

| 5 | 20 | Regular |

| 6 | 20 | Bold |

| 7 | 29 | Regular |

| 8 | 38 | Regular |

| 9 | 38 | Bold |

| 10 | 58 | Regular |

| 11 | 58 | Bold, mono-space |

| 12 | 14 | Regular, mono-space, 8 pixels wide |

| 13 | 15 | Regular, mono-space, 9 pixels wide |

| 14 | 17 | Regular, mono-space, 9 pixels wide |

| 15 | 20 | Regular, mono-space, 11 pixels wide |

| 16 | 21 | Regular, mono-space, 12 pixels wide |

| 17 | 25 | Regular, mono-space, 14 pixels wide |

| 18 | 30 | Regular, mono-space, 17 pixels wide |

Parameters

<i>verticalScrollArrowsVisible</i>	Display vertical scroll arrows by default
<i>borederedListItems</i>	Draw border around list items
<i>borederedScrollArrows</i>	Draw border around scroll arrows (if visible)
<i>touchSensitive</i>	List items are touch enabled
<i>automaticScrolling</i>	Enable automatic scrolling of list when new items exceed display area
<i>graphicsID</i>	A four byte array containing the ID of the created element (optional) if graphicsID is NULL, the SDK will not return graphicsID if graphicsID is not NULL, the SDK will return graphicsID, but it will need 4 bytes of memory

Returns

RETURN_CODE: Values can be parsed with [device_getIDGStatusCodeString\(\)](#)

11.17.4.112 lcd_createList_Len()

```

int lcd_createList_Len (
    IN int  posX,
    IN int  posY,
    IN int  numOfColumns,
    IN int  numOfRows,
    IN int  fontDesignation,
    IN int  fontID,
    IN int  verticalScrollArrowsVisible,
    IN int  borderedListItems,
    IN int  borderdScrollArrows,
    IN int  touchSensitive,
    IN int  automaticScrolling,
    OUT BYTE * graphicsID,
    IN_OUT int * graphicsIDLen )

```

Creates a display list.

Parameters

<i>posX</i>	X coordinate in pixels
<i>posY</i>	Y coordinate in pixels
<i>numOfColumns</i>	Number of columns to display
<i>numOfRows</i>	Number of rows to display
<i>fontDesignation</i>	Font Designation 1 - Default font
<i>fontID</i>	Font styling Font ID Height in pixels Font Properties

```

| ----- | ----- | ----- |
| 1 | 13 | Regular |
| 2 | 17 | Regular |
| 3 | 17 | Bold |
| 4 | 22 | Regular |
| 5 | 20 | Regular |
| 6 | 20 | Bold |
| 7 | 29 | Regular |
| 8 | 38 | Regular |
| 9 | 38 | Bold |
| 10 | 58 | Regular |
| 11 | 58 | Bold, mono-space |
| 12 | 14 | Regular, mono-space, 8 pixels wide |
| 13 | 15 | Regular, mono-space, 9 pixels wide |
| 14 | 17 | Regular, mono-space, 9 pixels wide |
| 15 | 20 | Regular, mono-space, 11 pixels wide |
| 16 | 21 | Regular, mono-space, 12 pixels wide |
| 17 | 25 | Regular, mono-space, 14 pixels wide |
| 18 | 30 | Regular, mono-space, 17 pixels wide |

```

Parameters

<i>verticalScrollArrowsVisible</i>	Display vertical scroll arrows by default
<i>borederedListItems</i>	Draw border around list items
<i>borederedScrollArrows</i>	Draw border around scroll arrows (if visible)
<i>touchSensitive</i>	List items are touch enabled
<i>automaticScrolling</i>	Enable automatic scrolling of list when new items exceed display area
<i>graphicsID</i>	A four byte array containing the ID of the created element (optional)
<i>graphicsIDLen</i>	Length of graphicsID (optional)

Returns

RETURN_CODE: Values can be parsed with [device_getIDGStatusCodeString\(\)](#)

11.17.4.113 `lcd_customDisplayMode()`

```
int lcd_customDisplayMode (
    IN int enable )
```

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
---------------	----------------------------------

Returns

RETURN_CODE: Values can be parsed with [device_getIDGStatusCodeString\(\)](#)

11.17.4.114 `lcd_displayButton()`

```
int lcd_displayButton (
    IN int posX,
    IN int posY,
    IN int buttonWidth,
    IN int buttonHeight,
    IN int fontDesignation,
    IN int fontID,
    IN int displayPosition,
    IN char * buttonLabel,
    IN int buttonTextColorR,
    IN int buttonTextColorG,
    IN int buttonTextColorB,
    IN int buttonBackgroundColorR,
    IN int buttonBackgroundColorG,
    IN int buttonBackgroundColorB,
    OUT BYTE * graphicsID )
```

DEPRECATED : please use `lcd_displayButton_Len(IN int posX, IN int posY, IN int buttonWidth, IN int buttonHeight, IN int fontDesignation, IN int fontID, IN int displayPosition, IN char *buttonLabel, IN int buttonTextColorR, IN int`

buttonTextColorG, IN int buttonTextColorB, IN int buttonBackgroundColorR, IN int buttonBackgroundColorG, IN int buttonBackgroundColorB, OUT BYTE *graphicsID, IN_OUT int *graphicsIDLen)

Displays an interactive button.

Parameters

<i>posX</i>	X coordinate in pixels
<i>posY</i>	Y coordinate in pixels
<i>buttonWidth</i>	Width of the button
<i>buttonHeight</i>	Height of the button
<i>fontDesignation</i>	Font designation 1 - Default
<i>FontID</i>	Font styling Font ID Height in pixels Font Properties

-----		-----		-----	
1		13		Regular	
2		17		Regular	
3		17		Bold	
4		22		Regular	
5		20		Regular	
6		20		Bold	
7		29		Regular	
8		38		Regular	
9		38		Bold	
10		58		Regular	
11		58		Bold, mono-space	
12		14		Regular, mono-space, 8 pixels wide	
13		15		Regular, mono-space, 9 pixels wide	
14		17		Regular, mono-space, 9 pixels wide	
15		20		Regular, mono-space, 11 pixels wide	
16		21		Regular, mono-space, 12 pixels wide	
17		25		Regular, mono-space, 14 pixels wide	
18		30		Regular, mono-space, 17 pixels wide	

Parameters

<i>displayPosition</i>	Button display position 0 - Center on line Y without clearing screen and without word wrap 1 - Center on line Y after clearing screen and without word wrap 2 - Display at (X, Y) without clearing screen and without word wrap 3 - Display at (X, Y) after clearing screen and without word wrap 4 - Center button on screen without clearing screen and without word wrap 5 - Center button on screen after clearing screen and without word wrap 64 - Center on line Y without clearing screen and with word wrap 65 - Center on line Y after clearing the screen and with word wrap 66 - Display at (X, Y) without clearing screen and with word wrap 67 - Display at (X, Y) after clearing screen and with word wrap 68 - Center button on screen without clearing screen and with word wrap 69 - Center button on screen after clearing screen and with word wrap
<i>buttonLabel</i>	Button label text (Maximum: 31 characters)
<i>buttonTextColorR</i>	- Red component for foreground color (0 - 255)

Parameters

<i>buttonTextColorG</i>	- Green component for foreground color (0 - 255)
<i>buttonTextColorB</i>	- Blue component for foreground color (0 - 255)
<i>buttonBackgroundColorR</i>	- Red component for background color (0 - 255)
<i>buttonBackgroundColorG</i>	- Green component for background color (0 - 255)
<i>buttonBackgroundColorB</i>	- Blue component for background color (0 - 255)
<i>graphicsID</i>	A four byte array containing the ID of the created element (optional) if graphicsID is NULL, the SDK will not return graphicsID if graphicsID is not NULL, the SDK will return graphicsID, but it will need 4 bytes of memory

Returns

RETURN_CODE: Values can be parsed with [device_getIDGStatusCodeString\(\)](#)

11.17.4.115 `lcd_displayButton_Len()`

```
int lcd_displayButton_Len (
    IN int posX,
    IN int posY,
    IN int buttonWidth,
    IN int buttonHeight,
    IN int fontDesignation,
    IN int fontID,
    IN int displayPosition,
    IN char * buttonLabel,
    IN int buttonTextColorR,
    IN int buttonTextColorG,
    IN int buttonTextColorB,
    IN int buttonBackgroundColorR,
    IN int buttonBackgroundColorG,
    IN int buttonBackgroundColorB,
    OUT BYTE * graphicsID,
    IN_OUT int * graphicsIDLen )
```

Displays an interactive button.

Parameters

<i>posX</i>	X coordinate in pixels
<i>posY</i>	Y coordinate in pixels
<i>buttonWidth</i>	Width of the button
<i>buttonHeight</i>	Height of the button
<i>fontDesignation</i>	Font designation 1 - Default
<i>FontID</i>	Font styling Font ID Height in pixels Font Properties

```
| ----- | ----- | ----- |
| 1 | 13 | Regular |
| 2 | 17 | Regular |
| 3 | 17 | Bold |
```


4	22	Regular
5	20	Regular
6	20	Bold
7	29	Regular
8	38	Regular
9	38	Bold
10	58	Regular
11	58	Bold, mono-space
12	14	Regular, mono-space, 8 pixels wide
13	15	Regular, mono-space, 9 pixels wide
14	17	Regular, mono-space, 9 pixels wide
15	20	Regular, mono-space, 11 pixels wide
16	21	Regular, mono-space, 12 pixels wide
17	25	Regular, mono-space, 14 pixels wide
18	30	Regular, mono-space, 17 pixels wide

Parameters

<i>displayPosition</i>	Button display position 0 - Center on line Y without clearing screen and without word wrap 1 - Center on line Y after clearing screen and without word wrap 2 - Display at (X, Y) without clearing screen and without word wrap 3 - Display at (X, Y) after clearing screen and without word wrap 4 - Center button on screen without clearing screen and without word wrap 5 - Center button on screen after clearing screen and without word wrap 64 - Center on line Y without clearing screen and with word wrap 65 - Center on line Y after clearing the screen and with word wrap 66 - Display at (X, Y) without clearing screen and with word wrap 67 - Display at (X, Y) after clearing screen and with word wrap 68 - Center button on screen without clearing screen and with word wrap 69 - Center button on screen after clearing screen and with word wrap
<i>buttonLabel</i>	Button label text (Maximum: 31 characters)
<i>buttonTextColorR</i>	- Red component for foreground color (0 - 255)
<i>buttonTextColorG</i>	- Green component for foreground color (0 - 255)
<i>buttonTextColorB</i>	- Blue component for foreground color (0 - 255)
<i>buttonBackgroundColorR</i>	- Red component for background color (0 - 255)
<i>buttonBackgroundColorG</i>	- Green component for background color (0 - 255)
<i>buttonBackgroundColorB</i>	- Blue component for background color (0 - 255)
<i>graphicsID</i>	A four byte array containing the ID of the created element (optional)
<i>graphicsIDLen</i>	Length of graphicsID (optional)

Returns

RETURN_CODE: Values can be parsed with [device_getIDGStatusCodeString\(\)](#)

11.17.4.116 lcd_displayParagraph()

```
int lcd_displayParagraph (
    IN int posX,
```

```

IN int posY,
IN int displayWidth,
IN int displayHeight,
IN int fontDesignation,
IN int fontID,
IN int displayProperties,
IN char * displayText )

```

Displays text with scroll feature.

Custom Display Mode must be enabled.

Parameters

<i>posX</i>	X coordinate in pixels
<i>posY</i>	Y coordinate in pixels
<i>displayWidth</i>	Width of the display area in pixels (Minimum: 40px) 0 or NULL - Use the full width to display text
<i>displayHeight</i>	Height of the display area in pixels (Minimum: 100px) 0 or NULL - Use the full height to display text
<i>fontDesignation</i>	Font designation 1 - Default
<i>fontID</i>	Font styling Font ID Height in pixels Font Properties

1	13	Regular
2	17	Regular
3	17	Bold
4	22	Regular
5	20	Regular
6	20	Bold
7	29	Regular
8	38	Regular
9	38	Bold
10	58	Regular
11	58	Bold, mono-space
12	14	Regular, mono-space, 8 pixels wide
13	15	Regular, mono-space, 9 pixels wide
14	17	Regular, mono-space, 9 pixels wide
15	20	Regular, mono-space, 11 pixels wide
16	21	Regular, mono-space, 12 pixels wide
17	25	Regular, mono-space, 14 pixels wide
18	30	Regular, mono-space, 17 pixels wide

Parameters

<i>displayProperties</i>	Display properties for the text 0 - Center on line Y without clearing screen 1 - Center on line Y after clearing screen 2 - Display at (X, Y) without clearing screen 3 - Display at (X, Y) after clearing screen 4 - Center on screen without clearing screen 5 - Center on screen after clearing screen
<i>displayText</i>	Display text (Maximum: 3999 characters plus terminator)

11.17.4.117 lcd_displayText()

```
int lcd_displayText (
    IN int posX,
    IN int posY,
    IN int displayWidth,
    IN int displayHeight,
    IN int fontDesignation,
    IN int fontID,
    IN int screenPosition,
    IN char * displayText,
    OUT BYTE * graphicsID )
```

DEPRECATED : please use lcd_displayText_Len(IN int posX, IN int posY, IN int displayWidth, IN int displayHeight, IN int fontDesignation, IN int fontID, IN int screenPosition, IN char *displayText, OUT BYTE *graphicsID, IN_OUT int *graphicsIDLen)

Displays text.

Custom Display Mode must be enabled for custom text. PIN pad entry is not allowed in Custom Display Mode but the Cancel, OK, and Clear keys remain active.

Parameters

<i>posX</i>	X coordinate in pixels
<i>posY</i>	Y coordinate in pixels
<i>displayWidth</i>	Width of the display area in pixels (optional)
<i>displayHeight</i>	Height of the display area in pixels (optional)
<i>fontDesignation</i>	Font designation 1 - Default
<i>FontID</i>	Font styling Font ID Height in pixels Font Properties

```
| ----- | ----- | ----- |
| 1 | 13 | Regular |
| 2 | 17 | Regular |
| 3 | 17 | Bold |
| 4 | 22 | Regular |
| 5 | 20 | Regular |
| 6 | 20 | Bold |
| 7 | 29 | Regular |
| 8 | 38 | Regular |
| 9 | 38 | Bold |
| 10 | 58 | Regular |
| 11 | 58 | Bold, mono-space |
| 12 | 14 | Regular, mono-space, 8 pixels wide |
| 13 | 15 | Regular, mono-space, 9 pixels wide |
| 14 | 17 | Regular, mono-space, 9 pixels wide |
| 15 | 20 | Regular, mono-space, 11 pixels wide |
| 16 | 21 | Regular, mono-space, 12 pixels wide |
```

| 17 | 25 | Regular, mono-space, 14 pixels wide |

| 18 | 30 | Regular, mono-space, 17 pixels wide |

Parameters

<i>screenPosition</i>	Display position 0 - Center on line Y without clearing screen 1 - Center on line Y after clearing screen 2 - Display at (X, Y) without clearing screen 3 - Display at (X, Y) after clearing screen 4 - Display at center of screen without clearing screen 5 - Display at center of screen after clearing screen 6 - Display text right-justified without clearing screen 7 - Display text right-justified after clearing screen
<i>displayText</i>	Display text (Maximum: 1900 characters)
<i>graphicsID</i>	A four byte array containing the ID of the created element (optional) if graphicsID is NULL, the SDK will not return graphicsID if graphicsID is not NULL, the SDK will return graphicsID, but it will need 4 bytes of memory

Returns

RETURN_CODE: Values can be parsed with [device_getIDGStatusCodeString\(\)](#)

11.17.4.118 lcd_displayText_Len()

```
int lcd_displayText_Len (
    IN int posX,
    IN int posY,
    IN int displayWidth,
    IN int displayHeight,
    IN int fontDesignation,
    IN int fontID,
    IN int screenPosition,
    IN char * displayText,
    OUT BYTE * graphicsID,
    IN_OUT int * graphicsIDLen )
```

Displays text.

Custom Display Mode must be enabled for custom text. PIN pad entry is not allowed in Custom Display Mode but the Cancel, OK, and Clear keys remain active.

Parameters

<i>posX</i>	X coordinate in pixels
<i>posY</i>	Y coordinate in pixels
<i>displayWidth</i>	Width of the display area in pixels (optional)
<i>displayHeight</i>	Height of the display area in pixels (optional)
<i>fontDesignation</i>	Font designation 1 - Default
<i>FontID</i>	Font styling Font ID Height in pixels Font Properties

| ----- | ----- | ----- |

| 1 | 13 | Regular |

| 2 | 17 | Regular |

| 3 | 17 | Bold |

4 22 Regular
5 20 Regular
6 20 Bold
7 29 Regular
8 38 Regular
9 38 Bold
10 58 Regular
11 58 Bold, mono-space
12 14 Regular, mono-space, 8 pixels wide
13 15 Regular, mono-space, 9 pixels wide
14 17 Regular, mono-space, 9 pixels wide
15 20 Regular, mono-space, 11 pixels wide
16 21 Regular, mono-space, 12 pixels wide
17 25 Regular, mono-space, 14 pixels wide
18 30 Regular, mono-space, 17 pixels wide

Parameters

<i>screenPosition</i>	Display position 0 - Center on line Y without clearing screen 1 - Center on line Y after clearing screen 2 - Display at (X, Y) without clearing screen 3 - Display at (X, Y) after clearing screen 4 - Display at center of screen without clearing screen 5 - Display at center of screen after clearing screen 6 - Display text right-justified without clearing screen 7 - Display text right-justified after clearing screen
<i>displayText</i>	Display text (Maximum: 1900 characters)
<i>graphicsID</i>	A four byte array containing the ID of the created element (optional)
<i>graphicsIDLen</i>	Length of graphicsID (optional)

Returns

RETURN_CODE: Values can be parsed with [device_getIDGStatusCodeString\(\)](#)

11.17.4.119 lcd_getInputEvent()

```
int lcd_getInputEvent (
    IN int timeout,
    OUT int * dataReceived,
    OUT BYTE * eventType,
    OUT BYTE * graphicsID,
    OUT BYTE * eventData )
```

DEPRECATED : please use lcd_getInputEvent_Len(IN int timeout, OUT int *dataReceived, OUT BYTE *eventType, IN_OUT int *eventTypeLen, OUT BYTE *graphicsID, IN_OUT int *graphicsIDLen, OUT BYTE *eventData, IN_OUT int *eventDataLen)

Requests input from the reader.

Parameters

<i>timeout</i>	Timeout amount in seconds 0 - No timeout
----------------	--

Parameters

<i>dataReceived</i>	Indicates if an event occurred and data was received 0 - No data received 1 - Data received
<i>eventType</i>	The event type (required to be at least 4 bytes), see table below
<i>graphicsID</i>	The graphicID of the event (required to be at least 4 bytes)
<i>eventData</i>	The event data, see table below (required to be at least 73 bytes) Event Type Value (4 bytes) Event Specific Data

```

| ----- | ----- | ----- |
| Button Event | 00030000h | Length = Variable |
| | | Byte 1: State (1 = Pressed, other values RFU) |
| | | Byte 2 - n: Null terminated caption |
| ----- | ----- | ----- |
| Checkbox Event | 00030001h | Length = 1 byte |
| | | Byte 1: State (1 = Checked, 0 = Unchecked) |
| ----- | ----- | ----- |
| Line Item Event | 00030002h | Length = 5 bytes |
| | | Byte 1: State (1 = Item Selected, other values RFU) |
| | | Byte 2 - n: Caption of the selected item |
| ----- | ----- | ----- |
| Keypad Event | 00030003h | Length - 3 bytes |
| | | Byte 1: State (1 = key pressed, 2 = key released, other values RFU) |
| | | Byte 2 - 3: Key pressed and Key release |
| | | 0030h - KEYPAD_KEY_0 |
| | | 0031h - KEYPAD_KEY_1 |
| | | 0032h - KEYPAD_KEY_2 |
| | | 0033h - KEYPAD_KEY_3 |
| | | 0034h - KEYPAD_KEY_4 |
| | | 0035h - KEYPAD_KEY_5 |
| | | 0036h - KEYPAD_KEY_6 |
| | | 0037h - KEYPAD_KEY_7 |
| | | 0038h - KEYPAD_KEY_8 |
| | | 0039h - KEYPAD_KEY_9 |
| | | Byte 2 - 3: Only Key pressed |
| | | 000Dh - KEYPAD_KEY_ENTER |
| | | 0008h - KEYPAD_KEY_CLEAR |
| | | 001Bh - KEYPAD_KEY_CANCEL |
| | | 0070h - FUNC_KEY_F1 (Vend III) |
| | | 0071h - FUNC_KEY_F2 (Vend III) |
| | | 0072h - FUNC_KEY_F3 (Vend III) |
| | | 0073h - FUNC_KEY_F4 (Vend III) |

```

```

| ----- | ----- | ----- |
| Touchscreen Event | 00030004h | Length = 1 - 33 bytes |
| | | Byte 1: State (not used) |
| | | Byte 2 - 33: Image name (zero terminated) |
| ----- | ----- | ----- |
| Slideshow Event | 00030005h | Length = 1 byte |
| | | Byte 1: State (not used) |
| ----- | ----- | ----- |
| Transaction Event | 00030006h | Length = 9 bytes |
| | | Byte 1: State (not used) |
| | | Byte 2 - 5: Card type (0 = unknown) |
| | | Byte 6 - 9: Status - A four byte, big-endian field |
| | | Byte 9 is used to store the 1-byte status code |
| | | 00 - SUCCESS |
| | | 08 - TIMEOUT |
| | | 0A - FAILED |
| | | This is not related to the extended status codes |
| ----- | ----- | ----- |
| Radio Button Event | 00030007h | Length = 73 bytes |
| | | Byte 1: State (1 = Change ins selected button, other values RFU) |
| | | Byte 2 - 33: Null terminated group name |
| | | Byte 34 - 65: Radio button caption |

```

Returns

RETURN_CODE: Values can be parsed with [device_getIDGStatusCodeString\(\)](#)

11.17.4.120 lcd_getInputEvent_Len()

```

int lcd_getInputEvent_Len (
    IN int timeout,
    OUT int * dataReceived,
    OUT BYTE * eventType,
    IN_OUT int * eventTypeLen,
    OUT BYTE * graphicsID,
    IN_OUT int * graphicsIDLen,
    OUT BYTE * eventData,
    IN_OUT int * eventDataLen )

```

Requests input from the reader.

Parameters

<i>timeout</i>	Timeout amount in seconds 0 - No timeout
<i>dataReceived</i>	Indicates if an event occurred and data was received 0 - No data received 1 - Data received
<i>eventType</i>	The event type (required to be at least 4 bytes), see table below

Parameters

<i>eventTypeLen</i>	Length of eventType
<i>graphicsID</i>	The graphicID of the event (required to be at least 4 bytes)
<i>graphicsIDLen</i>	length of graphicID
<i>eventData</i>	The event data, see table below (required to be at least 73 bytes) Event Type Value (4 bytes) Event Specific Data

```

| ----- | ----- | ----- |
| Button Event | 00030000h | Length = Variable |
| | | Byte 1: State (1 = Pressed, other values RFU) |
| | | Byte 2 - n: Null terminated caption |
| ----- | ----- | ----- |
| Checkbox Event | 00030001h | Length = 1 byte |
| | | Byte 1: State (1 = Checked, 0 = Unchecked) |
| ----- | ----- | ----- |
| Line Item Event | 00030002h | Length = 5 bytes |
| | | Byte 1: State (1 = Item Selected, other values RFU) |
| | | Byte 2 - n: Caption of the selected item |
| ----- | ----- | ----- |
| Keypad Event | 00030003h | Length - 3 bytes |
| | | Byte 1: State (1 = key pressed, 2 = key released, other values RFU) |
| | | Byte 2 - 3: Key pressed and Key release |
| | | 0030h - KEYPAD_KEY_0 |
| | | 0031h - KEYPAD_KEY_1 |
| | | 0032h - KEYPAD_KEY_2 |
| | | 0033h - KEYPAD_KEY_3 |
| | | 0034h - KEYPAD_KEY_4 |
| | | 0035h - KEYPAD_KEY_5 |
| | | 0036h - KEYPAD_KEY_6 |
| | | 0037h - KEYPAD_KEY_7 |
| | | 0038h - KEYPAD_KEY_8 |
| | | 0039h - KEYPAD_KEY_9 |
| | | Byte 2 - 3: Only Key pressed |
| | | 000Dh - KEYPAD_KEY_ENTER |
| | | 0008h - KEYPAD_KEY_CLEAR |
| | | 001Bh - KEYPAD_KEY_CANCEL |
| | | 0070h - FUNC_KEY_F1 (Vend III) |
| | | 0071h - FUNC_KEY_F2 (Vend III) |
| | | 0072h - FUNC_KEY_F3 (Vend III) |
| | | 0073h - FUNC_KEY_F4 (Vend III) |

```



```

| ----- | ----- | ----- |
| Touchscreen Event | 00030004h | Length = 1 - 33 bytes |
| | | Byte 1: State (not used) |
| | | Byte 2 - 33: Image name (zero terminated) |
| ----- | ----- | ----- |
| Slideshow Event | 00030005h | Length = 1 byte |
| | | Byte 1: State (not used) |
| ----- | ----- | ----- |
| Transaction Event | 00030006h | Length = 9 bytes |
| | | Byte 1: State (not used) |
| | | Byte 2 - 5: Card type (0 = unknown) |
| | | Byte 6 - 9: Status - A four byte, big-endian field |
| | | Byte 9 is used to store the 1-byte status code |
| | | 00 - SUCCESS |
| | | 08 - TIMEOUT |
| | | 0A - FAILED |
| | | This is not related to the extended status codes |
| ----- | ----- | ----- |
| Radio Button Event | 00030007h | Length = 73 bytes |
| | | Byte 1: State (1 = Change ins selected button, other values RFU) |
| | | Byte 2 - 33: Null terminated group name |
| | | Byte 34 - 65: Radio button caption |

```

Parameters

<i>eventDataLen</i>	Length of eventData
---------------------	---------------------

Returns

RETURN_CODE: Values can be parsed with [device_getIDGStatusCodeString\(\)](#)

11.17.4.121 lcd_getInputFieldValue()

```

int lcd_getInputFieldValue (
    IN BYTE * graphicId,
    OUT BYTE * retData,
    IN_OUT int * retDataLen )

```

Get the keypad data that was entered into the specified Input Field.

Parameters

<i>graphicsID</i>	The graphicID of the input field (required to be 4 bytes)
<i>retData</i>	return keypad data
<i>retDataLen</i>	The length of retData

Returns

RETURN_CODE: Values can be parsed with [device_getIDGStatusCodeString\(\)](#)

11.17.4.122 lcd_getSelectedListItem()

```
int lcd_getSelectedListItem (
    IN BYTE * listGraphicsID,
    OUT char * itemID )
```

DEPRECATED : please use [lcd_getSelectedListItem_Len\(IN BYTE *listGraphicsID, OUT char *itemID, IN_OUT int *itemIDLen\)](#)

Retrieves the selected item's ID.

Parameters

<i>listGraphicsID</i>	Existing list's graphics ID (4 byte array) that is provided during creation
<i>itemID</i>	The selected item's ID (Maximum: 32 characters) Need 33 bytes of memory including '\0'

11.17.4.123 lcd_getSelectedListItem_Len()

```
int lcd_getSelectedListItem_Len (
    IN BYTE * listGraphicsID,
    OUT char * itemID,
    IN_OUT int * itemIDLen )
```

Retrieves the selected item's ID.

Parameters

<i>listGraphicsID</i>	Existing list's graphics ID (4 byte array) that is provided during creation
<i>itemID</i>	The selected item's ID (Maximum: 32 characters) Need 33 bytes of memory including '\0'
<i>itemIDLen</i>	Length of itemID

11.17.4.124 lcd_resetInitialState()

```
int lcd_resetInitialState ( )
```

Reset to Initial State This command places the reader UI into the idle state and displays the appropriate idle display.

Returns

RETURN_CODE: Values can be parsed with [device_getIDGStatusCodeString\(\)](#)

11.17.4.125 lcd_setBackgroundImage()

```
int lcd_setBackgroundImage (
    IN char * file,
```

```

    IN int fileLen,
    IN int enable )

```

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)
<i>fileLen</i>	Length of files
<i>enable</i>	TRUE = Use Background Image, FALSE = Use Background Color

Returns

RETURN_CODE: Values can be parsed with [device_getIDGStatusCodeString\(\)](#)

11.17.4.126 lcd_setDisplayImage()

```

int lcd_setDisplayImage (
    IN char * file,
    IN int fileLen,
    IN int posX,
    IN int posY,
    IN int posMode,
    IN int touchEnable,
    IN int clearScreen )

```

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>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)
<i>fileLen</i>	Length of files
<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) • 2 - Center on screen
<i>touchEnable</i>	TRUE = Image is touch sensitive
<i>clearScreen</i>	TRUE = Clear screen

Returns

RETURN_CODE: Values can be parsed with [device_getIDGStatusCodeString\(\)](#)

11.17.4.127 lcd_setForeBackColor()

```
int lcd_setForeBackColor (
    IN BYTE * foreRGB,
    IN int foreRGBLen,
    IN BYTE * backRGB,
    IN int backRGBLen )
```

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>Length</i>	of foreRGB. Must be 3.
<i>backRGB</i>	Background RGB. 000000 = black, FF0000 = red, 00FF00 = green, 0000FF = blue, FFFFFFFF = white
<i>Length</i>	of backRGB. Must be 3.

Returns

RETURN_CODE: Values can be parsed with [device_getIDGStatusCodeString\(\)](#)

11.17.4.128 lcd_startSlideShow()

```
int lcd_startSlideShow (
    IN char * files,
    IN int filesLen,
    IN int posX,
    IN int posY,
    IN int posMode,
    IN int touchEnable,
    IN int recursion,
    IN int touchTerminate,
    IN int delay,
    IN int loops,
    IN int clearScreen )
```

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 dirctory are displayed
<i>filesLen</i>	Length of files
<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) • 2 - Center on screen
<i>touchEnable</i>	TRUE = Image is touch sensitive

Parameters

<i>recursion</i>	TRUE = Recursively follow directorys in list
<i>touchTerminate</i>	TRUE = Terminate slideshow on touch (if touch enabled)
<i>delay</i>	Number of seconds between image displays
<i>loops</i>	Number of display loops. A zero indicates continuous display
<i>clearScreen</i>	TRUE = Clear screen

Returns

RETURN_CODE: Values can be parsed with [device_getIDGStatusCodeString\(\)](#)

11.17.4.129 msr_cancelMSRSwipe()

```
int msr_cancelMSRSwipe ( )
```

Disable MSR Swipe Cancels MSR swipe request.

Returns

RETURN_CODE: Values can be parsed with [device_getIDGStatusCodeString\(\)](#)

11.17.4.130 msr_flushTrackData()

```
int msr_flushTrackData ( )
```

Flush Track Data Clears any track data being retained in memory by future PIN Block request.

Returns

RETURN_CODE: Values can be parsed with [device_getIDGStatusCodeString\(\)](#)

11.17.4.131 msr_registerCallBk()

```
void msr_registerCallBk (
    pMSR_callback pMSRf )
```

To register the msr callback function to get the MSR card data. (Pass NULL to disable the callback.)

11.17.4.132 msr_registerCallBkp()

```
void msr_registerCallBkp (
    pMSR_callbackp pMSRf )
```

To register the msr callback function to get the MSR card data pointer. (Pass NULL to disable the callback.)

11.17.4.133 msr_startMSRSwipe()

```
int msr_startMSRSwipe (
    IN int _timeout )
```

Start 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
----------------	---------------------

Returns

RETURN_CODE: Values can be parsed with [device_getIDGStatusCodeString\(\)](#) Note: if auto poll mode is on, it will return command not allowed error

11.17.4.134 parseMSRData()

```
void parseMSRData (
    IN BYTE * resData,
    IN int resLen,
    IN_OUT IDTMSRData * cardData )
```

Parser the MSR data from the buffer into IDTMSTData structure

Parameters

<i>resData</i>	MSR card data buffer
<i>resLen</i>	the length of resData
<i>cardData</i>	the parser result with IDTMSTData structure

11.17.4.135 pin_getEncryptedOnlinePIN()

```
int pin_getEncryptedOnlinePIN (
    IN int keyType,
    IN int timeout )
```

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

Returns

RETURN_CODE: Values can be parsed with [device_getResponseCodeString\(\)](#)

11.17.4.136 pin_getPAN()

```
int pin_getPAN (
    IN int getCSC,
    IN int timeout )
```

Get PAN

Requests PAN Entry on pinpad

Parameters

<i>getCSC</i>	Include Customer Service Code (also known as CVV, CVC)
<i>timeout</i>	PAN entry timeout

Returns

RETURN_CODE: Values can be parsed with [device_getResponseCodeString\(\)](#)

11.17.4.137 pin_promptCreditDebit()

```
int pin_promptCreditDebit (
    IN char * currencySymbol,
    IN int currencySymbolLen,
    IN char * displayAmount,
    IN int displayAmountLen,
    IN int timeout,
    OUT BYTE * retData,
    IN_OUT int * retDataLen )
```

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>currencySymbolLen</i>	length of currencySymbol
<i>displayAmount</i>	Amount to display (can be NULL)
<i>displayAmountLen</i>	length of displayAmount
<i>timeout</i>	Menu entry timeout. Valid values 2-20 seconds

Returns

RETURN_CODE: Values can be parsed with [device_getResponseCodeString\(\)](#)

11.17.4.138 pin_registerCallBk()

```
void pin_registerCallBk (
    pPIN_callBack pPINf )
```

To register the pin callback function to get the PINPad data. (Pass NULL to disable the callback.)

11.17.4.139 registerHotplugCallBk()

```
void registerHotplugCallBk (
    pMessageHotplug pMsgHotplug )
```

To register the USB HID hot-plug callback function which implemented in the application to monitor the hotplug message from the SDK.

11.17.4.140 registerLogCallBk()

```
void registerLogCallBk (
    pSendDataLog pFSend,
    pReadDataLog pFRead )
```

To register the log callback function which implemented in the application to monitor sending/reading data between application and reader.

11.17.4.141 SDK_Version()

```
char* SDK_Version ( )
```

To Get SDK version

Returns

return the SDK version string

11.17.4.142 setAbsoluteLibraryPath()

```
int setAbsoluteLibraryPath (
    const char * absoluteLibraryPath )
```

Set the path to use when searching for ID TECH's libraries. If this is not set, the libraries will be searched for with the system's default procedures.

Parameters

<i>absoluteLibraryPath</i>	The absolute path to ID TECH's libraries.
----------------------------	---

Returns

RETURN_CODE: Values can be parsed with [device_getIDGStatusCodeString\(\)](#)

11.17.4.143 ws_deleteSSLCert()

```
int ws_deleteSSLCert (
```



```
IN char * name,  
IN int nameLen )
```

Delete SSL Certificate Deletes a SSL Certificate by name

Parameters

<i>name</i>	Name of certificate to delete
<i>nameLen</i>	Certificate Name Length

Returns

RETURN_CODE: Values can be parsed with [device_getIDGStatusCodeString\(\)](#)

11.17.4.144 ws_getCertChainType()

```
int ws_getCertChainType (  
OUT int * type )
```

Get Certificate Chain Type Returns indicator for using test/production certificate chain

Parameters

<i>type</i>	0 = test certificate chain, 1 = production certificate chain
-------------	--

Returns

RETURN_CODE: Values can be parsed with [device_getIDGStatusCodeString\(\)](#)

11.17.4.145 ws_loadSSLCert()

```
int ws_loadSSLCert (  
IN char * name,  
IN int nameLen,  
IN char * dataDER,  
IN int dataDERLen )
```

Load SSL Certificate Loads a SSL certificate

Parameters

<i>name</i>	Certificate Name
<i>nameLen</i>	Certificate Name Length
<i>dataDER</i>	DER encoded certificate data
<i>dataDERLen</i>	DER encoded certificate data length

Returns

RETURN_CODE: Values can be parsed with [device_getIDGStatusCodeString\(\)](#)

11.17.4.146 ws_requestCSR()

```
int ws_requestCSR (
    OUT RequestCSR * csr )
```

Request CSR Requests 3 sets of public keys: encrypting Keys, signing/validating keys, signing/validating 3rd party apps

Parameters

<i>csr</i>	RequestCSR structure to return the data
------------	---

Returns

RETURN_CODE: Values can be parsed with [device_getIDGStatusCodeString\(\)](#)

11.17.4.147 ws_revokeSSLCert()

```
int ws_revokeSSLCert (
    IN char * name,
    IN int nameLen )
```

Revoke SSL Certificate Revokes a SSL Certificate by name

Parameters

<i>name</i>	Name of certificate to revoke
<i>nameLen</i>	Certificate Name Length

Returns

RETURN_CODE: Values can be parsed with [device_getIDGStatusCodeString\(\)](#)

11.17.4.148 ws_updateRootCertificate()

```
int ws_updateRootCertificate (
    IN char * name,
    IN int nameLen,
    IN char * dataDER,
    IN int dataDERLen,
    IN char * signature,
    IN int signatureLen )
```

Update Root Certificate Updates the root certificate

Parameters

<i>name</i>	Certificate Name
<i>nameLen</i>	Certificate Name Length
<i>dataDER</i>	DER encoded certificate data
<i>dataDERLen</i>	DER encoded certificate data length
<i>signature</i>	Future Root CA signed (RSASSA PSS SHA256) by current Root CA
<i>signature</i>	length

Returns

RETURN_CODE: Values can be parsed with [device_getIDGStatusCodeString\(\)](#)

Index

cancelWorldNet
 liblDT_Device.h, [115](#)
 liblDT_NEO2.h, [450](#)
 liblDT_VP3300_AJ.h, [724](#)
 liblDT_VP3300_BT.h, [776](#)
 liblDT_VP3300_COM.h, [828](#)
 liblDT_VP3300_USB.h, [880](#)
cancelWorldPay
 liblDT_Device.h, [115](#)
 liblDT_NEO2.h, [450](#)
 liblDT_VP3300_AJ.h, [724](#)
 liblDT_VP3300_BT.h, [776](#)
 liblDT_VP3300_COM.h, [828](#)
 liblDT_VP3300_USB.h, [880](#)
comm_registerHTTPCallback
 liblDT_MiniSmartII.h, [396](#)
 liblDT_NEO2.h, [451](#)
 liblDT_SREDKey2.h, [645](#)
 liblDT_UniPayI_V.h, [666](#)
 liblDT_VP3300_AJ.h, [724](#)
 liblDT_VP3300_BT.h, [776](#)
 liblDT_VP3300_COM.h, [828](#)
 liblDT_VP3300_USB.h, [880](#)
 liblDT_VP8800.h, [932](#)
 liblDT_Vendi.h, [695](#)
comm_registerV4Callback
 liblDT_MiniSmartII.h, [396](#)
 liblDT_NEO2.h, [451](#)
 liblDT_SREDKey2.h, [645](#)
 liblDT_UniPayI_V.h, [666](#)
 liblDT_VP3300_AJ.h, [725](#)
 liblDT_VP3300_BT.h, [776](#)
 liblDT_VP3300_COM.h, [828](#)
 liblDT_VP3300_USB.h, [880](#)
 liblDT_VP8800.h, [932](#)
 liblDT_Vendi.h, [695](#)
config_getBeeperController
 liblDT_Augusta.h, [49](#)
 liblDT_Device.h, [115](#)
 liblDT_MiniSmartII.h, [396](#)
config_getEncryptionControl
 liblDT_Augusta.h, [49](#)
 liblDT_Device.h, [116](#)
 liblDT_MiniSmartII.h, [396](#)
config_getLEDController
 liblDT_Augusta.h, [50](#)
 liblDT_Device.h, [116](#)
 liblDT_MiniSmartII.h, [397](#)
config_getModelNumber
 liblDT_Augusta.h, [50](#)
 liblDT_Device.h, [117](#)
 liblDT_L100.h, [329](#)
 liblDT_L80.h, [362](#)
 liblDT_MiniSmartII.h, [398](#)
 liblDT_NEO2.h, [451](#)
 liblDT_SREDKey2.h, [645](#)
 liblDT_SpectrumPro.h, [597](#)
config_getModelNumber_Len
 liblDT_Augusta.h, [51](#)
 liblDT_Device.h, [117](#)
 liblDT_L100.h, [329](#)
 liblDT_L80.h, [362](#)
 liblDT_MiniSmartII.h, [398](#)
 liblDT_NEO2.h, [451](#)
 liblDT_SREDKey2.h, [646](#)
 liblDT_SpectrumPro.h, [597](#)
config_getSerialNumber
 liblDT_Augusta.h, [51](#)
 liblDT_Device.h, [117](#)
 liblDT_KioskIII.h, [301](#)
 liblDT_L100.h, [329](#)
 liblDT_L80.h, [362](#)
 liblDT_MiniSmartII.h, [398](#)
 liblDT_NEO2.h, [452](#)
 liblDT_PipReader.h, [570](#)
 liblDT_SREDKey2.h, [646](#)
 liblDT_SpectrumPro.h, [598](#)
 liblDT_UniPayI_V.h, [666](#)
 liblDT_VP3300_AJ.h, [725](#)
 liblDT_VP3300_BT.h, [777](#)
 liblDT_VP3300_COM.h, [829](#)
 liblDT_VP3300_USB.h, [880](#)
 liblDT_VP8800.h, [932](#)
 liblDT_Vendi.h, [695](#)
config_getSerialNumber_Len
 liblDT_Augusta.h, [51](#)
 liblDT_Device.h, [118](#)
 liblDT_KioskIII.h, [301](#)
 liblDT_L100.h, [330](#)
 liblDT_L80.h, [363](#)
 liblDT_MiniSmartII.h, [399](#)
 liblDT_NEO2.h, [452](#)
 liblDT_PipReader.h, [571](#)
 liblDT_SREDKey2.h, [646](#)
 liblDT_SpectrumPro.h, [598](#)
 liblDT_UniPayI_V.h, [667](#)
 liblDT_VP3300_AJ.h, [725](#)
 liblDT_VP3300_BT.h, [777](#)

- libIDT_VP3300_COM.h, [829](#)
- libIDT_VP3300_USB.h, [881](#)
- libIDT_VP8800.h, [932](#)
- libIDT_Vendi.h, [695](#)
- config_setBeeperController
 - libIDT_Augusta.h, [52](#)
 - libIDT_Device.h, [118](#)
 - libIDT_MiniSmartII.h, [399](#)
- config_setCmdTimeOutDuration
 - libIDT_Device.h, [118](#)
 - libIDT_NEO2.h, [452](#)
- config_setConfigByJsonFile
 - libIDT_NEO2.h, [453](#)
- config_setEncryptionControl
 - libIDT_Augusta.h, [52](#)
 - libIDT_Device.h, [119](#)
 - libIDT_MiniSmartII.h, [399](#)
- config_setLEDController
 - libIDT_Augusta.h, [53](#)
 - libIDT_Device.h, [119](#)
 - libIDT_MiniSmartII.h, [400](#)
- ctls_activateTransaction
 - libIDT_Device.h, [120](#)
 - libIDT_KioskIII.h, [301](#)
 - libIDT_NEO2.h, [453](#)
 - libIDT_PipReader.h, [571](#)
 - libIDT_VP3300_AJ.h, [726](#)
 - libIDT_VP3300_BT.h, [777](#)
 - libIDT_VP3300_COM.h, [829](#)
 - libIDT_VP3300_USB.h, [881](#)
 - libIDT_VP8800.h, [933](#)
 - libIDT_Vendi.h, [696](#)
- ctls_cancelTransaction
 - libIDT_Device.h, [121](#)
 - libIDT_KioskIII.h, [303](#)
 - libIDT_NEO2.h, [454](#)
 - libIDT_PipReader.h, [572](#)
 - libIDT_VP3300_AJ.h, [727](#)
 - libIDT_VP3300_BT.h, [778](#)
 - libIDT_VP3300_COM.h, [830](#)
 - libIDT_VP3300_USB.h, [882](#)
 - libIDT_VP8800.h, [934](#)
 - libIDT_Vendi.h, [697](#)
- ctls_displayOnlineAuthResult
 - libIDT_Device.h, [121](#)
 - libIDT_NEO2.h, [455](#)
 - libIDT_VP8800.h, [934](#)
- ctls_getAllConfigurationGroups
 - libIDT_Device.h, [122](#)
 - libIDT_KioskIII.h, [303](#)
 - libIDT_NEO2.h, [455](#)
 - libIDT_PipReader.h, [572](#)
 - libIDT_VP3300_AJ.h, [727](#)
 - libIDT_VP3300_BT.h, [779](#)
 - libIDT_VP3300_COM.h, [831](#)
 - libIDT_VP3300_USB.h, [883](#)
 - libIDT_VP8800.h, [935](#)
 - libIDT_Vendi.h, [697](#)
- ctls_getConfigurationGroup
 - libIDT_Device.h, [122](#)
 - libIDT_KioskIII.h, [303](#)
 - libIDT_NEO2.h, [455](#)
 - libIDT_PipReader.h, [573](#)
 - libIDT_VP3300_AJ.h, [727](#)
 - libIDT_VP3300_BT.h, [779](#)
 - libIDT_VP3300_COM.h, [831](#)
 - libIDT_VP3300_USB.h, [883](#)
 - libIDT_VP8800.h, [935](#)
 - libIDT_Vendi.h, [698](#)
- ctls_registerCallBk
 - libIDT_KioskIII.h, [304](#)
 - libIDT_NEO2.h, [456](#)
 - libIDT_PipReader.h, [573](#)
 - libIDT_SREDKey2.h, [647](#)
 - libIDT_VP3300_AJ.h, [728](#)
 - libIDT_VP3300_BT.h, [779](#)
 - libIDT_VP3300_COM.h, [831](#)
 - libIDT_VP3300_USB.h, [883](#)
 - libIDT_VP8800.h, [935](#)
 - libIDT_Vendi.h, [698](#)
- ctls_registerCallBkp
 - libIDT_Device.h, [123](#)
 - libIDT_KioskIII.h, [304](#)
 - libIDT_NEO2.h, [456](#)
 - libIDT_PipReader.h, [573](#)
 - libIDT_SREDKey2.h, [647](#)
 - libIDT_VP3300_AJ.h, [728](#)
 - libIDT_VP3300_BT.h, [780](#)
 - libIDT_VP3300_COM.h, [832](#)
 - libIDT_VP3300_USB.h, [884](#)
 - libIDT_VP8800.h, [935](#)
 - libIDT_Vendi.h, [698](#)
- ctls_removeAllApplicationData
 - libIDT_Device.h, [123](#)
 - libIDT_KioskIII.h, [304](#)
 - libIDT_NEO2.h, [456](#)
 - libIDT_PipReader.h, [573](#)
 - libIDT_VP3300_AJ.h, [728](#)
 - libIDT_VP3300_BT.h, [780](#)
 - libIDT_VP3300_COM.h, [832](#)
 - libIDT_VP3300_USB.h, [884](#)
 - libIDT_VP8800.h, [936](#)
 - libIDT_Vendi.h, [698](#)
- ctls_removeAllCAPK
 - libIDT_Device.h, [123](#)
 - libIDT_KioskIII.h, [304](#)
 - libIDT_NEO2.h, [456](#)
 - libIDT_PipReader.h, [574](#)
 - libIDT_VP3300_AJ.h, [728](#)
 - libIDT_VP3300_BT.h, [780](#)
 - libIDT_VP3300_COM.h, [832](#)
 - libIDT_VP3300_USB.h, [884](#)
 - libIDT_VP8800.h, [936](#)
 - libIDT_Vendi.h, [698](#)
- ctls_removeApplicationData
 - libIDT_Device.h, [123](#)

- libIDT_KioskIII.h, 304
- libIDT_NEO2.h, 457
- libIDT_PipReader.h, 574
- libIDT_VP3300_AJ.h, 729
- libIDT_VP3300_BT.h, 780
- libIDT_VP3300_COM.h, 832
- libIDT_VP3300_USB.h, 884
- libIDT_VP8800.h, 936
- libIDT_Vendi.h, 699
- ctls_removeCAPK
 - libIDT_Device.h, 124
 - libIDT_KioskIII.h, 305
 - libIDT_NEO2.h, 457
 - libIDT_PipReader.h, 574
 - libIDT_VP3300_AJ.h, 729
 - libIDT_VP3300_BT.h, 781
 - libIDT_VP3300_COM.h, 833
 - libIDT_VP3300_USB.h, 885
 - libIDT_VP8800.h, 936
 - libIDT_Vendi.h, 699
- ctls_removeConfigurationGroup
 - libIDT_Device.h, 124
 - libIDT_KioskIII.h, 305
 - libIDT_NEO2.h, 457
 - libIDT_PipReader.h, 575
 - libIDT_VP3300_AJ.h, 729
 - libIDT_VP3300_BT.h, 781
 - libIDT_VP3300_COM.h, 833
 - libIDT_VP3300_USB.h, 885
 - libIDT_VP8800.h, 937
 - libIDT_Vendi.h, 699
- ctls_retrieveAIDList
 - libIDT_Device.h, 124
 - libIDT_KioskIII.h, 306
 - libIDT_NEO2.h, 458
 - libIDT_PipReader.h, 575
 - libIDT_VP3300_AJ.h, 730
 - libIDT_VP3300_BT.h, 781
 - libIDT_VP3300_COM.h, 833
 - libIDT_VP3300_USB.h, 885
 - libIDT_VP8800.h, 937
 - libIDT_Vendi.h, 700
- ctls_retrieveApplicationData
 - libIDT_Device.h, 125
 - libIDT_KioskIII.h, 306
 - libIDT_NEO2.h, 458
 - libIDT_PipReader.h, 575
 - libIDT_VP3300_AJ.h, 730
 - libIDT_VP3300_BT.h, 782
 - libIDT_VP3300_COM.h, 834
 - libIDT_VP3300_USB.h, 886
 - libIDT_VP8800.h, 938
 - libIDT_Vendi.h, 700
- ctls_retrieveCAPKList
 - libIDT_Device.h, 126
 - libIDT_KioskIII.h, 307
 - libIDT_NEO2.h, 459
 - libIDT_PipReader.h, 577
- libIDT_VP3300_AJ.h, 731
- libIDT_VP3300_BT.h, 783
- libIDT_VP3300_COM.h, 835
- libIDT_VP3300_USB.h, 887
- libIDT_VP8800.h, 939
- libIDT_Vendi.h, 701
- ctls_retrieveCAPK
 - libIDT_Device.h, 125
 - libIDT_KioskIII.h, 306
 - libIDT_NEO2.h, 458
 - libIDT_PipReader.h, 576
 - libIDT_VP3300_AJ.h, 730
 - libIDT_VP3300_BT.h, 782
 - libIDT_VP3300_COM.h, 834
 - libIDT_VP3300_USB.h, 886
 - libIDT_VP8800.h, 938
 - libIDT_Vendi.h, 701
- ctls_retrieveTerminalData
 - libIDT_Device.h, 126
 - libIDT_KioskIII.h, 308
 - libIDT_NEO2.h, 460
 - libIDT_PipReader.h, 577
 - libIDT_VP3300_AJ.h, 732
 - libIDT_VP3300_BT.h, 783
 - libIDT_VP3300_COM.h, 835
 - libIDT_VP3300_USB.h, 887
 - libIDT_VP8800.h, 939
 - libIDT_Vendi.h, 702
- ctls_setApplicationData
 - libIDT_Device.h, 127
 - libIDT_KioskIII.h, 308
 - libIDT_NEO2.h, 460
 - libIDT_PipReader.h, 577
 - libIDT_VP3300_AJ.h, 732
 - libIDT_VP3300_BT.h, 784
 - libIDT_VP3300_COM.h, 836
 - libIDT_VP3300_USB.h, 888
 - libIDT_VP8800.h, 940
 - libIDT_Vendi.h, 702
- ctls_setCAPK
 - libIDT_Device.h, 127
 - libIDT_KioskIII.h, 308
 - libIDT_NEO2.h, 460
 - libIDT_PipReader.h, 578
 - libIDT_VP3300_AJ.h, 732
 - libIDT_VP3300_BT.h, 784
 - libIDT_VP3300_COM.h, 836
 - libIDT_VP3300_USB.h, 888
 - libIDT_VP8800.h, 940
 - libIDT_Vendi.h, 703
- ctls_setConfigurationGroup
 - libIDT_Device.h, 128
 - libIDT_KioskIII.h, 309
 - libIDT_NEO2.h, 461
 - libIDT_PipReader.h, 578
 - libIDT_VP3300_AJ.h, 733
 - libIDT_VP3300_BT.h, 785
 - libIDT_VP3300_COM.h, 837

- libIDT_VP3300_USB.h, 889
- libIDT_VP8800.h, 941
- libIDT_Vendi.h, 703
- ctls_setTerminalData
 - libIDT_Device.h, 128
 - libIDT_KioskIII.h, 309
 - libIDT_NEO2.h, 461
 - libIDT_PipReader.h, 579
 - libIDT_VP3300_AJ.h, 733
 - libIDT_VP3300_BT.h, 785
 - libIDT_VP3300_COM.h, 837
 - libIDT_VP3300_USB.h, 889
 - libIDT_VP8800.h, 941
 - libIDT_Vendi.h, 704
- ctls_startTransaction
 - libIDT_Device.h, 129
 - libIDT_KioskIII.h, 310
 - libIDT_NEO2.h, 462
 - libIDT_PipReader.h, 579
 - libIDT_VP3300_AJ.h, 734
 - libIDT_VP3300_BT.h, 786
 - libIDT_VP3300_COM.h, 838
 - libIDT_VP3300_USB.h, 890
 - libIDT_VP8800.h, 942
 - libIDT_Vendi.h, 704
- device_SendDataCommand
 - libIDT_Augusta.h, 70
 - libIDT_Device.h, 175
 - libIDT_L100.h, 344
 - libIDT_L80.h, 378
 - libIDT_MiniSmartII.h, 416
 - libIDT_SREDKey2.h, 652
 - libIDT_SpectrumPro.h, 616
- device_SendDataCommandITP
 - libIDT_Device.h, 175
 - libIDT_SREDKey2.h, 653
- device_SendDataCommandNEO
 - libIDT_Device.h, 176
 - libIDT_KioskIII.h, 319
 - libIDT_NEO2.h, 498
 - libIDT_PipReader.h, 588
 - libIDT_SREDKey2.h, 653
 - libIDT_UniPayI_V.h, 673
 - libIDT_VP3300_AJ.h, 745
 - libIDT_VP3300_BT.h, 797
 - libIDT_VP3300_COM.h, 849
 - libIDT_VP3300_USB.h, 901
 - libIDT_VP8800.h, 956
 - libIDT_Vendi.h, 713
- device_activateTransaction
 - libIDT_Device.h, 130
 - libIDT_NEO2.h, 463
 - libIDT_VP3300_AJ.h, 735
 - libIDT_VP3300_BT.h, 787
 - libIDT_VP3300_COM.h, 839
 - libIDT_VP3300_USB.h, 891
 - libIDT_VP8800.h, 943
- device_buzzerOnOff
 - libIDT_Device.h, 132
 - libIDT_NEO2.h, 465
- device_calibrateParameters
 - libIDT_Device.h, 132
 - libIDT_VP8800.h, 945
- device_cancelTransaction
 - libIDT_Augusta.h, 53
 - libIDT_Device.h, 132
 - libIDT_MiniSmartII.h, 400
 - libIDT_NEO2.h, 465
 - libIDT_VP3300_AJ.h, 737
 - libIDT_VP3300_BT.h, 789
 - libIDT_VP3300_COM.h, 841
 - libIDT_VP3300_USB.h, 893
 - libIDT_VP8800.h, 945
- device_cancelTransactionSilent
 - libIDT_Device.h, 132
 - libIDT_NEO2.h, 465
- device_close
 - libIDT_Augusta.h, 53
 - libIDT_Device.h, 133
 - libIDT_KioskIII.h, 311
 - libIDT_L100.h, 330
 - libIDT_L80.h, 363
 - libIDT_MiniSmartII.h, 400
 - libIDT_NEO2.h, 466
 - libIDT_PipReader.h, 581
 - libIDT_SREDKey2.h, 647
 - libIDT_SpectrumPro.h, 598
 - libIDT_UniPayI_V.h, 667
 - libIDT_VP3300_AJ.h, 737
 - libIDT_VP3300_BT.h, 789
 - libIDT_VP3300_COM.h, 841
 - libIDT_VP3300_USB.h, 893
 - libIDT_VP8800.h, 945
 - libIDT_Vendi.h, 706
- device_configureButtons
 - libIDT_Device.h, 133
 - libIDT_NEO2.h, 466
- device_controlBeep
 - libIDT_Augusta.h, 54
 - libIDT_Device.h, 133
 - libIDT_MiniSmartII.h, 401
- device_controlIndicator
 - libIDT_Device.h, 134
 - libIDT_VP8800.h, 945
- device_controlLED_ICC
 - libIDT_Augusta.h, 55
 - libIDT_Device.h, 135
 - libIDT_MiniSmartII.h, 402
- device_controlLED_MSR
 - libIDT_Augusta.h, 55
 - libIDT_Device.h, 135
 - libIDT_MiniSmartII.h, 402
- device_controlLED
 - libIDT_Augusta.h, 54
 - libIDT_Device.h, 134
 - libIDT_MiniSmartII.h, 401

- device_controlUserInterface
 - liblDT_Device.h, [136](#)
 - liblDT_KioskIII.h, [312](#)
 - liblDT_NEO2.h, [466](#)
 - liblDT_PipReader.h, [581](#)
 - liblDT_VP3300_AJ.h, [737](#)
 - liblDT_VP3300_BT.h, [789](#)
 - liblDT_VP3300_COM.h, [841](#)
 - liblDT_VP3300_USB.h, [893](#)
 - liblDT_VP8800.h, [946](#)
 - liblDT_Vendi.h, [706](#)
- device_createDirectory
 - liblDT_Device.h, [137](#)
 - liblDT_VP8800.h, [947](#)
- device_deleteDirectory
 - liblDT_Device.h, [138](#)
 - liblDT_NEO2.h, [467](#)
 - liblDT_VP8800.h, [948](#)
- device_deleteFile
 - liblDT_Device.h, [138](#)
 - liblDT_NEO2.h, [468](#)
 - liblDT_VP8800.h, [948](#)
- device_disableBlueLED
 - liblDT_Device.h, [139](#)
 - liblDT_NEO2.h, [468](#)
- device_enableBlueLED
 - liblDT_Device.h, [139](#)
 - liblDT_NEO2.h, [468](#)
- device_enableExternalLCDMessages
 - liblDT_Device.h, [139](#)
 - liblDT_NEO2.h, [469](#)
- device_enableL100PassThrough
 - liblDT_Device.h, [140](#)
 - liblDT_NEO2.h, [469](#)
- device_enableL80PassThrough
 - liblDT_Device.h, [140](#)
 - liblDT_NEO2.h, [470](#)
- device_enablePassThrough
 - liblDT_Device.h, [140](#)
 - liblDT_KioskIII.h, [313](#)
 - liblDT_NEO2.h, [470](#)
 - liblDT_PipReader.h, [582](#)
 - liblDT_UniPayI_V.h, [667](#)
 - liblDT_VP3300_AJ.h, [738](#)
 - liblDT_VP3300_BT.h, [790](#)
 - liblDT_VP3300_COM.h, [842](#)
 - liblDT_VP3300_USB.h, [894](#)
 - liblDT_VP8800.h, [949](#)
 - liblDT_Vendi.h, [707](#)
- device_enableRFAntenna
 - liblDT_Device.h, [141](#)
 - liblDT_NEO2.h, [470](#)
- device_enhancedPassthrough
 - liblDT_Device.h, [141](#)
 - liblDT_VP8800.h, [949](#)
- device_enterStopMode
 - liblDT_Device.h, [142](#)
 - liblDT_L100.h, [330](#)
- liblDT_L80.h, [363](#)
- device_getAudioVolume
 - liblDT_Device.h, [142](#)
 - liblDT_NEO2.h, [471](#)
- device_getButtonConfiguration
 - liblDT_Device.h, [142](#)
 - liblDT_NEO2.h, [471](#)
- device_getCameraParameters
 - liblDT_Device.h, [143](#)
 - liblDT_NEO2.h, [471](#)
- device_getCurrentDeviceType
 - liblDT_Augusta.h, [56](#)
 - liblDT_Device.h, [143](#)
 - liblDT_KioskIII.h, [314](#)
 - liblDT_L100.h, [330](#)
 - liblDT_L80.h, [364](#)
 - liblDT_MiniSmartII.h, [403](#)
 - liblDT_NEO2.h, [472](#)
 - liblDT_PipReader.h, [583](#)
 - liblDT_SREDKey2.h, [647](#)
 - liblDT_SpectrumPro.h, [599](#)
 - liblDT_UniPayI_V.h, [667](#)
 - liblDT_VP3300_AJ.h, [739](#)
 - liblDT_VP3300_BT.h, [791](#)
 - liblDT_VP3300_COM.h, [843](#)
 - liblDT_VP3300_USB.h, [895](#)
 - liblDT_VP8800.h, [949](#)
 - liblDT_Vendi.h, [708](#)
- device_getDRS
 - liblDT_Augusta.h, [56](#)
 - liblDT_Device.h, [145](#)
 - liblDT_NEO2.h, [473](#)
- device_getDateTime
 - liblDT_Device.h, [143](#)
 - liblDT_L100.h, [331](#)
 - liblDT_L80.h, [364](#)
- device_getDateTime_Len
 - liblDT_Device.h, [143](#)
 - liblDT_L100.h, [331](#)
 - liblDT_L80.h, [364](#)
- device_getDeviceMemoryUsageInfo
 - liblDT_Device.h, [144](#)
 - liblDT_NEO2.h, [472](#)
- device_getDeviceTreeVersion
 - liblDT_Device.h, [145](#)
 - liblDT_NEO2.h, [472](#)
- device_getDriveFreeSpace
 - liblDT_Device.h, [145](#)
 - liblDT_VP8800.h, [950](#)
- device_getFirmwareVersion
 - liblDT_Augusta.h, [57](#)
 - liblDT_Device.h, [146](#)
 - liblDT_KioskIII.h, [314](#)
 - liblDT_L100.h, [331](#)
 - liblDT_L80.h, [364](#)
 - liblDT_MiniSmartII.h, [403](#)
 - liblDT_NEO2.h, [473](#)
 - liblDT_PipReader.h, [583](#)

- liblDT_SREDKey2.h, [647](#)
- liblDT_SpectrumPro.h, [599](#)
- liblDT_UniPayI_V.h, [668](#)
- liblDT_VP3300_AJ.h, [739](#)
- liblDT_VP3300_BT.h, [791](#)
- liblDT_VP3300_COM.h, [843](#)
- liblDT_VP3300_USB.h, [895](#)
- liblDT_VP8800.h, [950](#)
- liblDT_Vendi.h, [708](#)
- device_getFirmwareVersion_Len
 - liblDT_Augusta.h, [57](#)
 - liblDT_Device.h, [146](#)
 - liblDT_KioskIII.h, [314](#)
 - liblDT_L100.h, [332](#)
 - liblDT_L80.h, [365](#)
 - liblDT_MiniSmartII.h, [403](#)
 - liblDT_NEO2.h, [474](#)
 - liblDT_PipReader.h, [583](#)
 - liblDT_SREDKey2.h, [648](#)
 - liblDT_SpectrumPro.h, [599](#)
 - liblDT_UniPayI_V.h, [668](#)
 - liblDT_VP3300_AJ.h, [739](#)
 - liblDT_VP3300_BT.h, [791](#)
 - liblDT_VP3300_COM.h, [843](#)
 - liblDT_VP3300_USB.h, [895](#)
 - liblDT_VP8800.h, [950](#)
 - liblDT_Vendi.h, [708](#)
- device_getIDGStatusCodeString
 - liblDT_Device.h, [147](#)
 - liblDT_KioskIII.h, [315](#)
 - liblDT_NEO2.h, [474](#)
 - liblDT_PipReader.h, [584](#)
 - liblDT_SREDKey2.h, [648](#)
 - liblDT_UniPayI_V.h, [668](#)
 - liblDT_VP3300_AJ.h, [740](#)
 - liblDT_VP3300_BT.h, [792](#)
 - liblDT_VP3300_COM.h, [844](#)
 - liblDT_VP3300_USB.h, [896](#)
 - liblDT_VP8800.h, [951](#)
 - liblDT_Vendi.h, [709](#)
- device_getKeyStatus
 - liblDT_Augusta.h, [57](#)
 - liblDT_Device.h, [148](#)
 - liblDT_L100.h, [332](#)
 - liblDT_L80.h, [365](#)
 - liblDT_MiniSmartII.h, [404](#)
 - liblDT_NEO2.h, [476](#)
 - liblDT_SREDKey2.h, [649](#)
- device_getL100PassThroughMode
 - liblDT_Device.h, [149](#)
 - liblDT_NEO2.h, [477](#)
- device_getL80PassThroughMode
 - liblDT_Device.h, [149](#)
 - liblDT_NEO2.h, [477](#)
- device_getMerchantRecord
 - liblDT_Device.h, [150](#)
 - liblDT_KioskIII.h, [316](#)
 - liblDT_NEO2.h, [477](#)
- liblDT_PipReader.h, [585](#)
- liblDT_UniPayI_V.h, [670](#)
- liblDT_VP3300_AJ.h, [741](#)
- liblDT_VP3300_BT.h, [793](#)
- liblDT_VP3300_COM.h, [845](#)
- liblDT_VP3300_USB.h, [897](#)
- liblDT_VP8800.h, [952](#)
- liblDT_Vendi.h, [710](#)
- device_getMerchantRecord_Len
 - liblDT_Device.h, [150](#)
 - liblDT_KioskIII.h, [317](#)
 - liblDT_NEO2.h, [478](#)
 - liblDT_PipReader.h, [586](#)
 - liblDT_UniPayI_V.h, [670](#)
 - liblDT_VP3300_AJ.h, [742](#)
 - liblDT_VP3300_BT.h, [794](#)
 - liblDT_VP3300_COM.h, [846](#)
 - liblDT_VP3300_USB.h, [898](#)
 - liblDT_VP8800.h, [953](#)
 - liblDT_Vendi.h, [711](#)
- device_getNEOAltDevice
 - liblDT_Device.h, [151](#)
 - liblDT_NEO2.h, [478](#)
- device_getRTCDateTime
 - liblDT_Device.h, [161](#)
 - liblDT_VP3300_AJ.h, [742](#)
 - liblDT_VP3300_BT.h, [794](#)
 - liblDT_VP3300_COM.h, [846](#)
 - liblDT_VP3300_USB.h, [898](#)
- device_getResponseCodeString
 - liblDT_Augusta.h, [58](#)
 - liblDT_Device.h, [151](#)
 - liblDT_L100.h, [333](#)
 - liblDT_L80.h, [366](#)
 - liblDT_MiniSmartII.h, [405](#)
 - liblDT_NEO2.h, [478](#)
 - liblDT_SpectrumPro.h, [600](#)
- device_getSDKWaitTime
 - liblDT_Augusta.h, [68](#)
 - liblDT_Device.h, [161](#)
 - liblDT_KioskIII.h, [317](#)
 - liblDT_MiniSmartII.h, [415](#)
 - liblDT_NEO2.h, [488](#)
 - liblDT_PipReader.h, [586](#)
 - liblDT_SpectrumPro.h, [610](#)
 - liblDT_UniPayI_V.h, [671](#)
 - liblDT_VP3300_AJ.h, [743](#)
 - liblDT_VP3300_BT.h, [795](#)
 - liblDT_VP3300_COM.h, [847](#)
 - liblDT_VP3300_USB.h, [899](#)
 - liblDT_VP8800.h, [953](#)
 - liblDT_Vendi.h, [711](#)
- device_getSpectrumProKSN_Len
 - liblDT_Device.h, [162](#)
 - liblDT_SpectrumPro.h, [610](#)
- device_getSpectrumProKSN
 - liblDT_Device.h, [161](#)
 - liblDT_SpectrumPro.h, [610](#)

- device_getTamperStatus
 - libIDT_Device.h, [163](#)
 - libIDT_NEO2.h, [489](#)
- device_getThreadStackSize
 - libIDT_Augusta.h, [69](#)
 - libIDT_Device.h, [163](#)
 - libIDT_KioskIII.h, [317](#)
 - libIDT_MiniSmartII.h, [415](#)
 - libIDT_NEO2.h, [489](#)
 - libIDT_SpectrumPro.h, [611](#)
 - libIDT_UniPayI_V.h, [671](#)
 - libIDT_VP3300_AJ.h, [743](#)
 - libIDT_VP3300_BT.h, [795](#)
 - libIDT_VP3300_COM.h, [847](#)
 - libIDT_VP3300_USB.h, [899](#)
 - libIDT_VP8800.h, [953](#)
 - libIDT_Vendi.h, [711](#)
- device_getTransactionResults
 - libIDT_KioskIII.h, [318](#)
 - libIDT_NEO2.h, [489](#)
 - libIDT_PipReader.h, [586](#)
 - libIDT_VP3300_AJ.h, [743](#)
 - libIDT_VP3300_BT.h, [795](#)
 - libIDT_VP3300_COM.h, [847](#)
 - libIDT_VP3300_USB.h, [899](#)
 - libIDT_VP8800.h, [954](#)
 - libIDT_Vendi.h, [712](#)
- device_init
 - libIDT_Augusta.h, [69](#)
 - libIDT_Device.h, [163](#)
 - libIDT_KioskIII.h, [318](#)
 - libIDT_L100.h, [343](#)
 - libIDT_L80.h, [376](#)
 - libIDT_MiniSmartII.h, [415](#)
 - libIDT_NEO2.h, [490](#)
 - libIDT_PipReader.h, [587](#)
 - libIDT_SREDKey2.h, [650](#)
 - libIDT_SpectrumPro.h, [611](#)
 - libIDT_UniPayI_V.h, [671](#)
 - libIDT_VP3300_AJ.h, [743](#)
 - libIDT_VP3300_BT.h, [795](#)
 - libIDT_VP3300_COM.h, [847](#)
 - libIDT_VP3300_USB.h, [899](#)
 - libIDT_VP8800.h, [954](#)
 - libIDT_Vendi.h, [712](#)
- device_isAttached
 - libIDT_Augusta.h, [69](#)
 - libIDT_Device.h, [163](#)
 - libIDT_KioskIII.h, [318](#)
 - libIDT_L100.h, [343](#)
 - libIDT_L80.h, [376](#)
 - libIDT_MiniSmartII.h, [415](#)
 - libIDT_NEO2.h, [490](#)
 - libIDT_PipReader.h, [587](#)
 - libIDT_SREDKey2.h, [651](#)
 - libIDT_SpectrumPro.h, [611](#)
 - libIDT_UniPayI_V.h, [672](#)
 - libIDT_VP3300_AJ.h, [744](#)
- libIDT_VP3300_BT.h, [796](#)
- libIDT_VP3300_COM.h, [848](#)
- libIDT_VP3300_USB.h, [900](#)
- libIDT_VP8800.h, [954](#)
- libIDT_Vendi.h, [712](#)
- device_isConnected
 - libIDT_Augusta.h, [70](#)
 - libIDT_Device.h, [164](#)
 - libIDT_KioskIII.h, [319](#)
 - libIDT_L100.h, [343](#)
 - libIDT_L80.h, [377](#)
 - libIDT_MiniSmartII.h, [416](#)
 - libIDT_NEO2.h, [490](#)
 - libIDT_PipReader.h, [587](#)
 - libIDT_SREDKey2.h, [651](#)
 - libIDT_SpectrumPro.h, [612](#)
 - libIDT_UniPayI_V.h, [672](#)
 - libIDT_VP3300_AJ.h, [744](#)
 - libIDT_VP3300_BT.h, [796](#)
 - libIDT_VP3300_COM.h, [848](#)
 - libIDT_VP3300_USB.h, [900](#)
 - libIDT_VP8800.h, [955](#)
 - libIDT_Vendi.h, [713](#)
- device_lcdDisplayClear
 - libIDT_Device.h, [164](#)
 - libIDT_NEO2.h, [490](#)
- device_lcdDisplayLine1Message
 - libIDT_Device.h, [164](#)
 - libIDT_NEO2.h, [491](#)
- device_lcdDisplayLine2Message
 - libIDT_Device.h, [164](#)
 - libIDT_NEO2.h, [491](#)
- device_listDirectory
 - libIDT_Device.h, [165](#)
 - libIDT_NEO2.h, [491](#)
 - libIDT_VP8800.h, [955](#)
- device_loadCertCA
 - libIDT_Device.h, [165](#)
 - libIDT_NEO2.h, [492](#)
- device_outputLog
 - libIDT_Device.h, [166](#)
 - libIDT_NEO2.h, [492](#)
- device_pingDevice
 - libIDT_Device.h, [166](#)
 - libIDT_KioskIII.h, [319](#)
 - libIDT_NEO2.h, [493](#)
 - libIDT_PipReader.h, [588](#)
 - libIDT_SREDKey2.h, [651](#)
 - libIDT_UniPayI_V.h, [672](#)
 - libIDT_VP3300_AJ.h, [744](#)
 - libIDT_VP3300_BT.h, [796](#)
 - libIDT_VP3300_COM.h, [848](#)
 - libIDT_VP3300_USB.h, [900](#)
 - libIDT_VP8800.h, [955](#)
 - libIDT_Vendi.h, [713](#)
- device_playAudio
 - libIDT_Device.h, [166](#)
 - libIDT_NEO2.h, [493](#)

- device_pollCardReader
 - liblDT_Device.h, [167](#)
 - liblDT_SpectrumPro.h, [612](#)
- device_pollCardReader_Len
 - liblDT_Device.h, [168](#)
 - liblDT_SpectrumPro.h, [614](#)
- device_pollForToken
 - liblDT_Device.h, [170](#)
 - liblDT_NEO2.h, [493](#)
 - liblDT_VP3300_AJ.h, [744](#)
 - liblDT_VP3300_BT.h, [796](#)
 - liblDT_VP3300_COM.h, [848](#)
 - liblDT_VP3300_USB.h, [900](#)
- device_queryFile
 - liblDT_Device.h, [171](#)
 - liblDT_NEO2.h, [494](#)
- device_readFileFromSD
 - liblDT_Device.h, [171](#)
 - liblDT_NEO2.h, [494](#)
- device_rebootDevice
 - liblDT_Augusta.h, [70](#)
 - liblDT_Device.h, [172](#)
 - liblDT_L100.h, [344](#)
 - liblDT_L80.h, [377](#)
 - liblDT_MiniSmartII.h, [416](#)
 - liblDT_NEO2.h, [495](#)
 - liblDT_SREDKey2.h, [651](#)
 - liblDT_SpectrumPro.h, [616](#)
- device_registerCameraCallBk
 - liblDT_Augusta.h, [70](#)
 - liblDT_Device.h, [172](#)
 - liblDT_KioskIII.h, [319](#)
 - liblDT_L100.h, [344](#)
 - liblDT_L80.h, [377](#)
 - liblDT_MiniSmartII.h, [416](#)
 - liblDT_NEO2.h, [495](#)
 - liblDT_PipReader.h, [588](#)
 - liblDT_SREDKey2.h, [652](#)
 - liblDT_SpectrumPro.h, [616](#)
 - liblDT_UniPayI_V.h, [672](#)
 - liblDT_VP3300_AJ.h, [745](#)
 - liblDT_VP3300_BT.h, [797](#)
 - liblDT_VP3300_COM.h, [849](#)
 - liblDT_VP3300_USB.h, [901](#)
 - liblDT_VP8800.h, [955](#)
 - liblDT_Vendi.h, [713](#)
- device_registerCardStatusFrontSwitchCallBk
 - liblDT_Augusta.h, [70](#)
 - liblDT_Device.h, [172](#)
 - liblDT_KioskIII.h, [319](#)
 - liblDT_L100.h, [344](#)
 - liblDT_L80.h, [377](#)
 - liblDT_MiniSmartII.h, [416](#)
 - liblDT_NEO2.h, [495](#)
 - liblDT_PipReader.h, [588](#)
 - liblDT_SREDKey2.h, [652](#)
 - liblDT_SpectrumPro.h, [616](#)
 - liblDT_UniPayI_V.h, [673](#)
 - liblDT_VP3300_AJ.h, [745](#)
 - liblDT_VP3300_BT.h, [797](#)
 - liblDT_VP3300_COM.h, [849](#)
 - liblDT_VP3300_USB.h, [901](#)
 - liblDT_Vendi.h, [713](#)
- device_registerFWCallBk
 - liblDT_Augusta.h, [70](#)
 - liblDT_Device.h, [172](#)
 - liblDT_L100.h, [344](#)
 - liblDT_L80.h, [377](#)
 - liblDT_NEO2.h, [495](#)
 - liblDT_SREDKey2.h, [652](#)
- device_registerRKICallBk
 - liblDT_Device.h, [173](#)
 - liblDT_NEO2.h, [496](#)
 - liblDT_VP3300_AJ.h, [745](#)
 - liblDT_VP3300_BT.h, [797](#)
 - liblDT_VP3300_COM.h, [849](#)
 - liblDT_VP3300_USB.h, [901](#)
- device_rrcConnect
 - liblDT_Device.h, [173](#)
 - liblDT_NEO2.h, [496](#)
- device_rrcDisconnect
 - liblDT_Device.h, [173](#)
 - liblDT_NEO2.h, [496](#)
- device_rrcDownloadApp
 - liblDT_Device.h, [173](#)
 - liblDT_NEO2.h, [496](#)
- device_rrcInstallApp
 - liblDT_Device.h, [174](#)
 - liblDT_NEO2.h, [497](#)
- device_rrcRunApp
 - liblDT_Device.h, [174](#)
 - liblDT_NEO2.h, [497](#)
- device_rrcUninstallApp
 - liblDT_Device.h, [174](#)
 - liblDT_NEO2.h, [497](#)
- device_selfCheck
 - liblDT_Device.h, [175](#)
- device_setAudioVolume
 - liblDT_Device.h, [176](#)
 - liblDT_NEO2.h, [498](#)
- device_setBurstMode
 - liblDT_Device.h, [177](#)
 - liblDT_KioskIII.h, [320](#)
 - liblDT_NEO2.h, [498](#)
 - liblDT_PipReader.h, [589](#)
 - liblDT_VP3300_AJ.h, [746](#)
 - liblDT_VP3300_BT.h, [798](#)
 - liblDT_VP3300_COM.h, [850](#)
 - liblDT_VP3300_USB.h, [902](#)
 - liblDT_Vendi.h, [714](#)
- device_setCameraParameters
 - liblDT_Device.h, [177](#)
 - liblDT_NEO2.h, [499](#)
- device_setCancelTransactionMode
 - liblDT_Device.h, [177](#)

- liblDT_NEO2.h, [499](#)
- device_setConfigPath
 - liblDT_Device.h, [178](#)
 - liblDT_NEO2.h, [499](#)
 - liblDT_SREDKey2.h, [654](#)
- device_setCoreDumpLogFile
 - liblDT_Device.h, [178](#)
 - liblDT_NEO2.h, [500](#)
- device_setCurrentDevice
 - liblDT_Augusta.h, [71](#)
 - liblDT_Device.h, [178](#)
 - liblDT_KioskIII.h, [320](#)
 - liblDT_L100.h, [345](#)
 - liblDT_L80.h, [378](#)
 - liblDT_MiniSmartII.h, [417](#)
 - liblDT_NEO2.h, [500](#)
 - liblDT_PipReader.h, [589](#)
 - liblDT_SREDKey2.h, [654](#)
 - liblDT_SpectrumPro.h, [617](#)
 - liblDT_UniPayI_V.h, [674](#)
 - liblDT_VP3300_AJ.h, [746](#)
 - liblDT_VP3300_BT.h, [798](#)
 - liblDT_VP3300_COM.h, [850](#)
 - liblDT_VP3300_USB.h, [902](#)
 - liblDT_VP8800.h, [957](#)
 - liblDT_Vendi.h, [715](#)
- device_setMerchantRecord
 - liblDT_Device.h, [179](#)
 - liblDT_KioskIII.h, [321](#)
 - liblDT_NEO2.h, [501](#)
 - liblDT_PipReader.h, [590](#)
 - liblDT_UniPayI_V.h, [674](#)
 - liblDT_VP3300_AJ.h, [747](#)
 - liblDT_VP3300_BT.h, [799](#)
 - liblDT_VP3300_COM.h, [851](#)
 - liblDT_VP3300_USB.h, [903](#)
 - liblDT_VP8800.h, [957](#)
 - liblDT_Vendi.h, [715](#)
- device_setNEO2DevicesConfigs
 - liblDT_Device.h, [179](#)
 - liblDT_NEO2.h, [501](#)
 - liblDT_SREDKey2.h, [654](#)
- device_setNEOAltDevice
 - liblDT_Device.h, [180](#)
 - liblDT_NEO2.h, [502](#)
- device_setNEOGen
 - liblDT_Device.h, [180](#)
 - liblDT_NEO2.h, [502](#)
- device_setPollMode
 - liblDT_Device.h, [180](#)
 - liblDT_KioskIII.h, [321](#)
 - liblDT_NEO2.h, [502](#)
 - liblDT_PipReader.h, [590](#)
 - liblDT_VP3300_AJ.h, [747](#)
 - liblDT_VP3300_BT.h, [799](#)
 - liblDT_VP3300_COM.h, [851](#)
 - liblDT_VP3300_USB.h, [903](#)
 - liblDT_Vendi.h, [715](#)
- device_setRKI_URL
 - liblDT_Device.h, [181](#)
 - liblDT_KioskIII.h, [322](#)
 - liblDT_NEO2.h, [502](#)
 - liblDT_VP3300_AJ.h, [748](#)
 - liblDT_VP3300_BT.h, [800](#)
 - liblDT_VP3300_COM.h, [852](#)
 - liblDT_VP3300_USB.h, [904](#)
- device_setRTCDateTime
 - liblDT_Device.h, [181](#)
 - liblDT_VP3300_AJ.h, [748](#)
 - liblDT_VP3300_BT.h, [800](#)
 - liblDT_VP3300_COM.h, [852](#)
 - liblDT_VP3300_USB.h, [904](#)
- device_setSDKWaitTime
 - liblDT_Augusta.h, [71](#)
 - liblDT_Device.h, [181](#)
 - liblDT_KioskIII.h, [322](#)
 - liblDT_MiniSmartII.h, [417](#)
 - liblDT_NEO2.h, [504](#)
 - liblDT_PipReader.h, [590](#)
 - liblDT_SpectrumPro.h, [617](#)
 - liblDT_UniPayI_V.h, [674](#)
 - liblDT_VP3300_AJ.h, [748](#)
 - liblDT_VP3300_BT.h, [800](#)
 - liblDT_VP3300_COM.h, [852](#)
 - liblDT_VP3300_USB.h, [904](#)
 - liblDT_VP8800.h, [957](#)
 - liblDT_Vendi.h, [716](#)
- device_setSleepModeTime
 - liblDT_Device.h, [182](#)
 - liblDT_L100.h, [345](#)
 - liblDT_L80.h, [379](#)
- device_setSystemLanguage
 - liblDT_Device.h, [182](#)
 - liblDT_SREDKey2.h, [655](#)
- device_setThreadStackSize
 - liblDT_Augusta.h, [72](#)
 - liblDT_Device.h, [183](#)
 - liblDT_KioskIII.h, [322](#)
 - liblDT_MiniSmartII.h, [418](#)
 - liblDT_NEO2.h, [504](#)
 - liblDT_SpectrumPro.h, [617](#)
 - liblDT_UniPayI_V.h, [675](#)
 - liblDT_VP3300_AJ.h, [749](#)
 - liblDT_VP3300_BT.h, [801](#)
 - liblDT_VP3300_COM.h, [853](#)
 - liblDT_VP3300_USB.h, [905](#)
 - liblDT_VP8800.h, [958](#)
 - liblDT_Vendi.h, [716](#)
- device_setTransactionExponent
 - liblDT_Augusta.h, [72](#)
 - liblDT_Device.h, [183](#)
 - liblDT_MiniSmartII.h, [418](#)
 - liblDT_NEO2.h, [504](#)
 - liblDT_SREDKey2.h, [655](#)
 - liblDT_VP3300_AJ.h, [749](#)
 - liblDT_VP3300_BT.h, [801](#)

- libIDT_VP3300_COM.h, [853](#)
- libIDT_VP3300_USB.h, [905](#)
- libIDT_VP8800.h, [958](#)
- device_startListenNotifications
 - libIDT_Device.h, [183](#)
 - libIDT_NEO2.h, [504](#)
- device_startQRCodeScan
 - libIDT_Device.h, [183](#)
 - libIDT_NEO2.h, [505](#)
- device_startQRCodeScanWithDisplayWindowInfo
 - libIDT_Device.h, [184](#)
 - libIDT_NEO2.h, [505](#)
- device_startRKI
 - libIDT_Device.h, [184](#)
 - libIDT_KioskIII.h, [322](#)
 - libIDT_NEO2.h, [505](#)
 - libIDT_VP3300_AJ.h, [749](#)
 - libIDT_VP3300_BT.h, [801](#)
 - libIDT_VP3300_COM.h, [853](#)
 - libIDT_VP3300_USB.h, [905](#)
- device_startTakingPhoto
 - libIDT_Device.h, [184](#)
 - libIDT_NEO2.h, [506](#)
- device_startTransaction
 - libIDT_Augusta.h, [72](#)
 - libIDT_Device.h, [185](#)
 - libIDT_MiniSmartII.h, [418](#)
 - libIDT_NEO2.h, [506](#)
 - libIDT_VP3300_AJ.h, [750](#)
 - libIDT_VP3300_BT.h, [802](#)
 - libIDT_VP3300_COM.h, [854](#)
 - libIDT_VP3300_USB.h, [906](#)
 - libIDT_VP8800.h, [958](#)
- device_stopAudio
 - libIDT_Device.h, [186](#)
 - libIDT_NEO2.h, [508](#)
- device_stopListenNotifications
 - libIDT_Device.h, [187](#)
 - libIDT_NEO2.h, [508](#)
- device_stopQRCodeScan
 - libIDT_Device.h, [187](#)
 - libIDT_NEO2.h, [508](#)
- device_stopTakingPhoto
 - libIDT_Device.h, [187](#)
 - libIDT_NEO2.h, [508](#)
- device_toSDCard
 - libIDT_Device.h, [187](#)
 - libIDT_NEO2.h, [509](#)
- device_transferFile
 - libIDT_Device.h, [188](#)
 - libIDT_NEO2.h, [509](#)
 - libIDT_VP8800.h, [960](#)
- device_turnOffYellowLED
 - libIDT_Device.h, [188](#)
 - libIDT_NEO2.h, [510](#)
- device_turnOnYellowLED
 - libIDT_Device.h, [188](#)
 - libIDT_NEO2.h, [510](#)
- device_updateFirmware
 - libIDT_Augusta.h, [73](#)
 - libIDT_Device.h, [188](#)
 - libIDT_L100.h, [346](#)
 - libIDT_L80.h, [379](#)
 - libIDT_MiniSmartII.h, [419](#)
 - libIDT_NEO2.h, [510](#)
 - libIDT_SREDKey2.h, [655](#)
 - libIDT_SpectrumPro.h, [617](#)
- device_verifyBackdoorKey
 - libIDT_Device.h, [189](#)
- emv_activateTransaction
 - libIDT_Augusta.h, [74](#)
 - libIDT_Device.h, [189](#)
 - libIDT_MiniSmartII.h, [420](#)
 - libIDT_NEO2.h, [511](#)
 - libIDT_SpectrumPro.h, [618](#)
 - libIDT_UniPayI_V.h, [675](#)
 - libIDT_VP3300_AJ.h, [751](#)
 - libIDT_VP3300_BT.h, [803](#)
 - libIDT_VP3300_COM.h, [854](#)
 - libIDT_VP3300_USB.h, [906](#)
 - libIDT_VP8800.h, [960](#)
- emv_allowFallback
 - libIDT_Augusta.h, [74](#)
 - libIDT_Device.h, [190](#)
 - libIDT_MiniSmartII.h, [420](#)
 - libIDT_NEO2.h, [511](#)
 - libIDT_SpectrumPro.h, [619](#)
 - libIDT_UniPayI_V.h, [675](#)
 - libIDT_VP3300_AJ.h, [752](#)
 - libIDT_VP3300_BT.h, [804](#)
 - libIDT_VP3300_COM.h, [855](#)
 - libIDT_VP3300_USB.h, [907](#)
 - libIDT_VP8800.h, [961](#)
- emv_authenticateTransaction
 - libIDT_Augusta.h, [75](#)
 - libIDT_Device.h, [190](#)
 - libIDT_MiniSmartII.h, [421](#)
 - libIDT_NEO2.h, [512](#)
 - libIDT_SpectrumPro.h, [619](#)
 - libIDT_UniPayI_V.h, [676](#)
 - libIDT_VP3300_AJ.h, [752](#)
 - libIDT_VP3300_BT.h, [804](#)
 - libIDT_VP3300_COM.h, [855](#)
 - libIDT_VP3300_USB.h, [907](#)
 - libIDT_VP8800.h, [961](#)
- emv_authenticateTransactionWithTimeout
 - libIDT_Augusta.h, [75](#)
 - libIDT_Device.h, [191](#)
 - libIDT_MiniSmartII.h, [421](#)
 - libIDT_NEO2.h, [512](#)
 - libIDT_SpectrumPro.h, [620](#)
 - libIDT_UniPayI_V.h, [676](#)
 - libIDT_VP3300_AJ.h, [752](#)
 - libIDT_VP3300_BT.h, [804](#)
 - libIDT_VP3300_COM.h, [856](#)
 - libIDT_VP3300_USB.h, [908](#)

- libIDT_VP8800.h, 961
- emv_callbackResponseLCD
 - libIDT_Augusta.h, 76
 - libIDT_Device.h, 191
 - libIDT_MiniSmartII.h, 422
 - libIDT_SpectrumPro.h, 620
- emv_callbackResponseMSR
 - libIDT_Augusta.h, 76
 - libIDT_Device.h, 192
 - libIDT_MiniSmartII.h, 422
 - libIDT_SpectrumPro.h, 621
- emv_cancelTransaction
 - libIDT_Augusta.h, 77
 - libIDT_Device.h, 192
 - libIDT_MiniSmartII.h, 423
 - libIDT_NEO2.h, 513
 - libIDT_SpectrumPro.h, 621
 - libIDT_UniPayI_V.h, 677
 - libIDT_VP3300_AJ.h, 753
 - libIDT_VP3300_BT.h, 805
 - libIDT_VP3300_COM.h, 856
 - libIDT_VP3300_USB.h, 908
 - libIDT_VP8800.h, 962
- emv_completeTransaction
 - libIDT_Augusta.h, 77
 - libIDT_Device.h, 193
 - libIDT_MiniSmartII.h, 423
 - libIDT_NEO2.h, 513
 - libIDT_SpectrumPro.h, 622
 - libIDT_UniPayI_V.h, 677
 - libIDT_VP3300_AJ.h, 753
 - libIDT_VP3300_BT.h, 805
 - libIDT_VP3300_COM.h, 856
 - libIDT_VP3300_USB.h, 908
 - libIDT_VP8800.h, 962
- emv_getAutoAuthenticateTransaction
 - libIDT_Augusta.h, 78
 - libIDT_Device.h, 193
 - libIDT_MiniSmartII.h, 424
 - libIDT_NEO2.h, 515
 - libIDT_SpectrumPro.h, 622
 - libIDT_UniPayI_V.h, 678
 - libIDT_VP3300_AJ.h, 754
 - libIDT_VP3300_BT.h, 806
 - libIDT_VP3300_COM.h, 857
 - libIDT_VP3300_USB.h, 909
 - libIDT_VP8800.h, 963
- emv_getAutoCompleteTransaction
 - libIDT_Augusta.h, 78
 - libIDT_Device.h, 193
 - libIDT_MiniSmartII.h, 424
 - libIDT_NEO2.h, 515
 - libIDT_SpectrumPro.h, 622
 - libIDT_UniPayI_V.h, 678
 - libIDT_VP3300_AJ.h, 754
 - libIDT_VP3300_BT.h, 806
 - libIDT_VP3300_COM.h, 857
 - libIDT_VP3300_USB.h, 909
- libIDT_VP8800.h, 963
- emv_getEMVConfigurationCheckValue
 - libIDT_Augusta.h, 78
 - libIDT_Device.h, 194
 - libIDT_MiniSmartII.h, 424
 - libIDT_NEO2.h, 515
 - libIDT_SpectrumPro.h, 623
 - libIDT_VP8800.h, 963
- emv_getEMVKernelCheckValue
 - libIDT_Augusta.h, 78
 - libIDT_Device.h, 194
 - libIDT_MiniSmartII.h, 424
 - libIDT_NEO2.h, 516
 - libIDT_SpectrumPro.h, 623
 - libIDT_VP8800.h, 964
- emv_getEMVKernelVersion
 - libIDT_Augusta.h, 79
 - libIDT_Device.h, 194
 - libIDT_MiniSmartII.h, 425
 - libIDT_NEO2.h, 516
 - libIDT_SpectrumPro.h, 623
 - libIDT_VP8800.h, 964
- emv_getEMVKernelVersion_Len
 - libIDT_Augusta.h, 79
 - libIDT_Device.h, 195
 - libIDT_MiniSmartII.h, 425
 - libIDT_NEO2.h, 516
 - libIDT_SpectrumPro.h, 624
 - libIDT_VP8800.h, 964
- emv_registerCallBk
 - libIDT_Augusta.h, 79
 - libIDT_Device.h, 195
 - libIDT_KioskIII.h, 323
 - libIDT_L100.h, 347
 - libIDT_L80.h, 380
 - libIDT_MiniSmartII.h, 425
 - libIDT_NEO2.h, 517
 - libIDT_PipReader.h, 591
 - libIDT_SREDKey2.h, 656
 - libIDT_SpectrumPro.h, 624
 - libIDT_UniPayI_V.h, 678
 - libIDT_VP3300_AJ.h, 754
 - libIDT_VP3300_BT.h, 806
 - libIDT_VP3300_COM.h, 858
 - libIDT_VP3300_USB.h, 910
 - libIDT_VP8800.h, 966
 - libIDT_Vendi.h, 716
- emv_removeAllApplicationData
 - libIDT_Augusta.h, 80
 - libIDT_Device.h, 195
 - libIDT_MiniSmartII.h, 426
 - libIDT_NEO2.h, 517
 - libIDT_SpectrumPro.h, 624
 - libIDT_UniPayI_V.h, 678
 - libIDT_VP3300_AJ.h, 754
 - libIDT_VP3300_BT.h, 806
 - libIDT_VP3300_COM.h, 858
 - libIDT_VP3300_USB.h, 910

- libIDT_VP8800.h, [966](#)
- emv_removeAllCAPK
 - libIDT_Augusta.h, [80](#)
 - libIDT_Device.h, [195](#)
 - libIDT_MiniSmartII.h, [426](#)
 - libIDT_NEO2.h, [517](#)
 - libIDT_SpectrumPro.h, [624](#)
 - libIDT_UniPayI_V.h, [679](#)
 - libIDT_VP3300_AJ.h, [755](#)
 - libIDT_VP3300_BT.h, [807](#)
 - libIDT_VP3300_COM.h, [858](#)
 - libIDT_VP3300_USB.h, [910](#)
 - libIDT_VP8800.h, [966](#)
- emv_removeAllCRL
 - libIDT_Augusta.h, [80](#)
 - libIDT_Device.h, [196](#)
 - libIDT_MiniSmartII.h, [426](#)
 - libIDT_NEO2.h, [517](#)
 - libIDT_SpectrumPro.h, [625](#)
 - libIDT_UniPayI_V.h, [679](#)
 - libIDT_VP3300_AJ.h, [755](#)
 - libIDT_VP3300_BT.h, [807](#)
 - libIDT_VP3300_COM.h, [858](#)
 - libIDT_VP3300_USB.h, [910](#)
 - libIDT_VP8800.h, [966](#)
- emv_removeAllExceptions
 - libIDT_VP8800.h, [966](#)
- emv_removeApplicationData
 - libIDT_Augusta.h, [80](#)
 - libIDT_Device.h, [196](#)
 - libIDT_MiniSmartII.h, [426](#)
 - libIDT_NEO2.h, [517](#)
 - libIDT_SpectrumPro.h, [625](#)
 - libIDT_UniPayI_V.h, [679](#)
 - libIDT_VP3300_AJ.h, [755](#)
 - libIDT_VP3300_BT.h, [807](#)
 - libIDT_VP3300_COM.h, [858](#)
 - libIDT_VP3300_USB.h, [910](#)
 - libIDT_VP8800.h, [967](#)
- emv_removeCAPK
 - libIDT_Augusta.h, [81](#)
 - libIDT_Device.h, [196](#)
 - libIDT_MiniSmartII.h, [427](#)
 - libIDT_NEO2.h, [519](#)
 - libIDT_SpectrumPro.h, [625](#)
 - libIDT_UniPayI_V.h, [679](#)
 - libIDT_VP3300_AJ.h, [756](#)
 - libIDT_VP3300_BT.h, [808](#)
 - libIDT_VP3300_COM.h, [860](#)
 - libIDT_VP3300_USB.h, [912](#)
 - libIDT_VP8800.h, [967](#)
- emv_removeCRL
 - libIDT_Augusta.h, [81](#)
 - libIDT_Device.h, [197](#)
 - libIDT_MiniSmartII.h, [427](#)
 - libIDT_NEO2.h, [519](#)
 - libIDT_SpectrumPro.h, [626](#)
 - libIDT_UniPayI_V.h, [680](#)
- libIDT_VP3300_AJ.h, [756](#)
- libIDT_VP3300_BT.h, [808](#)
- libIDT_VP3300_COM.h, [860](#)
- libIDT_VP3300_USB.h, [912](#)
- libIDT_VP8800.h, [967](#)
- emv_removeException
 - libIDT_VP8800.h, [968](#)
- emv_removeTerminalData
 - libIDT_Augusta.h, [81](#)
 - libIDT_Device.h, [197](#)
 - libIDT_MiniSmartII.h, [427](#)
 - libIDT_SpectrumPro.h, [626](#)
- emv_removeTransactionLog
 - libIDT_VP8800.h, [968](#)
- emv_retrieveAIDList
 - libIDT_Augusta.h, [82](#)
 - libIDT_Device.h, [197](#)
 - libIDT_MiniSmartII.h, [428](#)
 - libIDT_NEO2.h, [519](#)
 - libIDT_SpectrumPro.h, [626](#)
 - libIDT_UniPayI_V.h, [680](#)
 - libIDT_VP3300_AJ.h, [756](#)
 - libIDT_VP3300_BT.h, [808](#)
 - libIDT_VP3300_COM.h, [860](#)
 - libIDT_VP3300_USB.h, [912](#)
 - libIDT_VP8800.h, [968](#)
- emv_retrieveApplicationData
 - libIDT_Augusta.h, [82](#)
 - libIDT_Device.h, [198](#)
 - libIDT_MiniSmartII.h, [428](#)
 - libIDT_NEO2.h, [520](#)
 - libIDT_SpectrumPro.h, [627](#)
 - libIDT_UniPayI_V.h, [681](#)
 - libIDT_VP3300_AJ.h, [757](#)
 - libIDT_VP3300_BT.h, [809](#)
 - libIDT_VP3300_COM.h, [861](#)
 - libIDT_VP3300_USB.h, [913](#)
 - libIDT_VP8800.h, [969](#)
- emv_retrieveCAPKList
 - libIDT_Augusta.h, [83](#)
 - libIDT_Device.h, [199](#)
 - libIDT_MiniSmartII.h, [429](#)
 - libIDT_NEO2.h, [521](#)
 - libIDT_SpectrumPro.h, [628](#)
 - libIDT_UniPayI_V.h, [682](#)
 - libIDT_VP3300_AJ.h, [758](#)
 - libIDT_VP3300_BT.h, [810](#)
 - libIDT_VP3300_COM.h, [862](#)
 - libIDT_VP3300_USB.h, [914](#)
 - libIDT_VP8800.h, [970](#)
- emv_retrieveCAPK
 - libIDT_Augusta.h, [83](#)
 - libIDT_Device.h, [198](#)
 - libIDT_MiniSmartII.h, [429](#)
 - libIDT_NEO2.h, [520](#)
 - libIDT_SpectrumPro.h, [627](#)
 - libIDT_UniPayI_V.h, [681](#)
 - libIDT_VP3300_AJ.h, [757](#)

- libIDT_VP3300_BT.h, [809](#)
- libIDT_VP3300_COM.h, [861](#)
- libIDT_VP3300_USB.h, [913](#)
- libIDT_VP8800.h, [969](#)
- emv_retrieveCRL
 - libIDT_Augusta.h, [84](#)
 - libIDT_Device.h, [199](#)
 - libIDT_MiniSmartII.h, [430](#)
 - libIDT_NEO2.h, [521](#)
 - libIDT_SpectrumPro.h, [628](#)
 - libIDT_UniPayI_V.h, [682](#)
 - libIDT_VP3300_AJ.h, [758](#)
 - libIDT_VP3300_BT.h, [810](#)
 - libIDT_VP3300_COM.h, [862](#)
 - libIDT_VP3300_USB.h, [914](#)
 - libIDT_VP8800.h, [970](#)
- emv_retrieveExceptionList
 - libIDT_VP8800.h, [971](#)
- emv_retrieveExceptionLogStatus
 - libIDT_VP8800.h, [971](#)
- emv_retrieveTerminalData
 - libIDT_Augusta.h, [84](#)
 - libIDT_Device.h, [200](#)
 - libIDT_MiniSmartII.h, [430](#)
 - libIDT_NEO2.h, [522](#)
 - libIDT_SpectrumPro.h, [629](#)
 - libIDT_UniPayI_V.h, [683](#)
 - libIDT_VP3300_AJ.h, [759](#)
 - libIDT_VP3300_BT.h, [811](#)
 - libIDT_VP3300_COM.h, [863](#)
 - libIDT_VP3300_USB.h, [915](#)
 - libIDT_VP8800.h, [972](#)
- emv_retrieveTerminalID_Len
 - libIDT_Augusta.h, [85](#)
 - libIDT_Device.h, [200](#)
 - libIDT_MiniSmartII.h, [431](#)
 - libIDT_SpectrumPro.h, [629](#)
- emv_retrieveTerminalID
 - libIDT_Augusta.h, [84](#)
 - libIDT_Device.h, [200](#)
 - libIDT_MiniSmartII.h, [430](#)
 - libIDT_SpectrumPro.h, [629](#)
- emv_retrieveTransactionLog
 - libIDT_VP8800.h, [972](#)
- emv_retrieveTransactionLogStatus
 - libIDT_VP8800.h, [973](#)
- emv_retrieveTransactionResult
 - libIDT_Augusta.h, [85](#)
 - libIDT_Device.h, [201](#)
 - libIDT_MiniSmartII.h, [431](#)
 - libIDT_NEO2.h, [522](#)
 - libIDT_SpectrumPro.h, [630](#)
- emv_setApplicationData
 - libIDT_Augusta.h, [86](#)
 - libIDT_Device.h, [201](#)
 - libIDT_MiniSmartII.h, [432](#)
 - libIDT_NEO2.h, [523](#)
 - libIDT_SpectrumPro.h, [630](#)
- libIDT_UniPayI_V.h, [683](#)
- libIDT_VP3300_AJ.h, [759](#)
- libIDT_VP3300_BT.h, [811](#)
- libIDT_VP3300_COM.h, [863](#)
- libIDT_VP3300_USB.h, [915](#)
- libIDT_VP8800.h, [974](#)
- emv_setApplicationDataTLV
 - libIDT_Device.h, [202](#)
 - libIDT_NEO2.h, [523](#)
 - libIDT_UniPayI_V.h, [684](#)
 - libIDT_VP3300_AJ.h, [760](#)
 - libIDT_VP3300_BT.h, [812](#)
 - libIDT_VP3300_COM.h, [864](#)
 - libIDT_VP3300_USB.h, [916](#)
 - libIDT_VP8800.h, [974](#)
- emv_setAutoAuthenticateTransaction
 - libIDT_Augusta.h, [86](#)
 - libIDT_Device.h, [202](#)
 - libIDT_MiniSmartII.h, [432](#)
 - libIDT_NEO2.h, [523](#)
 - libIDT_SpectrumPro.h, [631](#)
 - libIDT_UniPayI_V.h, [684](#)
 - libIDT_VP3300_AJ.h, [760](#)
 - libIDT_VP3300_BT.h, [812](#)
 - libIDT_VP3300_COM.h, [864](#)
 - libIDT_VP3300_USB.h, [916](#)
 - libIDT_VP8800.h, [975](#)
- emv_setAutoCompleteTransaction
 - libIDT_Augusta.h, [86](#)
 - libIDT_Device.h, [202](#)
 - libIDT_MiniSmartII.h, [432](#)
 - libIDT_NEO2.h, [524](#)
 - libIDT_SpectrumPro.h, [631](#)
 - libIDT_UniPayI_V.h, [684](#)
 - libIDT_VP3300_AJ.h, [760](#)
 - libIDT_VP3300_BT.h, [812](#)
 - libIDT_VP3300_COM.h, [864](#)
 - libIDT_VP3300_USB.h, [916](#)
 - libIDT_VP8800.h, [975](#)
- emv_setCAPK
 - libIDT_Augusta.h, [87](#)
 - libIDT_Device.h, [203](#)
 - libIDT_MiniSmartII.h, [433](#)
 - libIDT_NEO2.h, [524](#)
 - libIDT_SpectrumPro.h, [631](#)
 - libIDT_UniPayI_V.h, [684](#)
 - libIDT_VP3300_AJ.h, [760](#)
 - libIDT_VP3300_BT.h, [812](#)
 - libIDT_VP3300_COM.h, [865](#)
 - libIDT_VP3300_USB.h, [917](#)
 - libIDT_VP8800.h, [975](#)
- emv_setCRL
 - libIDT_Augusta.h, [87](#)
 - libIDT_Device.h, [203](#)
 - libIDT_MiniSmartII.h, [433](#)
 - libIDT_NEO2.h, [525](#)
 - libIDT_SpectrumPro.h, [632](#)
 - libIDT_UniPayI_V.h, [685](#)

- libIDT_VP3300_AJ.h, [761](#)
- libIDT_VP3300_BT.h, [813](#)
- libIDT_VP3300_COM.h, [865](#)
- libIDT_VP3300_USB.h, [917](#)
- libIDT_VP8800.h, [976](#)
- emv_setException
 - libIDT_VP8800.h, [976](#)
- emv_setTerminalData
 - libIDT_Augusta.h, [88](#)
 - libIDT_Device.h, [204](#)
 - libIDT_MiniSmartII.h, [434](#)
 - libIDT_NEO2.h, [525](#)
 - libIDT_SpectrumPro.h, [632](#)
 - libIDT_UniPayI_V.h, [685](#)
 - libIDT_VP3300_AJ.h, [761](#)
 - libIDT_VP3300_BT.h, [813](#)
 - libIDT_VP3300_COM.h, [866](#)
 - libIDT_VP3300_USB.h, [918](#)
 - libIDT_VP8800.h, [978](#)
- emv_setTerminalID
 - libIDT_Augusta.h, [88](#)
 - libIDT_Device.h, [204](#)
 - libIDT_MiniSmartII.h, [434](#)
 - libIDT_SpectrumPro.h, [634](#)
- emv_setTerminalMajorConfiguration
 - libIDT_Device.h, [204](#)
 - libIDT_NEO2.h, [525](#)
 - libIDT_UniPayI_V.h, [686](#)
 - libIDT_VP3300_AJ.h, [762](#)
 - libIDT_VP3300_BT.h, [814](#)
 - libIDT_VP3300_COM.h, [866](#)
 - libIDT_VP3300_USB.h, [918](#)
- emv_setTransactionParameters
 - libIDT_Device.h, [205](#)
 - libIDT_NEO2.h, [526](#)
 - libIDT_VP3300_AJ.h, [762](#)
 - libIDT_VP3300_BT.h, [814](#)
 - libIDT_VP3300_COM.h, [866](#)
 - libIDT_VP3300_USB.h, [918](#)
- emv_startTransaction
 - libIDT_Augusta.h, [88](#)
 - libIDT_Device.h, [205](#)
 - libIDT_MiniSmartII.h, [434](#)
 - libIDT_NEO2.h, [526](#)
 - libIDT_SpectrumPro.h, [634](#)
 - libIDT_UniPayI_V.h, [686](#)
 - libIDT_VP3300_AJ.h, [763](#)
 - libIDT_VP3300_BT.h, [815](#)
 - libIDT_VP3300_COM.h, [867](#)
 - libIDT_VP3300_USB.h, [919](#)
 - libIDT_VP8800.h, [978](#)
- executeTransaction
 - libIDT_Device.h, [206](#)
 - libIDT_NEO2.h, [527](#)
 - libIDT_VP3300_AJ.h, [764](#)
 - libIDT_VP3300_BT.h, [816](#)
 - libIDT_VP3300_COM.h, [868](#)
 - libIDT_VP3300_USB.h, [920](#)
- executeTransaction_WorldNet
 - libIDT_Device.h, [207](#)
 - libIDT_NEO2.h, [528](#)
 - libIDT_VP3300_AJ.h, [764](#)
 - libIDT_VP3300_BT.h, [816](#)
 - libIDT_VP3300_COM.h, [868](#)
 - libIDT_VP3300_USB.h, [920](#)
- felica_SendCommand
 - libIDT_Device.h, [210](#)
 - libIDT_NEO2.h, [531](#)
- felica_authentication
 - libIDT_Device.h, [207](#)
 - libIDT_NEO2.h, [528](#)
- felica_cancelCodeEntry
 - libIDT_Device.h, [208](#)
 - libIDT_NEO2.h, [529](#)
- felica_getCode
 - libIDT_Device.h, [208](#)
 - libIDT_NEO2.h, [529](#)
- felica_poll
 - libIDT_Device.h, [208](#)
 - libIDT_NEO2.h, [529](#)
- felica_read
 - libIDT_Device.h, [209](#)
 - libIDT_NEO2.h, [530](#)
- felica_readWithMac
 - libIDT_Device.h, [209](#)
 - libIDT_NEO2.h, [530](#)
- felica_requestService
 - libIDT_Device.h, [210](#)
 - libIDT_NEO2.h, [531](#)
- felica_write
 - libIDT_Device.h, [210](#)
 - libIDT_NEO2.h, [531](#)
- felica_writeWithMac
 - libIDT_Device.h, [211](#)
 - libIDT_NEO2.h, [532](#)
- forwardTransaction
 - libIDT_Device.h, [211](#)
 - libIDT_NEO2.h, [532](#)
 - libIDT_VP3300_AJ.h, [765](#)
 - libIDT_VP3300_BT.h, [817](#)
 - libIDT_VP3300_COM.h, [869](#)
 - libIDT_VP3300_USB.h, [921](#)
- forwardTransaction_WorldNet
 - libIDT_Device.h, [212](#)
 - libIDT_NEO2.h, [533](#)
 - libIDT_VP3300_AJ.h, [765](#)
 - libIDT_VP3300_BT.h, [817](#)
 - libIDT_VP3300_COM.h, [869](#)
 - libIDT_VP3300_USB.h, [921](#)
- ftpComm_callBack
 - libIDT_Augusta.h, [47](#)
 - libIDT_Device.h, [112](#)
 - libIDT_KioskIII.h, [299](#)
 - libIDT_L100.h, [327](#)
 - libIDT_L80.h, [360](#)
 - libIDT_MiniSmartII.h, [394](#)

- libIDT_NEO2.h, [448](#)
- libIDT_PipReader.h, [569](#)
- libIDT_SREDKey2.h, [643](#)
- libIDT_SpectrumPro.h, [595](#)
- libIDT_UniPayI_V.h, [664](#)
- libIDT_VP3300_AJ.h, [722](#)
- libIDT_VP3300_BT.h, [774](#)
- libIDT_VP3300_COM.h, [826](#)
- libIDT_VP3300_USB.h, [877](#)
- libIDT_VP8800.h, [930](#)
- libIDT_Vendi.h, [693](#)
- httpComm_callBack
 - libIDT_Augusta.h, [47](#)
 - libIDT_Device.h, [113](#)
 - libIDT_KioskIII.h, [299](#)
 - libIDT_L100.h, [327](#)
 - libIDT_L80.h, [360](#)
 - libIDT_MiniSmartII.h, [394](#)
 - libIDT_NEO2.h, [448](#)
 - libIDT_PipReader.h, [569](#)
 - libIDT_SREDKey2.h, [643](#)
 - libIDT_SpectrumPro.h, [596](#)
 - libIDT_UniPayI_V.h, [664](#)
 - libIDT_VP3300_AJ.h, [722](#)
 - libIDT_VP3300_BT.h, [774](#)
 - libIDT_VP3300_COM.h, [826](#)
 - libIDT_VP3300_USB.h, [878](#)
 - libIDT_VP8800.h, [930](#)
 - libIDT_Vendi.h, [693](#)
- IN_OUT
 - libIDT_Augusta.h, [47](#)
 - libIDT_Device.h, [112](#)
 - libIDT_KioskIII.h, [299](#)
 - libIDT_L100.h, [326](#)
 - libIDT_L80.h, [360](#)
 - libIDT_MiniSmartII.h, [394](#)
 - libIDT_NEO2.h, [448](#)
 - libIDT_PipReader.h, [568](#)
 - libIDT_SREDKey2.h, [643](#)
 - libIDT_SpectrumPro.h, [595](#)
 - libIDT_UniPayI_V.h, [664](#)
 - libIDT_VP3300_AJ.h, [722](#)
 - libIDT_VP3300_BT.h, [773](#)
 - libIDT_VP3300_COM.h, [825](#)
 - libIDT_VP3300_USB.h, [877](#)
 - libIDT_VP8800.h, [929](#)
 - libIDT_Vendi.h, [693](#)
- icc_disable
 - libIDT_Augusta.h, [89](#)
 - libIDT_Device.h, [213](#)
 - libIDT_MiniSmartII.h, [435](#)
- icc_enable
 - libIDT_Augusta.h, [89](#)
 - libIDT_Device.h, [213](#)
 - libIDT_MiniSmartII.h, [435](#)
- icc_exchangeAPDU
 - libIDT_Augusta.h, [90](#)
- libIDT_Device.h, [213](#)
- libIDT_MiniSmartII.h, [436](#)
- libIDT_NEO2.h, [534](#)
- libIDT_UniPayI_V.h, [687](#)
- libIDT_VP3300_AJ.h, [766](#)
- libIDT_VP3300_BT.h, [818](#)
- libIDT_VP3300_COM.h, [870](#)
- libIDT_VP3300_USB.h, [922](#)
- icc_exchangeEncryptedAPDU
 - libIDT_Augusta.h, [90](#)
 - libIDT_Device.h, [214](#)
 - libIDT_MiniSmartII.h, [436](#)
- icc_getAPDU_KSN
 - libIDT_Augusta.h, [91](#)
 - libIDT_Device.h, [214](#)
 - libIDT_MiniSmartII.h, [437](#)
- icc_getFunctionStatus
 - libIDT_Augusta.h, [91](#)
 - libIDT_Device.h, [215](#)
 - libIDT_MiniSmartII.h, [437](#)
- icc_getICCReaderStatus
 - libIDT_Augusta.h, [92](#)
 - libIDT_Device.h, [215](#)
 - libIDT_MiniSmartII.h, [438](#)
 - libIDT_NEO2.h, [534](#)
 - libIDT_SpectrumPro.h, [635](#)
 - libIDT_UniPayI_V.h, [688](#)
 - libIDT_VP3300_AJ.h, [766](#)
 - libIDT_VP3300_BT.h, [818](#)
 - libIDT_VP3300_COM.h, [870](#)
 - libIDT_VP3300_USB.h, [922](#)
- icc_getKeyFormatForICCDUKPT
 - libIDT_Augusta.h, [92](#)
 - libIDT_Device.h, [216](#)
 - libIDT_MiniSmartII.h, [438](#)
- icc_getKeyTypeForICCDUKPT
 - libIDT_Augusta.h, [93](#)
 - libIDT_Device.h, [216](#)
 - libIDT_MiniSmartII.h, [439](#)
- icc_powerOffICC
 - libIDT_Augusta.h, [93](#)
 - libIDT_Device.h, [216](#)
 - libIDT_MiniSmartII.h, [439](#)
 - libIDT_NEO2.h, [534](#)
 - libIDT_SpectrumPro.h, [635](#)
 - libIDT_UniPayI_V.h, [688](#)
 - libIDT_VP3300_AJ.h, [768](#)
 - libIDT_VP3300_BT.h, [820](#)
 - libIDT_VP3300_COM.h, [871](#)
 - libIDT_VP3300_USB.h, [923](#)
- icc_powerOnICC
 - libIDT_Augusta.h, [93](#)
 - libIDT_Device.h, [217](#)
 - libIDT_MiniSmartII.h, [439](#)
 - libIDT_NEO2.h, [535](#)
 - libIDT_SpectrumPro.h, [635](#)
 - libIDT_UniPayI_V.h, [688](#)
 - libIDT_VP3300_AJ.h, [768](#)

- liblDT_VP3300_BT.h, [820](#)
- liblDT_VP3300_COM.h, [871](#)
- liblDT_VP3300_USB.h, [923](#)
- icc_setKeyFormatForICCDUKPT
 - liblDT_Device.h, [217](#)
- icc_setKeyTypeForICCDUKPT
 - liblDT_Device.h, [217](#)
- IN
 - liblDT_Augusta.h, [47](#)
 - liblDT_Device.h, [112](#)
 - liblDT_KioskIII.h, [299](#)
 - liblDT_L100.h, [326](#)
 - liblDT_L80.h, [360](#)
 - liblDT_MiniSmartII.h, [394](#)
 - liblDT_NEO2.h, [448](#)
 - liblDT_PipReader.h, [568](#)
 - liblDT_SREDKey2.h, [642](#)
 - liblDT_SpectrumPro.h, [595](#)
 - liblDT_UniPayI_V.h, [664](#)
 - liblDT_VP3300_AJ.h, [721](#)
 - liblDT_VP3300_BT.h, [773](#)
 - liblDT_VP3300_COM.h, [825](#)
 - liblDT_VP3300_USB.h, [877](#)
 - liblDT_VP8800.h, [929](#)
 - liblDT_Vendi.h, [692](#)
- iso8583_deserializeFromXML
 - liblDT_Device.h, [218](#)
- iso8583_displayMessage
 - liblDT_Device.h, [218](#)
- iso8583_freeMessage
 - liblDT_Device.h, [219](#)
- iso8583_get1987Handler
 - liblDT_Device.h, [219](#)
- iso8583_get1993Handler
 - liblDT_Device.h, [219](#)
- iso8583_get2003Handler
 - liblDT_Device.h, [220](#)
- iso8583_getField
 - liblDT_Device.h, [220](#)
- iso8583_getMessageField
 - liblDT_Device.h, [220](#)
- iso8583_initializeMessage
 - liblDT_Device.h, [221](#)
- iso8583_packMessage
 - liblDT_Device.h, [221](#)
- iso8583_removeMessageField
 - liblDT_Device.h, [221](#)
- iso8583_serializeToXML
 - liblDT_Device.h, [222](#)
- iso8583_setMessageField
 - liblDT_Device.h, [222](#)
- iso8583_unpackMessage
 - liblDT_Device.h, [223](#)
- lcd_addButton
 - liblDT_Device.h, [223](#)
 - liblDT_NEO2.h, [535](#)
- lcd_addEthernet
 - liblDT_Device.h, [224](#)
- liblDT_NEO2.h, [536](#)
- lcd_addExtVideo
 - liblDT_Device.h, [225](#)
 - liblDT_NEO2.h, [537](#)
- lcd_addImage
 - liblDT_Device.h, [226](#)
 - liblDT_NEO2.h, [538](#)
- lcd_addItemToList
 - liblDT_Device.h, [227](#)
 - liblDT_VP8800.h, [979](#)
- lcd_addLED
 - liblDT_Device.h, [228](#)
 - liblDT_NEO2.h, [539](#)
- lcd_addText
 - liblDT_Device.h, [229](#)
 - liblDT_NEO2.h, [541](#)
- lcd_addVideo
 - liblDT_Device.h, [231](#)
 - liblDT_NEO2.h, [543](#)
- lcd_cancelSlideShow
 - liblDT_Device.h, [233](#)
 - liblDT_VP8800.h, [979](#)
- lcd_captureSignature
 - liblDT_Device.h, [233](#)
 - liblDT_VP8800.h, [980](#)
- lcd_clearDisplay
 - liblDT_Device.h, [233](#)
 - liblDT_NEO2.h, [544](#)
 - liblDT_VP8800.h, [980](#)
- lcd_clearEventQueue
 - liblDT_Device.h, [234](#)
 - liblDT_VP8800.h, [980](#)
- lcd_clearScreenInfo
 - liblDT_Device.h, [234](#)
 - liblDT_NEO2.h, [545](#)
- lcd_cloneScreen
 - liblDT_Device.h, [234](#)
 - liblDT_NEO2.h, [545](#)
- lcd_createInputField
 - liblDT_Device.h, [234](#)
 - liblDT_VP8800.h, [981](#)
- lcd_createInputField_Len
 - liblDT_Device.h, [236](#)
 - liblDT_VP8800.h, [982](#)
- lcd_createList
 - liblDT_Device.h, [237](#)
 - liblDT_VP8800.h, [983](#)
- lcd_createList_Len
 - liblDT_Device.h, [238](#)
 - liblDT_VP8800.h, [984](#)
- lcd_createScreen
 - liblDT_Device.h, [240](#)
 - liblDT_NEO2.h, [545](#)
- lcd_customDisplayMode
 - liblDT_Device.h, [240](#)
 - liblDT_VP8800.h, [986](#)
- lcd_destroyScreen
 - liblDT_Device.h, [240](#)

- libIDT_NEO2.h, 546
- lcd_displayButton
 - libIDT_Device.h, 241
 - libIDT_VP8800.h, 986
- lcd_displayButton_Len
 - libIDT_Device.h, 242
 - libIDT_VP8800.h, 988
- lcd_displayMessage
 - libIDT_Device.h, 244
 - libIDT_L100.h, 347
 - libIDT_L80.h, 380
 - libIDT_NEO2.h, 546
- lcd_displayParagraph
 - libIDT_Device.h, 244
 - libIDT_VP8800.h, 989
- lcd_displayPrompt
 - libIDT_Device.h, 246
 - libIDT_L100.h, 347
 - libIDT_L80.h, 380
- lcd_displayText
 - libIDT_Device.h, 246
 - libIDT_VP8800.h, 991
- lcd_displayText_Len
 - libIDT_Device.h, 247
 - libIDT_VP8800.h, 992
- lcd_enableBacklight
 - libIDT_Device.h, 249
 - libIDT_L100.h, 347
 - libIDT_L80.h, 381
- lcd_getActiveScreen
 - libIDT_Device.h, 249
 - libIDT_NEO2.h, 546
- lcd_getAllObjects
 - libIDT_Device.h, 249
 - libIDT_NEO2.h, 547
- lcd_getAllScreens
 - libIDT_Device.h, 250
 - libIDT_NEO2.h, 547
- lcd_getBacklightStatus
 - libIDT_Device.h, 250
 - libIDT_L100.h, 348
 - libIDT_L80.h, 381
- lcd_getButtonEvent
 - libIDT_Device.h, 250
 - libIDT_NEO2.h, 548
- lcd_getInputEvent
 - libIDT_Device.h, 251
 - libIDT_VP8800.h, 993
- lcd_getInputEvent_Len
 - libIDT_Device.h, 253
 - libIDT_VP8800.h, 995
- lcd_getInputFieldValue
 - libIDT_Device.h, 255
 - libIDT_VP8800.h, 997
- lcd_getSelectedListItem
 - libIDT_Device.h, 255
 - libIDT_VP8800.h, 998
- lcd_getSelectedListItem_Len
 - libIDT_Device.h, 256
 - libIDT_VP8800.h, 998
- lcd_linkUIWithTransactionMessageId
 - libIDT_Device.h, 256
 - libIDT_NEO2.h, 548
- lcd_loadScreenInfo
 - libIDT_Device.h, 256
 - libIDT_NEO2.h, 549
- lcd_queryObjectbyID
 - libIDT_Device.h, 257
 - libIDT_NEO2.h, 549
- lcd_queryObjectbyName
 - libIDT_Device.h, 257
 - libIDT_NEO2.h, 549
- lcd_queryScreenbyID
 - libIDT_Device.h, 258
 - libIDT_NEO2.h, 550
- lcd_queryScreenbyName
 - libIDT_Device.h, 258
 - libIDT_NEO2.h, 550
- lcd_registerCallBk
 - libIDT_Device.h, 258
 - libIDT_NEO2.h, 551
 - libIDT_SREDKey2.h, 656
- lcd_removeItem
 - libIDT_Device.h, 259
 - libIDT_NEO2.h, 551
- lcd_resetInitialState
 - libIDT_Device.h, 259
 - libIDT_VP8800.h, 998
- lcd_savePrompt
 - libIDT_Device.h, 259
 - libIDT_L100.h, 348
 - libIDT_L80.h, 381
- lcd_setBackgroundImage
 - libIDT_Device.h, 260
 - libIDT_VP8800.h, 998
- lcd_setBacklight
 - libIDT_Device.h, 260
 - libIDT_NEO2.h, 551
- lcd_setDisplayImage
 - libIDT_Device.h, 260
 - libIDT_VP8800.h, 999
- lcd_setForeBackColor
 - libIDT_Device.h, 261
 - libIDT_VP8800.h, 999
- lcd_showScreen
 - libIDT_Device.h, 262
 - libIDT_NEO2.h, 552
- lcd_startSlideShow
 - libIDT_Device.h, 262
 - libIDT_VP8800.h, 1000
- lcd_storeScreenInfo
 - libIDT_Device.h, 263
 - libIDT_NEO2.h, 552
- lcd_updateColor
 - libIDT_Device.h, 263
 - libIDT_NEO2.h, 552

- lcd_updateLabel
 - libIDT_Device.h, 264
 - libIDT_NEO2.h, 553
- lcd_updatePosition
 - libIDT_Device.h, 264
 - libIDT_NEO2.h, 554
- libIDT_Augusta.h
 - config_getBeeperController, 49
 - config_getEncryptionControl, 49
 - config_getLEDController, 50
 - config_getModelNumber, 50
 - config_getModelNumber_Len, 51
 - config_getSerialNumber, 51
 - config_getSerialNumber_Len, 51
 - config_setBeeperController, 52
 - config_setEncryptionControl, 52
 - config_setLEDController, 53
 - device_SendDataCommand, 70
 - device_cancelTransaction, 53
 - device_close, 53
 - device_controlBeep, 54
 - device_controlLED_ICC, 55
 - device_controlLED_MSR, 55
 - device_controlLED, 54
 - device_getCurrentDeviceType, 56
 - device_getDRS, 56
 - device_getFirmwareVersion, 57
 - device_getFirmwareVersion_Len, 57
 - device_getKeyStatus, 57
 - device_getResponseCodeString, 58
 - device_getSDKWaitTime, 68
 - device_getThreadStackSize, 69
 - device_init, 69
 - device_isAttached, 69
 - device_isConnected, 70
 - device_rebootDevice, 70
 - device_registerCameraCallBk, 70
 - device_registerCardStatusFrontSwitchCallBk, 70
 - device_registerFWCallBk, 70
 - device_setCurrentDevice, 71
 - device_setSDKWaitTime, 71
 - device_setThreadStackSize, 72
 - device_setTransactionExponent, 72
 - device_startTransaction, 72
 - device_updateFirmware, 73
 - emv_activateTransaction, 74
 - emv_allowFallback, 74
 - emv_authenticateTransaction, 75
 - emv_authenticateTransactionWithTimeout, 75
 - emv_callbackResponseLCD, 76
 - emv_callbackResponseMSR, 76
 - emv_cancelTransaction, 77
 - emv_completeTransaction, 77
 - emv_getAutoAuthenticateTransaction, 78
 - emv_getAutoCompleteTransaction, 78
 - emv_getEMVConfigurationCheckValue, 78
 - emv_getEMVKernelCheckValue, 78
 - emv_getEMVKernelVersion, 79
 - emv_getEMVKernelVersion_Len, 79
 - emv_registerCallBk, 79
 - emv_removeAllApplicationData, 80
 - emv_removeAllCAPK, 80
 - emv_removeAllCRL, 80
 - emv_removeApplicationData, 80
 - emv_removeCAPK, 81
 - emv_removeCRL, 81
 - emv_removeTerminalData, 81
 - emv_retrieveAIDList, 82
 - emv_retrieveApplicationData, 82
 - emv_retrieveCAPKList, 83
 - emv_retrieveCAPK, 83
 - emv_retrieveCRL, 84
 - emv_retrieveTerminalData, 84
 - emv_retrieveTerminalID_Len, 85
 - emv_retrieveTerminalID, 84
 - emv_retrieveTransactionResult, 85
 - emv_setApplicationData, 86
 - emv_setAutoAuthenticateTransaction, 86
 - emv_setAutoCompleteTransaction, 86
 - emv_setCAPK, 87
 - emv_setCRL, 87
 - emv_setTerminalData, 88
 - emv_setTerminalID, 88
 - emv_startTransaction, 88
 - ftpComm_callBack, 47
 - httpComm_callBack, 47
 - IN_OUT, 47
 - icc_disable, 89
 - icc_enable, 89
 - icc_exchangeAPDU, 90
 - icc_exchangeEncryptedAPDU, 90
 - icc_getAPDU_KSN, 91
 - icc_getFunctionStatus, 91
 - icc_getICCReaderStatus, 92
 - icc_getKeyFormatForICCDUKPT, 92
 - icc_getKeyTypeForICCDUKPT, 93
 - icc_powerOffICC, 93
 - icc_powerOnICC, 93
 - IN, 47
 - msr_cancelMSRSwipe, 94
 - msr_captureMode, 94
 - msr_disable, 94
 - msr_getClearPANID, 94
 - msr_getExpirationMask, 95
 - msr_getKeyFormatForICCDUKPT, 95
 - msr_getKeyTypeForICCDUKPT, 96
 - msr_getMSRData, 96
 - msr_getSetting, 96
 - msr_getSwipeForcedEncryptionOption, 98
 - msr_getSwipeMaskOption, 98
 - msr_registerCallBk, 98
 - msr_registerCallBkp, 99
 - msr_setClearPANID, 99
 - msr_setExpirationMask, 99
 - msr_setKeyFormatForICCDUKPT, 99
 - msr_setKeyTypeForICCDUKPT, 100

- msr_setSetting, 100
- msr_setSwipeForcedEncryptionOption, 101
- msr_setSwipeMaskOption, 101
- msr_startMSRSwipe, 101
- OUT, 47
- pCMR_callback, 47
- pCSFS_callback, 48
- pEMV_callback, 48
- pFW_callback, 48
- pMSR_callback, 48
- pMSR_callbackp, 48
- pMessageHotplug, 48
- pPIN_callback, 48
- pReadDataLog, 49
- pSendDataLog, 49
- parseMSRData, 102
- pin_cancelPINEntry, 102
- pin_registerCallBk, 102
- registerHotplugCallBk, 102
- registerLogCallBk, 103
- SDK_Version, 103
- setAbsoluteLibraryPath, 103
- v4Comm_callback, 49
- libIDT_Device.h
 - cancelWorldNet, 115
 - cancelWorldPay, 115
 - config_getBeeperController, 115
 - config_getEncryptionControl, 116
 - config_getLEDController, 116
 - config_getModelNumber, 117
 - config_getModelNumber_Len, 117
 - config_getSerialNumber, 117
 - config_getSerialNumber_Len, 118
 - config_setBeeperController, 118
 - config_setCmdTimeOutDuration, 118
 - config_setEncryptionControl, 119
 - config_setLEDController, 119
 - ctls_activateTransaction, 120
 - ctls_cancelTransaction, 121
 - ctls_displayOnlineAuthResult, 121
 - ctls_getAllConfigurationGroups, 122
 - ctls_getConfigurationGroup, 122
 - ctls_registerCallBkp, 123
 - ctls_removeAllApplicationData, 123
 - ctls_removeAllCAPK, 123
 - ctls_removeApplicationData, 123
 - ctls_removeCAPK, 124
 - ctls_removeConfigurationGroup, 124
 - ctls_retrieveAIDList, 124
 - ctls_retrieveApplicationData, 125
 - ctls_retrieveCAPKList, 126
 - ctls_retrieveCAPK, 125
 - ctls_retrieveTerminalData, 126
 - ctls_setApplicationData, 127
 - ctls_setCAPK, 127
 - ctls_setConfigurationGroup, 128
 - ctls_setTerminalData, 128
 - ctls_startTransaction, 129
 - device_SendDataCommand, 175
 - device_SendDataCommandITP, 175
 - device_SendDataCommandNEO, 176
 - device_activateTransaction, 130
 - device_buzzerOnOff, 132
 - device_calibrateParameters, 132
 - device_cancelTransaction, 132
 - device_cancelTransactionSilent, 132
 - device_close, 133
 - device_configureButtons, 133
 - device_controlBeep, 133
 - device_controlIndicator, 134
 - device_controlLED_ICC, 135
 - device_controlLED_MSR, 135
 - device_controlLED, 134
 - device_controlUserInterface, 136
 - device_createDirectory, 137
 - device_deleteDirectory, 138
 - device_deleteFile, 138
 - device_disableBlueLED, 139
 - device_enableBlueLED, 139
 - device_enableExternalLCDMessages, 139
 - device_enableL100PassThrough, 140
 - device_enableL80PassThrough, 140
 - device_enablePassThrough, 140
 - device_enableRFAntenna, 141
 - device_enhancedPassthrough, 141
 - device_enterStopMode, 142
 - device_getAudioVolume, 142
 - device_getButtonConfiguration, 142
 - device_getCameraParameters, 143
 - device_getCurrentDeviceType, 143
 - device_getDRS, 145
 - device_getDateTime, 143
 - device_getDateTime_Len, 143
 - device_getDeviceMemoryUsageInfo, 144
 - device_getDeviceTreeVersion, 145
 - device_getDriveFreeSpace, 145
 - device_getFirmwareVersion, 146
 - device_getFirmwareVersion_Len, 146
 - device_getIDGStatusCodeString, 147
 - device_getKeyStatus, 148
 - device_getL100PassThroughMode, 149
 - device_getL80PassThroughMode, 149
 - device_getMerchantRecord, 150
 - device_getMerchantRecord_Len, 150
 - device_getNEOAltDevice, 151
 - device_getRTCDateTime, 161
 - device_getResponseCodeString, 151
 - device_getSDKWaitTime, 161
 - device_getSpectrumProKSN_Len, 162
 - device_getSpectrumProKSN, 161
 - device_getTamperStatus, 163
 - device_getThreadStackSize, 163
 - device_init, 163
 - device_isAttached, 163
 - device_isConnected, 164
 - device_lcdDisplayClear, 164

- device_lcdDisplayLine1Message, 164
- device_lcdDisplayLine2Message, 164
- device_listDirectory, 165
- device_loadCertCA, 165
- device_outputLog, 166
- device_pingDevice, 166
- device_playAudio, 166
- device_pollCardReader, 167
- device_pollCardReader_Len, 168
- device_pollForToken, 170
- device_queryFile, 171
- device_readFileFromSD, 171
- device_rebootDevice, 172
- device_registerCameraCallBk, 172
- device_registerCardStatusFrontSwitchCallBk, 172
- device_registerFWCallBk, 172
- device_registerRKICallBk, 173
- device_rrcConnect, 173
- device_rrcDisconnect, 173
- device_rrcDownloadApp, 173
- device_rrcInstallApp, 174
- device_rrcRunApp, 174
- device_rrcUninstallApp, 174
- device_selfCheck, 175
- device_setAudioVolume, 176
- device_setBurstMode, 177
- device_setCameraParameters, 177
- device_setCancelTransactionMode, 177
- device_setConfigPath, 178
- device_setCoreDumpLogFile, 178
- device_setCurrentDevice, 178
- device_setMerchantRecord, 179
- device_setNEO2DevicesConfigs, 179
- device_setNEOAltDevice, 180
- device_setNEOGen, 180
- device_setPollMode, 180
- device_setRKI_URL, 181
- device_setRTCDateTime, 181
- device_setSDKWaitTime, 181
- device_setSleepModeTime, 182
- device_setSystemLanguage, 182
- device_setThreadStackSize, 183
- device_setTransactionExponent, 183
- device_startListenNotifications, 183
- device_startQRCodeScan, 183
- device_startQRCodeScanWithDisplayWindowInfo, 184
- device_startRKI, 184
- device_startTakingPhoto, 184
- device_startTransaction, 185
- device_stopAudio, 186
- device_stopListenNotifications, 187
- device_stopQRCodeScan, 187
- device_stopTakingPhoto, 187
- device_toSDCard, 187
- device_transferFile, 188
- device_turnOffYellowLED, 188
- device_turnOnYellowLED, 188
- device_updateFirmware, 188
- device_verifyBackdoorKey, 189
- emv_activateTransaction, 189
- emv_allowFallback, 190
- emv_authenticateTransaction, 190
- emv_authenticateTransactionWithTimeout, 191
- emv_callbackResponseLCD, 191
- emv_callbackResponseMSR, 192
- emv_cancelTransaction, 192
- emv_completeTransaction, 193
- emv_getAutoAuthenticateTransaction, 193
- emv_getAutoCompleteTransaction, 193
- emv_getEMVConfigurationCheckValue, 194
- emv_getEMVKernelCheckValue, 194
- emv_getEMVKernelVersion, 194
- emv_getEMVKernelVersion_Len, 195
- emv_registerCallBk, 195
- emv_removeAllApplicationData, 195
- emv_removeAllCAPK, 195
- emv_removeAllCRL, 196
- emv_removeApplicationData, 196
- emv_removeCAPK, 196
- emv_removeCRL, 197
- emv_removeTerminalData, 197
- emv_retrieveAIDList, 197
- emv_retrieveApplicationData, 198
- emv_retrieveCAPKList, 199
- emv_retrieveCAPK, 198
- emv_retrieveCRL, 199
- emv_retrieveTerminalData, 200
- emv_retrieveTerminalID_Len, 200
- emv_retrieveTerminalID, 200
- emv_retrieveTransactionResult, 201
- emv_setApplicationData, 201
- emv_setApplicationDataTLV, 202
- emv_setAutoAuthenticateTransaction, 202
- emv_setAutoCompleteTransaction, 202
- emv_setCAPK, 203
- emv_setCRL, 203
- emv_setTerminalData, 204
- emv_setTerminalID, 204
- emv_setTerminalMajorConfiguration, 204
- emv_setTransactionParameters, 205
- emv_startTransaction, 205
- executeTransaction, 206
- executeTransaction_WorldNet, 207
- felica_SendCommand, 210
- felica_authentication, 207
- felica_cancelCodeEntry, 208
- felica_getCode, 208
- felica_poll, 208
- felica_read, 209
- felica_readWithMac, 209
- felica_requestService, 210
- felica_write, 210
- felica_writeWithMac, 211
- forwardTransaction, 211
- forwardTransaction_WorldNet, 212

[ftpComm_callBack, 112](#)
[httpComm_callBack, 113](#)
[IN_OUT, 112](#)
[icc_disable, 213](#)
[icc_enable, 213](#)
[icc_exchangeAPDU, 213](#)
[icc_exchangeEncryptedAPDU, 214](#)
[icc_getAPDU_KSN, 214](#)
[icc_getFunctionStatus, 215](#)
[icc_getICCReaderStatus, 215](#)
[icc_getKeyFormatForICCDUKPT, 216](#)
[icc_getKeyTypeForICCDUKPT, 216](#)
[icc_powerOffICC, 216](#)
[icc_powerOnICC, 217](#)
[icc_setKeyFormatForICCDUKPT, 217](#)
[icc_setKeyTypeForICCDUKPT, 217](#)
[IN, 112](#)
[iso8583_deserializeFromXML, 218](#)
[iso8583_displayMessage, 218](#)
[iso8583_freeMessage, 219](#)
[iso8583_get1987Handler, 219](#)
[iso8583_get1993Handler, 219](#)
[iso8583_get2003Handler, 220](#)
[iso8583_getField, 220](#)
[iso8583_getMessageField, 220](#)
[iso8583_initializeMessage, 221](#)
[iso8583_packMessage, 221](#)
[iso8583_removeMessageField, 221](#)
[iso8583_serializeToXML, 222](#)
[iso8583_setMessageField, 222](#)
[iso8583_unpackMessage, 223](#)
[lcd_addButton, 223](#)
[lcd_addEthernet, 224](#)
[lcd_addExtVideo, 225](#)
[lcd_addImage, 226](#)
[lcd_addItemToList, 227](#)
[lcd_addLED, 228](#)
[lcd_addText, 229](#)
[lcd_addVideo, 231](#)
[lcd_cancelSlideShow, 233](#)
[lcd_captureSignature, 233](#)
[lcd_clearDisplay, 233](#)
[lcd_clearEventQueue, 234](#)
[lcd_clearScreenInfo, 234](#)
[lcd_cloneScreen, 234](#)
[lcd_createInputField, 234](#)
[lcd_createInputField_Len, 236](#)
[lcd_createList, 237](#)
[lcd_createList_Len, 238](#)
[lcd_createScreen, 240](#)
[lcd_customDisplayMode, 240](#)
[lcd_destroyScreen, 240](#)
[lcd_displayButton, 241](#)
[lcd_displayButton_Len, 242](#)
[lcd_displayMessage, 244](#)
[lcd_displayParagraph, 244](#)
[lcd_displayPrompt, 246](#)
[lcd_displayText, 246](#)
[lcd_displayText_Len, 247](#)
[lcd_enableBacklight, 249](#)
[lcd_getActiveScreen, 249](#)
[lcd_getAllObjects, 249](#)
[lcd_getAllScreens, 250](#)
[lcd_getBacklightStatus, 250](#)
[lcd_getButtonEvent, 250](#)
[lcd_getInputEvent, 251](#)
[lcd_getInputEvent_Len, 253](#)
[lcd_getInputFieldValue, 255](#)
[lcd_getSelectedListItem, 255](#)
[lcd_getSelectedListItem_Len, 256](#)
[lcd_linkUIWithTransactionMessageId, 256](#)
[lcd_loadScreenInfo, 256](#)
[lcd_queryObjectbyID, 257](#)
[lcd_queryObjectbyName, 257](#)
[lcd_queryScreenbyID, 258](#)
[lcd_queryScreenbyName, 258](#)
[lcd_registerCallBk, 258](#)
[lcd_removeItem, 259](#)
[lcd_resetInitialState, 259](#)
[lcd_savePrompt, 259](#)
[lcd_setBackgroundImage, 260](#)
[lcd_setBacklight, 260](#)
[lcd_setDisplayImage, 260](#)
[lcd_setForeBackColor, 261](#)
[lcd_showScreen, 262](#)
[lcd_startSlideShow, 262](#)
[lcd_storeScreenInfo, 263](#)
[lcd_updateColor, 263](#)
[lcd_updateLabel, 264](#)
[lcd_updatePosition, 264](#)
[loyalty_cancelTransaction, 265](#)
[loyalty_cancelTransactionSilent, 265](#)
[loyalty_registerCallBk, 266](#)
[loyalty_startTransaction, 266](#)
[msr_cancelMSRSwipe, 268](#)
[msr_captureMode, 268](#)
[msr_clearMSRData, 268](#)
[msr_disable, 269](#)
[msr_flushTrackData, 269](#)
[msr_getClearPANID, 269](#)
[msr_getExpirationMask, 269](#)
[msr_getFunctionStatus, 270](#)
[msr_getKeyFormatForICCDUKPT, 270](#)
[msr_getKeyTypeForICCDUKPT, 271](#)
[msr_getMSRData, 271](#)
[msr_getSwipeForcedEncryptionOption, 271](#)
[msr_getSwipeMaskOption, 272](#)
[msr_registerCallBk, 272](#)
[msr_registerCallBkp, 272](#)
[msr_setClearPANID, 272](#)
[msr_setExpirationMask, 273](#)
[msr_setKeyFormatForICCDUKPT, 273](#)
[msr_setKeyTypeForICCDUKPT, 273](#)
[msr_setSwipeForcedEncryptionOption, 274](#)
[msr_setSwipeMaskOption, 274](#)
[msr_startMSRSwipe, 275](#)

- OUT, 112
- pCMR_callBack, 113
- pCSFS_callBack, 113
- pEMV_callBack, 113
- pFW_callBack, 113
- pLCD_callBack, 113
- pLog_callback, 113
- pMSR_callBack, 114
- pMSR_callBackp, 114
- pMessageHotplug, 113
- pPIN_callBack, 114
- pRKI_callBack, 114
- pReadDataLog, 114
- pSendDataLog, 114
- pWN_callBack, 114
- pWP_callBack, 114
- parseMSRData, 275
- parsePINBlockData, 275
- parsePINData, 276
- pin_cancelPINEntry, 276
- pin_capturePin, 276
- pin_capturePinExt, 277
- pin_getEncryptedOnlinePIN, 278
- pin_getEncryptedPIN, 279
- pin_getFunctionKey, 279
- pin_getPAN, 280
- pin_getPIN, 281
- pin_getPanEntry, 280
- pin_inputFromPrompt, 281
- pin_promptCreditDebit, 282
- pin_promptForAmount, 283
- pin_promptForAmountInput, 283
- pin_promptForKeyInput, 286
- pin_promptForNumericKey, 289
- pin_promptForNumericKeyWithSwipe, 290
- pin_registerCallBk, 291
- pin_sendBeep, 291
- pin_setKeyValues, 292
- registerHotplugCallBk, 292
- registerLogCallBk, 292
- rs232_device_init, 292
- SDK_Version, 294
- set_open_com_port_timeout, 294
- setAbsoluteLibraryPath, 294
- v4Comm_callBack, 115
- ws_deleteSSLCert, 295
- ws_getCertChainType, 295
- ws_loadSSLCert, 295
- ws_requestCSR, 296
- ws_revokeSSLCert, 296
- ws_updateRootCertificate, 296
- libIDT_KioskIII.h
 - config_getSerialNumber, 301
 - config_getSerialNumber_Len, 301
 - ctls_activateTransaction, 301
 - ctls_cancelTransaction, 303
 - ctls_getAllConfigurationGroups, 303
 - ctls_getConfigurationGroup, 303
 - ctls_registerCallBk, 304
 - ctls_registerCallBkp, 304
 - ctls_removeAllApplicationData, 304
 - ctls_removeAllCAPK, 304
 - ctls_removeApplicationData, 304
 - ctls_removeCAPK, 305
 - ctls_removeConfigurationGroup, 305
 - ctls_retrieveAIDList, 306
 - ctls_retrieveApplicationData, 306
 - ctls_retrieveCAPKList, 307
 - ctls_retrieveCAPK, 306
 - ctls_retrieveTerminalData, 308
 - ctls_setApplicationData, 308
 - ctls_setCAPK, 308
 - ctls_setConfigurationGroup, 309
 - ctls_setTerminalData, 309
 - ctls_startTransaction, 310
 - device_SendDataCommandNEO, 319
 - device_close, 311
 - device_controlUserInterface, 312
 - device_enablePassThrough, 313
 - device_getCurrentDeviceType, 314
 - device_getFirmwareVersion, 314
 - device_getFirmwareVersion_Len, 314
 - device_getIDGStatusCodeString, 315
 - device_getMerchantRecord, 316
 - device_getMerchantRecord_Len, 317
 - device_getSDKWaitTime, 317
 - device_getThreadStackSize, 317
 - device_getTransactionResults, 318
 - device_init, 318
 - device_isAttached, 318
 - device_isConnected, 319
 - device_pingDevice, 319
 - device_registerCameraCallBk, 319
 - device_registerCardStatusFrontSwitchCallBk, 319
 - device_setBurstMode, 320
 - device_setCurrentDevice, 320
 - device_setMerchantRecord, 321
 - device_setPollMode, 321
 - device_setRKI_URL, 322
 - device_setSDKWaitTime, 322
 - device_setThreadStackSize, 322
 - device_startRKI, 322
 - emv_registerCallBk, 323
 - ftpComm_callBack, 299
 - httpComm_callBack, 299
 - IN_OUT, 299
 - IN, 299
 - OUT, 299
 - pCMR_callBack, 299
 - pCSFS_callBack, 299
 - pEMV_callBack, 300
 - pMSR_callBack, 300
 - pMSR_callBackp, 300
 - pMessageHotplug, 300
 - pPIN_callBack, 300
 - pReadDataLog, 300

- pSendDataLog, [300](#)
- parseMSRData, [323](#)
- pin_registerCallBk, [323](#)
- registerHotplugCallBk, [323](#)
- registerLogCallBk, [323](#)
- rs232_device_init, [324](#)
- SDK_Version, [324](#)
- setAbsoluteLibraryPath, [324](#)
- v4Comm_callBack, [301](#)
- libIDT_L100.h
 - config_getModelNumber, [329](#)
 - config_getModelNumber_Len, [329](#)
 - config_getSerialNumber, [329](#)
 - config_getSerialNumber_Len, [330](#)
 - device_SendDataCommand, [344](#)
 - device_close, [330](#)
 - device_enterStopMode, [330](#)
 - device_getCurrentDeviceType, [330](#)
 - device_getDateTime, [331](#)
 - device_getDateTime_Len, [331](#)
 - device_getFirmwareVersion, [331](#)
 - device_getFirmwareVersion_Len, [332](#)
 - device_getKeyStatus, [332](#)
 - device_getResponseCodeString, [333](#)
 - device_init, [343](#)
 - device_isAttached, [343](#)
 - device_isConnected, [343](#)
 - device_rebootDevice, [344](#)
 - device_registerCameraCallBk, [344](#)
 - device_registerCardStatusFrontSwitchCallBk, [344](#)
 - device_registerFWCallBk, [344](#)
 - device_setCurrentDevice, [345](#)
 - device_setSleepModeTime, [345](#)
 - device_updateFirmware, [346](#)
 - emv_registerCallBk, [347](#)
 - ftpComm_callBack, [327](#)
 - httpComm_callBack, [327](#)
 - IN_OUT, [326](#)
 - IN, [326](#)
 - lcd_displayMessage, [347](#)
 - lcd_displayPrompt, [347](#)
 - lcd_enableBacklight, [347](#)
 - lcd_getBacklightStatus, [348](#)
 - lcd_savePrompt, [348](#)
 - msr_registerCallBk, [349](#)
 - msr_registerCallBkp, [349](#)
 - OUT, [327](#)
 - pCMR_callBack, [327](#)
 - pCSFS_callBack, [327](#)
 - pEMV_callBack, [327](#)
 - pFW_callBack, [327](#)
 - pMSR_callBack, [328](#)
 - pMSR_callBackp, [328](#)
 - pMessageHotplug, [328](#)
 - pPIN_callBack, [328](#)
 - pReadDataLog, [328](#)
 - pSendDataLog, [328](#)
 - pin_getEncryptedPIN, [349](#)
 - pin_getFunctionKey, [349](#)
 - pin_promptForAmountInput, [350](#)
 - pin_promptForKeyInput, [353](#)
 - pin_registerCallBk, [356](#)
 - pin_sendBeep, [356](#)
 - pin_setKeyValues, [357](#)
 - registerHotplugCallBk, [357](#)
 - registerLogCallBk, [357](#)
 - SDK_Version, [357](#)
 - setAbsoluteLibraryPath, [358](#)
 - v4Comm_callBack, [328](#)
- libIDT_L80.h
 - config_getModelNumber, [362](#)
 - config_getModelNumber_Len, [362](#)
 - config_getSerialNumber, [362](#)
 - config_getSerialNumber_Len, [363](#)
 - device_SendDataCommand, [378](#)
 - device_close, [363](#)
 - device_enterStopMode, [363](#)
 - device_getCurrentDeviceType, [364](#)
 - device_getDateTime, [364](#)
 - device_getDateTime_Len, [364](#)
 - device_getFirmwareVersion, [364](#)
 - device_getFirmwareVersion_Len, [365](#)
 - device_getKeyStatus, [365](#)
 - device_getResponseCodeString, [366](#)
 - device_init, [376](#)
 - device_isAttached, [376](#)
 - device_isConnected, [377](#)
 - device_rebootDevice, [377](#)
 - device_registerCameraCallBk, [377](#)
 - device_registerCardStatusFrontSwitchCallBk, [377](#)
 - device_registerFWCallBk, [377](#)
 - device_setCurrentDevice, [378](#)
 - device_setSleepModeTime, [379](#)
 - device_updateFirmware, [379](#)
 - emv_registerCallBk, [380](#)
 - ftpComm_callBack, [360](#)
 - httpComm_callBack, [360](#)
 - IN_OUT, [360](#)
 - IN, [360](#)
 - lcd_displayMessage, [380](#)
 - lcd_displayPrompt, [380](#)
 - lcd_enableBacklight, [381](#)
 - lcd_getBacklightStatus, [381](#)
 - lcd_savePrompt, [381](#)
 - msr_registerCallBk, [382](#)
 - msr_registerCallBkp, [382](#)
 - OUT, [360](#)
 - pCMR_callBack, [360](#)
 - pCSFS_callBack, [360](#)
 - pEMV_callBack, [360](#)
 - pFW_callBack, [361](#)
 - pMSR_callBack, [361](#)
 - pMSR_callBackp, [361](#)
 - pMessageHotplug, [361](#)
 - pPIN_callBack, [361](#)
 - pReadDataLog, [361](#)

- pSendDataLog, 361
- pin_getEncryptedPIN, 382
- pin_getFunctionKey, 383
- pin_promptForAmountInput, 383
- pin_promptForKeyInput, 386
- pin_registerCallBk, 389
- pin_sendBeep, 389
- pin_setKeyValues, 390
- registerHotplugCallBk, 390
- registerLogCallBk, 390
- SDK_Version, 390
- setAbsoluteLibraryPath, 391
- v4Comm_callBack, 362
- libIDT_MiniSmartII.h
 - comm_registerHTTPCallback, 396
 - comm_registerV4Callback, 396
 - config_getBeeperController, 396
 - config_getEncryptionControl, 396
 - config_getLEDController, 397
 - config_getModelNumber, 398
 - config_getModelNumber_Len, 398
 - config_getSerialNumber, 398
 - config_getSerialNumber_Len, 399
 - config_setBeeperController, 399
 - config_setEncryptionControl, 399
 - config_setLEDController, 400
 - device_SendDataCommand, 416
 - device_cancelTransaction, 400
 - device_close, 400
 - device_controlBeep, 401
 - device_controlLED_ICC, 402
 - device_controlLED_MSR, 402
 - device_controlLED, 401
 - device_getCurrentDeviceType, 403
 - device_getFirmwareVersion, 403
 - device_getFirmwareVersion_Len, 403
 - device_getKeyStatus, 404
 - device_getResponseCodeString, 405
 - device_getSDKWaitTime, 415
 - device_getThreadStackSize, 415
 - device_init, 415
 - device_isAttached, 415
 - device_isConnected, 416
 - device_rebootDevice, 416
 - device_registerCameraCallBk, 416
 - device_registerCardStatusFrontSwitchCallBk, 416
 - device_setCurrentDevice, 417
 - device_setSDKWaitTime, 417
 - device_setThreadStackSize, 418
 - device_setTransactionExponent, 418
 - device_startTransaction, 418
 - device_updateFirmware, 419
 - emv_activateTransaction, 420
 - emv_allowFallback, 420
 - emv_authenticateTransaction, 421
 - emv_authenticateTransactionWithTimeout, 421
 - emv_callbackResponseLCD, 422
 - emv_callbackResponseMSR, 422
 - emv_cancelTransaction, 423
 - emv_completeTransaction, 423
 - emv_getAutoAuthenticateTransaction, 424
 - emv_getAutoCompleteTransaction, 424
 - emv_getEMVConfigurationCheckValue, 424
 - emv_getEMVKernelCheckValue, 424
 - emv_getEMVKernelVersion, 425
 - emv_getEMVKernelVersion_Len, 425
 - emv_registerCallBk, 425
 - emv_removeAllApplicationData, 426
 - emv_removeAllCAPK, 426
 - emv_removeAllCRL, 426
 - emv_removeApplicationData, 426
 - emv_removeCAPK, 427
 - emv_removeCRL, 427
 - emv_removeTerminalData, 427
 - emv_retrieveAIDList, 428
 - emv_retrieveApplicationData, 428
 - emv_retrieveCAPKList, 429
 - emv_retrieveCAPK, 429
 - emv_retrieveCRL, 430
 - emv_retrieveTerminalData, 430
 - emv_retrieveTerminalID_Len, 431
 - emv_retrieveTerminalID, 430
 - emv_retrieveTransactionResult, 431
 - emv_setApplicationData, 432
 - emv_setAutoAuthenticateTransaction, 432
 - emv_setAutoCompleteTransaction, 432
 - emv_setCAPK, 433
 - emv_setCRL, 433
 - emv_setTerminalData, 434
 - emv_setTerminalID, 434
 - emv_startTransaction, 434
 - ftpComm_callBack, 394
 - httpComm_callBack, 394
 - IN_OUT, 394
 - icc_disable, 435
 - icc_enable, 435
 - icc_exchangeAPDU, 436
 - icc_exchangeEncryptedAPDU, 436
 - icc_getAPDU_KSN, 437
 - icc_getFunctionStatus, 437
 - icc_getICCReaderStatus, 438
 - icc_getKeyFormatForICCDUKPT, 438
 - icc_getKeyTypeForICCDUKPT, 439
 - icc_powerOffICC, 439
 - icc_powerOnICC, 439
 - IN, 394
 - msr_registerCallBk, 440
 - msr_registerCallBkp, 440
 - OUT, 394
 - pCMR_callBack, 394
 - pCSFS_callBack, 394
 - pEMV_callBack, 394
 - pMSR_callBack, 395
 - pMSR_callBackp, 395
 - pMessageHotplug, 395
 - pPIN_callBack, 395

- pReadDataLog, 395
- pSendDataLog, 395
- pin_registerCallBk, 440
- registerHotplugCallBk, 440
- registerLogCallBk, 440
- rs232_device_init, 440
- SDK_Version, 441
- setAbsoluteLibraryPath, 441
- v4Comm_callBack, 395
- libIDT_NEO2.h
 - cancelWorldNet, 450
 - cancelWorldPay, 450
 - comm_registerHTTPCallback, 451
 - comm_registerV4Callback, 451
 - config_getModelNumber, 451
 - config_getModelNumber_Len, 451
 - config_getSerialNumber, 452
 - config_getSerialNumber_Len, 452
 - config_setCmdTimeOutDuration, 452
 - config_setConfigByJsonFile, 453
 - ctls_activateTransaction, 453
 - ctls_cancelTransaction, 454
 - ctls_displayOnlineAuthResult, 455
 - ctls_getAllConfigurationGroups, 455
 - ctls_getConfigurationGroup, 455
 - ctls_registerCallBk, 456
 - ctls_registerCallBkp, 456
 - ctls_removeAllApplicationData, 456
 - ctls_removeAllCAPK, 456
 - ctls_removeApplicationData, 457
 - ctls_removeCAPK, 457
 - ctls_removeConfigurationGroup, 457
 - ctls_retrieveAIDList, 458
 - ctls_retrieveApplicationData, 458
 - ctls_retrieveCAPKList, 459
 - ctls_retrieveCAPK, 458
 - ctls_retrieveTerminalData, 460
 - ctls_setApplicationData, 460
 - ctls_setCAPK, 460
 - ctls_setConfigurationGroup, 461
 - ctls_setTerminalData, 461
 - ctls_startTransaction, 462
 - device_SendDataCommandNEO, 498
 - device_activateTransaction, 463
 - device_buzzerOnOff, 465
 - device_cancelTransaction, 465
 - device_cancelTransactionSilent, 465
 - device_close, 466
 - device_configureButtons, 466
 - device_controlUserInterface, 466
 - device_deleteDirectory, 467
 - device_deleteFile, 468
 - device_disableBlueLED, 468
 - device_enableBlueLED, 468
 - device_enableExternalLCDMessages, 469
 - device_enableL100PassThrough, 469
 - device_enableL80PassThrough, 470
 - device_enablePassThrough, 470
 - device_enableRFAntenna, 470
 - device_getAudioVolume, 471
 - device_getButtonConfiguration, 471
 - device_getCameraParameters, 471
 - device_getCurrentDeviceType, 472
 - device_getDRS, 473
 - device_getDeviceMemoryUsageInfo, 472
 - device_getDeviceTreeVersion, 472
 - device_getFirmwareVersion, 473
 - device_getFirmwareVersion_Len, 474
 - device_getIDGStatusCodeString, 474
 - device_getKeyStatus, 476
 - device_getL100PassThroughMode, 477
 - device_getL80PassThroughMode, 477
 - device_getMerchantRecord, 477
 - device_getMerchantRecord_Len, 478
 - device_getNEOAltDevice, 478
 - device_getResponseCodeString, 478
 - device_getSDKWaitTime, 488
 - device_getTamperStatus, 489
 - device_getThreadStackSize, 489
 - device_getTransactionResults, 489
 - device_init, 490
 - device_isAttached, 490
 - device_isConnected, 490
 - device_lcdDisplayClear, 490
 - device_lcdDisplayLine1Message, 491
 - device_lcdDisplayLine2Message, 491
 - device_listDirectory, 491
 - device_loadCertCA, 492
 - device_outputLog, 492
 - device_pingDevice, 493
 - device_playAudio, 493
 - device_pollForToken, 493
 - device_queryFile, 494
 - device_readFileFromSD, 494
 - device_rebootDevice, 495
 - device_registerCameraCallBk, 495
 - device_registerCardStatusFrontSwitchCallBk, 495
 - device_registerFWCallBk, 495
 - device_registerRKICallBk, 496
 - device_rrcConnect, 496
 - device_rrcDisconnect, 496
 - device_rrcDownloadApp, 496
 - device_rrcInstallApp, 497
 - device_rrcRunApp, 497
 - device_rrcUninstallApp, 497
 - device_setAudioVolume, 498
 - device_setBurstMode, 498
 - device_setCameraParameters, 499
 - device_setCancelTransactionMode, 499
 - device_setConfigPath, 499
 - device_setCoreDumpLogFile, 500
 - device_setCurrentDevice, 500
 - device_setMerchantRecord, 501
 - device_setNEO2DevicesConfigs, 501
 - device_setNEOAltDevice, 502
 - device_setNEOGen, 502

- device_setPollMode, 502
- device_setRKI_URL, 502
- device_setSDKWaitTime, 504
- device_setThreadStackSize, 504
- device_setTransactionExponent, 504
- device_startListenNotifications, 504
- device_startQRCodeScan, 505
- device_startQRCodeScanWithDisplayWindowInfo, 505
- device_startRKI, 505
- device_startTakingPhoto, 506
- device_startTransaction, 506
- device_stopAudio, 508
- device_stopListenNotifications, 508
- device_stopQRCodeScan, 508
- device_stopTakingPhoto, 508
- device_toSDCard, 509
- device_transferFile, 509
- device_turnOffYellowLED, 510
- device_turnOnYellowLED, 510
- device_updateFirmware, 510
- emv_activateTransaction, 511
- emv_allowFallback, 511
- emv_authenticateTransaction, 512
- emv_authenticateTransactionWithTimeout, 512
- emv_cancelTransaction, 513
- emv_completeTransaction, 513
- emv_getAutoAuthenticateTransaction, 515
- emv_getAutoCompleteTransaction, 515
- emv_getEMVConfigurationCheckValue, 515
- emv_getEMVKernelCheckValue, 516
- emv_getEMVKernelVersion, 516
- emv_getEMVKernelVersion_Len, 516
- emv_registerCallBk, 517
- emv_removeAllApplicationData, 517
- emv_removeAllCAPK, 517
- emv_removeAllCRL, 517
- emv_removeApplicationData, 517
- emv_removeCAPK, 519
- emv_removeCRL, 519
- emv_retrieveAIDList, 519
- emv_retrieveApplicationData, 520
- emv_retrieveCAPKList, 521
- emv_retrieveCAPK, 520
- emv_retrieveCRL, 521
- emv_retrieveTerminalData, 522
- emv_retrieveTransactionResult, 522
- emv_setApplicationData, 523
- emv_setApplicationDataTLV, 523
- emv_setAutoAuthenticateTransaction, 523
- emv_setAutoCompleteTransaction, 524
- emv_setCAPK, 524
- emv_setCRL, 525
- emv_setTerminalData, 525
- emv_setTerminalMajorConfiguration, 525
- emv_setTransactionParameters, 526
- emv_startTransaction, 526
- executeTransaction, 527
- executeTransaction_WorldNet, 528
- felica_SendCommand, 531
- felica_authentication, 528
- felica_cancelCodeEntry, 529
- felica_getCode, 529
- felica_poll, 529
- felica_read, 530
- felica_readWithMac, 530
- felica_requestService, 531
- felica_write, 531
- felica_writeWithMac, 532
- forwardTransaction, 532
- forwardTransaction_WorldNet, 533
- ftpComm_callBack, 448
- httpComm_callBack, 448
- IN_OUT, 448
- icc_exchangeAPDU, 534
- icc_getICCRReaderStatus, 534
- icc_powerOffICC, 534
- icc_powerOnICC, 535
- IN, 448
- lcd_addButton, 535
- lcd_addEthernet, 536
- lcd_addExtVideo, 537
- lcd_addImage, 538
- lcd_addLED, 539
- lcd_addText, 541
- lcd_addVideo, 543
- lcd_clearDisplay, 544
- lcd_clearScreenInfo, 545
- lcd_cloneScreen, 545
- lcd_createScreen, 545
- lcd_destroyScreen, 546
- lcd_displayMessage, 546
- lcd_getActiveScreen, 546
- lcd_getAllObjects, 547
- lcd_getAllScreens, 547
- lcd_getButtonEvent, 548
- lcd_linkUIWithTransactionMessageId, 548
- lcd_loadScreenInfo, 549
- lcd_queryObjectbyID, 549
- lcd_queryObjectbyName, 549
- lcd_queryScreenbyID, 550
- lcd_queryScreenbyName, 550
- lcd_registerCallBk, 551
- lcd_removeItem, 551
- lcd_setBacklight, 551
- lcd_showScreen, 552
- lcd_storeScreenInfo, 552
- lcd_updateColor, 552
- lcd_updateLabel, 553
- lcd_updatePosition, 554
- loyalty_cancelTransaction, 554
- loyalty_cancelTransactionSilent, 554
- loyalty_registerCallBk, 555
- loyalty_startTransaction, 555
- msr_cancelMSRSwipe, 557
- msr_registerCallBk, 557

- msr_registerCallBkp, 557
- msr_startMSRSwipe, 557
- OUT, 448
- pCMR_callBack, 448
- pCSFS_callBack, 448
- pEMV_callBack, 448
- pFW_callBack, 449
- pLCD_callBack, 449
- pMSR_callBack, 449
- pMSR_callBackp, 449
- pMessageHotplug, 449
- pPIN_callBack, 449
- pRKI_callBack, 450
- pReadDataLog, 449
- pSendDataLog, 450
- pWN_callBack, 450
- pWP_callBack, 450
- parseMSRData, 558
- pin_cancelPINEntry, 558
- pin_capturePin, 558
- pin_capturePinExt, 559
- pin_getPanEntry, 560
- pin_inputFromPrompt, 561
- pin_promptForNumericKey, 562
- pin_promptForNumericKeyWithSwipe, 562
- pin_registerCallBk, 563
- pin_setKeyValues, 563
- registerHotplugCallBk, 564
- registerLogCallBk, 564
- rs232_device_init, 564
- SDK_Version, 566
- set_open_com_port_timeout, 566
- setAbsoluteLibraryPath, 566
- v4Comm_callBack, 450
- libIDT_PipReader.h
 - config_getSerialNumber, 570
 - config_getSerialNumber_Len, 571
 - ctls_activateTransaction, 571
 - ctls_cancelTransaction, 572
 - ctls_getAllConfigurationGroups, 572
 - ctls_getConfigurationGroup, 573
 - ctls_registerCallBk, 573
 - ctls_registerCallBkp, 573
 - ctls_removeAllApplicationData, 573
 - ctls_removeAllCAPK, 574
 - ctls_removeApplicationData, 574
 - ctls_removeCAPK, 574
 - ctls_removeConfigurationGroup, 575
 - ctls_retrieveAIDList, 575
 - ctls_retrieveApplicationData, 575
 - ctls_retrieveCAPKList, 577
 - ctls_retrieveCAPK, 576
 - ctls_retrieveTerminalData, 577
 - ctls_setApplicationData, 577
 - ctls_setCAPK, 578
 - ctls_setConfigurationGroup, 578
 - ctls_setTerminalData, 579
 - ctls_startTransaction, 579
- device_SendDataCommandNEO, 588
- device_close, 581
- device_controlUserInterface, 581
- device_enablePassThrough, 582
- device_getCurrentDeviceType, 583
- device_getFirmwareVersion, 583
- device_getFirmwareVersion_Len, 583
- device_getIDGStatusCodeString, 584
- device_getMerchantRecord, 585
- device_getMerchantRecord_Len, 586
- device_getSDKWaitTime, 586
- device_getTransactionResults, 586
- device_init, 587
- device_isAttached, 587
- device_isConnected, 587
- device_pingDevice, 588
- device_registerCameraCallBk, 588
- device_registerCardStatusFrontSwitchCallBk, 588
- device_setBurstMode, 589
- device_setCurrentDevice, 589
- device_setMerchantRecord, 590
- device_setPollMode, 590
- device_setSDKWaitTime, 590
- emv_registerCallBk, 591
- ftpComm_callBack, 569
- httpComm_callBack, 569
- IN_OUT, 568
- IN, 568
- OUT, 568
- pCMR_callBack, 569
- pCSFS_callBack, 569
- pEMV_callBack, 569
- pMSR_callBack, 569
- pMSR_callBackp, 569
- pMessageHotplug, 569
- pPIN_callBack, 570
- pReadDataLog, 570
- pSendDataLog, 570
- parseMSRData, 591
- pin_registerCallBk, 591
- registerHotplugCallBk, 591
- registerLogCallBk, 591
- rs232_device_init, 591
- SDK_Version, 592
- setAbsoluteLibraryPath, 592
- v4Comm_callBack, 570
- libIDT_SREDKey2.h
 - comm_registerHTTPCallback, 645
 - comm_registerV4Callback, 645
 - config_getModelNumber, 645
 - config_getModelNumber_Len, 646
 - config_getSerialNumber, 646
 - config_getSerialNumber_Len, 646
 - ctls_registerCallBk, 647
 - ctls_registerCallBkp, 647
 - device_SendDataCommand, 652
 - device_SendDataCommandITP, 653
 - device_SendDataCommandNEO, 653

- device_close, 647
- device_getCurrentDeviceType, 647
- device_getFirmwareVersion, 647
- device_getFirmwareVersion_Len, 648
- device_getIDGStatusCodeString, 648
- device_getKeyStatus, 649
- device_init, 650
- device_isAttached, 651
- device_isConnected, 651
- device_pingDevice, 651
- device_rebootDevice, 651
- device_registerCameraCallBk, 652
- device_registerCardStatusFrontSwitchCallBk, 652
- device_registerFWCallBk, 652
- device_setConfigPath, 654
- device_setCurrentDevice, 654
- device_setNEO2DevicesConfigs, 654
- device_setSystemLanguage, 655
- device_setTransactionExponent, 655
- device_updateFirmware, 655
- emv_registerCallBk, 656
- ftpComm_callBack, 643
- httpComm_callBack, 643
- IN_OUT, 643
- IN, 642
- lcd_registerCallBk, 656
- msr_disable, 656
- msr_getClearPANID, 657
- msr_getExpirationMask, 657
- msr_getFunctionStatus, 657
- msr_getSwipeForcedEncryptionOption, 658
- msr_getSwipeMaskOption, 658
- msr_registerCallBk, 658
- msr_registerCallBkp, 658
- msr_setClearPANID, 659
- msr_setExpirationMask, 659
- msr_setSwipeForcedEncryptionOption, 659
- msr_setSwipeMaskOption, 660
- OUT, 643
- pCMR_callBack, 643
- pCSFS_callBack, 643
- pEMV_callBack, 643
- pFW_callBack, 644
- pLCD_callBack, 644
- pMSR_callBack, 644
- pMSR_callBackp, 644
- pMessageHotplug, 644
- pPIN_callBack, 644
- pReadDataLog, 644
- pSendDataLog, 644
- pin_registerCallBk, 660
- registerHotplugCallBk, 660
- registerLogCallBk, 660
- rs232_device_init, 660
- SDK_Version, 661
- setAbsoluteLibraryPath, 661
- v4Comm_callBack, 645
- libIDT_SpectrumPro.h
- config_getModelNumber, 597
- config_getModelNumber_Len, 597
- config_getSerialNumber, 598
- config_getSerialNumber_Len, 598
- device_SendDataCommand, 616
- device_close, 598
- device_getCurrentDeviceType, 599
- device_getFirmwareVersion, 599
- device_getFirmwareVersion_Len, 599
- device_getResponseCodeString, 600
- device_getSDKWaitTime, 610
- device_getSpectrumProKSN_Len, 610
- device_getSpectrumProKSN, 610
- device_getThreadStackSize, 611
- device_init, 611
- device_isAttached, 611
- device_isConnected, 612
- device_pollCardReader, 612
- device_pollCardReader_Len, 614
- device_rebootDevice, 616
- device_registerCameraCallBk, 616
- device_registerCardStatusFrontSwitchCallBk, 616
- device_setCurrentDevice, 617
- device_setSDKWaitTime, 617
- device_setThreadStackSize, 617
- device_updateFirmware, 617
- emv_activateTransaction, 618
- emv_allowFallback, 619
- emv_authenticateTransaction, 619
- emv_authenticateTransactionWithTimeout, 620
- emv_callbackResponseLCD, 620
- emv_callbackResponseMSR, 621
- emv_cancelTransaction, 621
- emv_completeTransaction, 622
- emv_getAutoAuthenticateTransaction, 622
- emv_getAutoCompleteTransaction, 622
- emv_getEMVConfigurationCheckValue, 623
- emv_getEMVKernelCheckValue, 623
- emv_getEMVKernelVersion, 623
- emv_getEMVKernelVersion_Len, 624
- emv_registerCallBk, 624
- emv_removeAllApplicationData, 624
- emv_removeAllCAPK, 624
- emv_removeAllCRL, 625
- emv_removeApplicationData, 625
- emv_removeCAPK, 625
- emv_removeCRL, 626
- emv_removeTerminalData, 626
- emv_retrieveAIDList, 626
- emv_retrieveApplicationData, 627
- emv_retrieveCAPKList, 628
- emv_retrieveCAPK, 627
- emv_retrieveCRL, 628
- emv_retrieveTerminalData, 629
- emv_retrieveTerminalID_Len, 629
- emv_retrieveTerminalID, 629
- emv_retrieveTransactionResult, 630
- emv_setApplicationData, 630

- emv_setAutoAuthenticateTransaction, 631
- emv_setAutoCompleteTransaction, 631
- emv_setCAPK, 631
- emv_setCRL, 632
- emv_setTerminalData, 632
- emv_setTerminalID, 634
- emv_startTransaction, 634
- ftpComm_callBack, 595
- httpComm_callBack, 596
- IN_OUT, 595
- icc_getICCRReaderStatus, 635
- icc_powerOffICC, 635
- icc_powerOnICC, 635
- IN, 595
- msr_cancelMSRSwipe, 636
- msr_clearMSRData, 636
- msr_getMSRData, 636
- msr_registerCallBk, 636
- msr_registerCallBkp, 637
- msr_startMSRSwipe, 637
- OUT, 595
- pCMR_callBack, 596
- pCSFS_callBack, 596
- pEMV_callBack, 596
- pMSR_callBack, 596
- pMSR_callBackp, 596
- pMessageHotplug, 596
- pPIN_callBack, 597
- pReadDataLog, 597
- pSendDataLog, 597
- parseMSRData, 637
- parsePINBlockData, 637
- parsePINData, 638
- pin_cancelPINEntry, 638
- pin_getPIN, 638
- pin_registerCallBk, 639
- registerHotplugCallBk, 639
- registerLogCallBk, 639
- rs232_device_init, 639
- SDK_Version, 640
- setAbsoluteLibraryPath, 640
- v4Comm_callBack, 597
- libIDT_UniPayI_V.h
 - comm_registerHTTPCallback, 666
 - comm_registerV4Callback, 666
 - config_getSerialNumber, 666
 - config_getSerialNumber_Len, 667
 - device_SendDataCommandNEO, 673
 - device_close, 667
 - device_enablePassThrough, 667
 - device_getCurrentDeviceType, 667
 - device_getFirmwareVersion, 668
 - device_getFirmwareVersion_Len, 668
 - device_getIDGStatusCodeString, 668
 - device_getMerchantRecord, 670
 - device_getMerchantRecord_Len, 670
 - device_getSDKWaitTime, 671
 - device_getThreadStackSize, 671
 - device_init, 671
 - device_isAttached, 672
 - device_isConnected, 672
 - device_pingDevice, 672
 - device_registerCameraCallBk, 672
 - device_registerCardStatusFrontSwitchCallBk, 673
 - device_setCurrentDevice, 674
 - device_setMerchantRecord, 674
 - device_setSDKWaitTime, 674
 - device_setThreadStackSize, 675
 - emv_activateTransaction, 675
 - emv_allowFallback, 675
 - emv_authenticateTransaction, 676
 - emv_authenticateTransactionWithTimeout, 676
 - emv_cancelTransaction, 677
 - emv_completeTransaction, 677
 - emv_getAutoAuthenticateTransaction, 678
 - emv_getAutoCompleteTransaction, 678
 - emv_registerCallBk, 678
 - emv_removeAllApplicationData, 678
 - emv_removeAllCAPK, 679
 - emv_removeAllCRL, 679
 - emv_removeApplicationData, 679
 - emv_removeCAPK, 679
 - emv_removeCRL, 680
 - emv_retrieveAIDList, 680
 - emv_retrieveApplicationData, 681
 - emv_retrieveCAPKList, 682
 - emv_retrieveCAPK, 681
 - emv_retrieveCRL, 682
 - emv_retrieveTerminalData, 683
 - emv_setApplicationData, 683
 - emv_setApplicationDataTLV, 684
 - emv_setAutoAuthenticateTransaction, 684
 - emv_setAutoCompleteTransaction, 684
 - emv_setCAPK, 684
 - emv_setCRL, 685
 - emv_setTerminalData, 685
 - emv_setTerminalMajorConfiguration, 686
 - emv_startTransaction, 686
 - ftpComm_callBack, 664
 - httpComm_callBack, 664
 - IN_OUT, 664
 - icc_exchangeAPDU, 687
 - icc_getICCRReaderStatus, 688
 - icc_powerOffICC, 688
 - icc_powerOnICC, 688
 - IN, 664
 - msr_cancelMSRSwipe, 689
 - msr_registerCallBk, 689
 - msr_registerCallBkp, 689
 - msr_startMSRSwipe, 689
 - OUT, 664
 - pCMR_callBack, 664
 - pCSFS_callBack, 664
 - pEMV_callBack, 665
 - pMSR_callBack, 665
 - pMSR_callBackp, 665

- pMessageHotplug, [665](#)
- pPIN_callBack, [665](#)
- pReadDataLog, [665](#)
- pSendDataLog, [665](#)
- parseMSRData, [689](#)
- pin_registerCallBk, [690](#)
- registerHotplugCallBk, [690](#)
- registerLogCallBk, [690](#)
- SDK_Version, [690](#)
- setAbsoluteLibraryPath, [690](#)
- v4Comm_callBack, [666](#)
- libIDT_VP3300_AJ.h
 - cancelWorldNet, [724](#)
 - cancelWorldPay, [724](#)
 - comm_registerHTTPCallback, [724](#)
 - comm_registerV4Callback, [725](#)
 - config_getSerialNumber, [725](#)
 - config_getSerialNumber_Len, [725](#)
 - ctls_activateTransaction, [726](#)
 - ctls_cancelTransaction, [727](#)
 - ctls_getAllConfigurationGroups, [727](#)
 - ctls_getConfigurationGroup, [727](#)
 - ctls_registerCallBk, [728](#)
 - ctls_registerCallBkp, [728](#)
 - ctls_removeAllApplicationData, [728](#)
 - ctls_removeAllCAPK, [728](#)
 - ctls_removeApplicationData, [729](#)
 - ctls_removeCAPK, [729](#)
 - ctls_removeConfigurationGroup, [729](#)
 - ctls_retrieveAIDList, [730](#)
 - ctls_retrieveApplicationData, [730](#)
 - ctls_retrieveCAPKList, [731](#)
 - ctls_retrieveCAPK, [730](#)
 - ctls_retrieveTerminalData, [732](#)
 - ctls_setApplicationData, [732](#)
 - ctls_setCAPK, [732](#)
 - ctls_setConfigurationGroup, [733](#)
 - ctls_setTerminalData, [733](#)
 - ctls_startTransaction, [734](#)
 - device_SendDataCommandNEO, [745](#)
 - device_activateTransaction, [735](#)
 - device_cancelTransaction, [737](#)
 - device_close, [737](#)
 - device_controlUserInterface, [737](#)
 - device_enablePassThrough, [738](#)
 - device_getCurrentDeviceType, [739](#)
 - device_getFirmwareVersion, [739](#)
 - device_getFirmwareVersion_Len, [739](#)
 - device_getIDGStatusCodeString, [740](#)
 - device_getMerchantRecord, [741](#)
 - device_getMerchantRecord_Len, [742](#)
 - device_getRTCDateTime, [742](#)
 - device_getSDKWaitTime, [743](#)
 - device_getThreadStackSize, [743](#)
 - device_getTransactionResults, [743](#)
 - device_init, [743](#)
 - device_isAttached, [744](#)
 - device_isConnected, [744](#)
 - device_pingDevice, [744](#)
 - device_pollForToken, [744](#)
 - device_registerCameraCallBk, [745](#)
 - device_registerCardStatusFrontSwitchCallBk, [745](#)
 - device_registerRKICallBk, [745](#)
 - device_setBurstMode, [746](#)
 - device_setCurrentDevice, [746](#)
 - device_setMerchantRecord, [747](#)
 - device_setPollMode, [747](#)
 - device_setRKI_URL, [748](#)
 - device_setRTCDateTime, [748](#)
 - device_setSDKWaitTime, [748](#)
 - device_setThreadStackSize, [749](#)
 - device_setTransactionExponent, [749](#)
 - device_startRKI, [749](#)
 - device_startTransaction, [750](#)
 - emv_activateTransaction, [751](#)
 - emv_allowFallback, [752](#)
 - emv_authenticateTransaction, [752](#)
 - emv_authenticateTransactionWithTimeout, [752](#)
 - emv_cancelTransaction, [753](#)
 - emv_completeTransaction, [753](#)
 - emv_getAutoAuthenticateTransaction, [754](#)
 - emv_getAutoCompleteTransaction, [754](#)
 - emv_registerCallBk, [754](#)
 - emv_removeAllApplicationData, [754](#)
 - emv_removeAllCAPK, [755](#)
 - emv_removeAllCRL, [755](#)
 - emv_removeApplicationData, [755](#)
 - emv_removeCAPK, [756](#)
 - emv_removeCRL, [756](#)
 - emv_retrieveAIDList, [756](#)
 - emv_retrieveApplicationData, [757](#)
 - emv_retrieveCAPKList, [758](#)
 - emv_retrieveCAPK, [757](#)
 - emv_retrieveCRL, [758](#)
 - emv_retrieveTerminalData, [759](#)
 - emv_setApplicationData, [759](#)
 - emv_setApplicationDataTLV, [760](#)
 - emv_setAutoAuthenticateTransaction, [760](#)
 - emv_setAutoCompleteTransaction, [760](#)
 - emv_setCAPK, [760](#)
 - emv_setCRL, [761](#)
 - emv_setTerminalData, [761](#)
 - emv_setTerminalMajorConfiguration, [762](#)
 - emv_setTransactionParameters, [762](#)
 - emv_startTransaction, [763](#)
 - executeTransaction, [764](#)
 - executeTransaction_WorldNet, [764](#)
 - forwardTransaction, [765](#)
 - forwardTransaction_WorldNet, [765](#)
 - ftpComm_callBack, [722](#)
 - httpComm_callBack, [722](#)
 - IN_OUT, [722](#)
 - icc_exchangeAPDU, [766](#)
 - icc_getICCRReaderStatus, [766](#)
 - icc_powerOffICC, [768](#)
 - icc_powerOnICC, [768](#)

- IN, 721
- msr_cancelMSRSwipe, 768
- msr_registerCallBk, 768
- msr_registerCallBkp, 769
- msr_startMSRSwipe, 769
- OUT, 722
- pCMR_callBack, 722
- pCSFS_callBack, 722
- pEMV_callBack, 722
- pMSR_callBack, 723
- pMSR_callBackp, 723
- pMessageHotplug, 723
- pPIN_callBack, 723
- pRKI_callBack, 723
- pReadDataLog, 723
- pSendDataLog, 723
- pWN_callBack, 723
- pWP_callBack, 724
- parseMSRData, 769
- pin_registerCallBk, 769
- registerHotplugCallBk, 769
- registerLogCallBk, 770
- SDK_Version, 770
- setAbsoluteLibraryPath, 770
- v4Comm_callBack, 724
- libIDT_VP3300_BT.h
 - cancelWorldNet, 776
 - cancelWorldPay, 776
 - comm_registerHTTPCallback, 776
 - comm_registerV4Callback, 776
 - config_getSerialNumber, 777
 - config_getSerialNumber_Len, 777
 - ctls_activateTransaction, 777
 - ctls_cancelTransaction, 778
 - ctls_getAllConfigurationGroups, 779
 - ctls_getConfigurationGroup, 779
 - ctls_registerCallBk, 779
 - ctls_registerCallBkp, 780
 - ctls_removeAllApplicationData, 780
 - ctls_removeAllCAPK, 780
 - ctls_removeApplicationData, 780
 - ctls_removeCAPK, 781
 - ctls_removeConfigurationGroup, 781
 - ctls_retrieveAIDList, 781
 - ctls_retrieveApplicationData, 782
 - ctls_retrieveCAPKList, 783
 - ctls_retrieveCAPK, 782
 - ctls_retrieveTerminalData, 783
 - ctls_setApplicationData, 784
 - ctls_setCAPK, 784
 - ctls_setConfigurationGroup, 785
 - ctls_setTerminalData, 785
 - ctls_startTransaction, 786
 - device_SendDataCommandNEO, 797
 - device_activateTransaction, 787
 - device_cancelTransaction, 789
 - device_close, 789
 - device_controlUserInterface, 789
 - device_enablePassThrough, 790
 - device_getCurrentDeviceType, 791
 - device_getFirmwareVersion, 791
 - device_getFirmwareVersion_Len, 791
 - device_getIDGStatusCodeString, 792
 - device_getMerchantRecord, 793
 - device_getMerchantRecord_Len, 794
 - device_getRTCDatetime, 794
 - device_getSDKWaitTime, 795
 - device_getThreadStackSize, 795
 - device_getTransactionResults, 795
 - device_init, 795
 - device_isAttached, 796
 - device_isConnected, 796
 - device_pingDevice, 796
 - device_pollForToken, 796
 - device_registerCameraCallBk, 797
 - device_registerCardStatusFrontSwitchCallBk, 797
 - device_registerRKICallBk, 797
 - device_setBurstMode, 798
 - device_setCurrentDevice, 798
 - device_setMerchantRecord, 799
 - device_setPollMode, 799
 - device_setRKI_URL, 800
 - device_setRTCDatetime, 800
 - device_setSDKWaitTime, 800
 - device_setThreadStackSize, 801
 - device_setTransactionExponent, 801
 - device_startRKI, 801
 - device_startTransaction, 802
 - emv_activateTransaction, 803
 - emv_allowFallback, 804
 - emv_authenticateTransaction, 804
 - emv_authenticateTransactionWithTimeout, 804
 - emv_cancelTransaction, 805
 - emv_completeTransaction, 805
 - emv_getAutoAuthenticateTransaction, 806
 - emv_getAutoCompleteTransaction, 806
 - emv_registerCallBk, 806
 - emv_removeAllApplicationData, 806
 - emv_removeAllCAPK, 807
 - emv_removeAllCRL, 807
 - emv_removeApplicationData, 807
 - emv_removeCAPK, 808
 - emv_removeCRL, 808
 - emv_retrieveAIDList, 808
 - emv_retrieveApplicationData, 809
 - emv_retrieveCAPKList, 810
 - emv_retrieveCAPK, 809
 - emv_retrieveCRL, 810
 - emv_retrieveTerminalData, 811
 - emv_setApplicationData, 811
 - emv_setApplicationDataTLV, 812
 - emv_setAutoAuthenticateTransaction, 812
 - emv_setAutoCompleteTransaction, 812
 - emv_setCAPK, 812
 - emv_setCRL, 813
 - emv_setTerminalData, 813

- emv_setTerminalMajorConfiguration, [814](#)
- emv_setTransactionParameters, [814](#)
- emv_startTransaction, [815](#)
- executeTransaction, [816](#)
- executeTransaction_WorldNet, [816](#)
- forwardTransaction, [817](#)
- forwardTransaction_WorldNet, [817](#)
- ftpComm_callBack, [774](#)
- httpComm_callBack, [774](#)
- IN_OUT, [773](#)
- icc_exchangeAPDU, [818](#)
- icc_getICCRReaderStatus, [818](#)
- icc_powerOffICC, [820](#)
- icc_powerOnICC, [820](#)
- IN, [773](#)
- msr_cancelMSRSwipe, [820](#)
- msr_registerCallBk, [820](#)
- msr_registerCallBkp, [821](#)
- msr_startMSRSwipe, [821](#)
- OUT, [773](#)
- pCMR_callBack, [774](#)
- pCSFS_callBack, [774](#)
- pEMV_callBack, [774](#)
- pMSR_callBack, [774](#)
- pMSR_callBackp, [775](#)
- pMessageHotplug, [774](#)
- pPIN_callBack, [775](#)
- pRKI_callBack, [775](#)
- pReadDataLog, [775](#)
- pSendDataLog, [775](#)
- pWN_callBack, [775](#)
- pWP_callBack, [775](#)
- parseMSRData, [821](#)
- pin_registerCallBk, [821](#)
- registerHotplugCallBk, [821](#)
- registerLogCallBk, [822](#)
- SDK_Version, [822](#)
- setAbsoluteLibraryPath, [822](#)
- v4Comm_callBack, [775](#)
- libIDT_VP3300_COM.h
 - cancelWorldNet, [828](#)
 - cancelWorldPay, [828](#)
 - comm_registerHTTPCallback, [828](#)
 - comm_registerV4Callback, [828](#)
 - config_getSerialNumber, [829](#)
 - config_getSerialNumber_Len, [829](#)
 - ctls_activateTransaction, [829](#)
 - ctls_cancelTransaction, [830](#)
 - ctls_getAllConfigurationGroups, [831](#)
 - ctls_getConfigurationGroup, [831](#)
 - ctls_registerCallBk, [831](#)
 - ctls_registerCallBkp, [832](#)
 - ctls_removeAllApplicationData, [832](#)
 - ctls_removeAllCAPK, [832](#)
 - ctls_removeApplicationData, [832](#)
 - ctls_removeCAPK, [833](#)
 - ctls_removeConfigurationGroup, [833](#)
 - ctls_retrieveAIDList, [833](#)
 - ctls_retrieveApplicationData, [834](#)
 - ctls_retrieveCAPKList, [835](#)
 - ctls_retrieveCAPK, [834](#)
 - ctls_retrieveTerminalData, [835](#)
 - ctls_setApplicationData, [836](#)
 - ctls_setCAPK, [836](#)
 - ctls_setConfigurationGroup, [837](#)
 - ctls_setTerminalData, [837](#)
 - ctls_startTransaction, [838](#)
 - device_SendDataCommandNEO, [849](#)
 - device_activateTransaction, [839](#)
 - device_cancelTransaction, [841](#)
 - device_close, [841](#)
 - device_controlUserInterface, [841](#)
 - device_enablePassThrough, [842](#)
 - device_getCurrentDeviceType, [843](#)
 - device_getFirmwareVersion, [843](#)
 - device_getFirmwareVersion_Len, [843](#)
 - device_getIDGStatusCodeString, [844](#)
 - device_getMerchantRecord, [845](#)
 - device_getMerchantRecord_Len, [846](#)
 - device_getRTCTime, [846](#)
 - device_getSDKWaitTime, [847](#)
 - device_getThreadStackSize, [847](#)
 - device_getTransactionResults, [847](#)
 - device_init, [847](#)
 - device_isAttached, [848](#)
 - device_isConnected, [848](#)
 - device_pingDevice, [848](#)
 - device_pollForToken, [848](#)
 - device_registerCameraCallBk, [849](#)
 - device_registerCardStatusFrontSwitchCallBk, [849](#)
 - device_registerRKICallBk, [849](#)
 - device_setBurstMode, [850](#)
 - device_setCurrentDevice, [850](#)
 - device_setMerchantRecord, [851](#)
 - device_setPollMode, [851](#)
 - device_setRKI_URL, [852](#)
 - device_setRTCTime, [852](#)
 - device_setSDKWaitTime, [852](#)
 - device_setThreadStackSize, [853](#)
 - device_setTransactionExponent, [853](#)
 - device_startRKI, [853](#)
 - device_startTransaction, [854](#)
 - emv_activateTransaction, [854](#)
 - emv_allowFallback, [855](#)
 - emv_authenticateTransaction, [855](#)
 - emv_authenticateTransactionWithTimeout, [856](#)
 - emv_cancelTransaction, [856](#)
 - emv_completeTransaction, [856](#)
 - emv_getAutoAuthenticateTransaction, [857](#)
 - emv_getAutoCompleteTransaction, [857](#)
 - emv_registerCallBk, [858](#)
 - emv_removeAllApplicationData, [858](#)
 - emv_removeAllCAPK, [858](#)
 - emv_removeAllCRL, [858](#)
 - emv_removeApplicationData, [858](#)
 - emv_removeCAPK, [860](#)

- emv_removeCRL, 860
- emv_retrieveAIDList, 860
- emv_retrieveApplicationData, 861
- emv_retrieveCAPKList, 862
- emv_retrieveCAPK, 861
- emv_retrieveCRL, 862
- emv_retrieveTerminalData, 863
- emv_setApplicationData, 863
- emv_setApplicationDataTLV, 864
- emv_setAutoAuthenticateTransaction, 864
- emv_setAutoCompleteTransaction, 864
- emv_setCAPK, 865
- emv_setCRL, 865
- emv_setTerminalData, 866
- emv_setTerminalMajorConfiguration, 866
- emv_setTransactionParameters, 866
- emv_startTransaction, 867
- executeTransaction, 868
- executeTransaction_WorldNet, 868
- forwardTransaction, 869
- forwardTransaction_WorldNet, 869
- ftpComm_callBack, 826
- httpComm_callBack, 826
- IN_OUT, 825
- icc_exchangeAPDU, 870
- icc_getICCRReaderStatus, 870
- icc_powerOffICC, 871
- icc_powerOnICC, 871
- IN, 825
- msr_cancelMSRSwipe, 871
- msr_registerCallBk, 871
- msr_registerCallBkp, 872
- msr_startMSRSwipe, 872
- OUT, 825
- pCMR_callBack, 826
- pCSFS_callBack, 826
- pEMV_callBack, 826
- pMSR_callBack, 826
- pMSR_callBkp, 827
- pMessageHotplug, 826
- pPIN_callBack, 827
- pRKI_callBack, 827
- pReadDataLog, 827
- pSendDataLog, 827
- pWN_callBack, 827
- pWP_callBack, 827
- parseMSRData, 872
- pin_registerCallBk, 872
- registerHotplugCallBk, 872
- registerLogCallBk, 873
- rs232_device_init, 873
- SDK_Version, 873
- setAbsoluteLibraryPath, 874
- v4Comm_callBack, 827
- libIDT_VP3300_USB.h
 - cancelWorldNet, 880
 - cancelWorldPay, 880
 - comm_registerHTTPCallback, 880
 - comm_registerV4Callback, 880
 - config_getSerialNumber, 880
 - config_getSerialNumber_Len, 881
 - ctls_activateTransaction, 881
 - ctls_cancelTransaction, 882
 - ctls_getAllConfigurationGroups, 883
 - ctls_getConfigurationGroup, 883
 - ctls_registerCallBk, 883
 - ctls_registerCallBkp, 884
 - ctls_removeAllApplicationData, 884
 - ctls_removeAllCAPK, 884
 - ctls_removeApplicationData, 884
 - ctls_removeCAPK, 885
 - ctls_removeConfigurationGroup, 885
 - ctls_retrieveAIDList, 885
 - ctls_retrieveApplicationData, 886
 - ctls_retrieveCAPKList, 887
 - ctls_retrieveCAPK, 886
 - ctls_retrieveTerminalData, 887
 - ctls_setApplicationData, 888
 - ctls_setCAPK, 888
 - ctls_setConfigurationGroup, 889
 - ctls_setTerminalData, 889
 - ctls_startTransaction, 890
 - device_SendDataCommandNEO, 901
 - device_activateTransaction, 891
 - device_cancelTransaction, 893
 - device_close, 893
 - device_controlUserInterface, 893
 - device_enablePassThrough, 894
 - device_getCurrentDeviceType, 895
 - device_getFirmwareVersion, 895
 - device_getFirmwareVersion_Len, 895
 - device_getIDGStatusCodeString, 896
 - device_getMerchantRecord, 897
 - device_getMerchantRecord_Len, 898
 - device_getRTCDateTime, 898
 - device_getSDKWaitTime, 899
 - device_getThreadStackSize, 899
 - device_getTransactionResults, 899
 - device_init, 899
 - device_isAttached, 900
 - device_isConnected, 900
 - device_pingDevice, 900
 - device_pollForToken, 900
 - device_registerCameraCallBk, 901
 - device_registerCardStatusFrontSwitchCallBk, 901
 - device_registerRKICallBk, 901
 - device_setBurstMode, 902
 - device_setCurrentDevice, 902
 - device_setMerchantRecord, 903
 - device_setPollMode, 903
 - device_setRKI_URL, 904
 - device_setRTCDateTime, 904
 - device_setSDKWaitTime, 904
 - device_setThreadStackSize, 905
 - device_setTransactionExponent, 905
 - device_startRKI, 905

- device_startTransaction, 906
- emv_activateTransaction, 906
- emv_allowFallback, 907
- emv_authenticateTransaction, 907
- emv_authenticateTransactionWithTimeout, 908
- emv_cancelTransaction, 908
- emv_completeTransaction, 908
- emv_getAutoAuthenticateTransaction, 909
- emv_getAutoCompleteTransaction, 909
- emv_registerCallBk, 910
- emv_removeAllApplicationData, 910
- emv_removeAllCAPK, 910
- emv_removeAllCRL, 910
- emv_removeApplicationData, 910
- emv_removeCAPK, 912
- emv_removeCRL, 912
- emv_retrieveAIDList, 912
- emv_retrieveApplicationData, 913
- emv_retrieveCAPKList, 914
- emv_retrieveCAPK, 913
- emv_retrieveCRL, 914
- emv_retrieveTerminalData, 915
- emv_setApplicationData, 915
- emv_setApplicationDataTLV, 916
- emv_setAutoAuthenticateTransaction, 916
- emv_setAutoCompleteTransaction, 916
- emv_setCAPK, 917
- emv_setCRL, 917
- emv_setTerminalData, 918
- emv_setTerminalMajorConfiguration, 918
- emv_setTransactionParameters, 918
- emv_startTransaction, 919
- executeTransaction, 920
- executeTransaction_WorldNet, 920
- forwardTransaction, 921
- forwardTransaction_WorldNet, 921
- ftpComm_callBack, 877
- httpComm_callBack, 878
- IN_OUT, 877
- icc_exchangeAPDU, 922
- icc_getICCRReaderStatus, 922
- icc_powerOffICC, 923
- icc_powerOnICC, 923
- IN, 877
- msr_cancelMSRSwipe, 923
- msr_registerCallBk, 923
- msr_registerCallBkp, 924
- msr_startMSRSwipe, 924
- OUT, 877
- pCMR_callBack, 878
- pCSFS_callBack, 878
- pEMV_callBack, 878
- pMSR_callBack, 878
- pMSR_callBkp, 878
- pMessageHotplug, 878
- pPIN_callBack, 879
- pRKI_callBack, 879
- pReadDataLog, 879
- pSendDataLog, 879
- pWN_callBack, 879
- pWP_callBack, 879
- parseMSRData, 924
- pin_registerCallBk, 924
- registerHotplugCallBk, 924
- registerLogCallBk, 925
- SDK_Version, 925
- setAbsoluteLibraryPath, 925
- v4Comm_callBack, 879
- libIDT_VP8800.h
 - comm_registerHTTPCallback, 932
 - comm_registerV4Callback, 932
 - config_getSerialNumber, 932
 - config_getSerialNumber_Len, 932
 - ctls_activateTransaction, 933
 - ctls_cancelTransaction, 934
 - ctls_displayOnlineAuthResult, 934
 - ctls_getAllConfigurationGroups, 935
 - ctls_getConfigurationGroup, 935
 - ctls_registerCallBk, 935
 - ctls_registerCallBkp, 935
 - ctls_removeAllApplicationData, 936
 - ctls_removeAllCAPK, 936
 - ctls_removeApplicationData, 936
 - ctls_removeCAPK, 936
 - ctls_removeConfigurationGroup, 937
 - ctls_retrieveAIDList, 937
 - ctls_retrieveApplicationData, 938
 - ctls_retrieveCAPKList, 939
 - ctls_retrieveCAPK, 938
 - ctls_retrieveTerminalData, 939
 - ctls_setApplicationData, 940
 - ctls_setCAPK, 940
 - ctls_setConfigurationGroup, 941
 - ctls_setTerminalData, 941
 - ctls_startTransaction, 942
 - device_SendDataCommandNEO, 956
 - device_activateTransaction, 943
 - device_calibrateParameters, 945
 - device_cancelTransaction, 945
 - device_close, 945
 - device_controlIndicator, 945
 - device_controlUserInterface, 946
 - device_createDirectory, 947
 - device_deleteDirectory, 948
 - device_deleteFile, 948
 - device_enablePassThrough, 949
 - device_enhancedPassthrough, 949
 - device_getCurrentDeviceType, 949
 - device_getDriveFreeSpace, 950
 - device_getFirmwareVersion, 950
 - device_getFirmwareVersion_Len, 950
 - device_getIDGStatusCodeString, 951
 - device_getMerchantRecord, 952
 - device_getMerchantRecord_Len, 953
 - device_getSDKWaitTime, 953
 - device_getThreadStackSize, 953

- device_getTransactionResults, 954
- device_init, 954
- device_isAttached, 954
- device_isConnected, 955
- device_listDirectory, 955
- device_pingDevice, 955
- device_registerCameraCallBk, 955
- device_registerCardStatusFrontSwitchCallBk, 956
- device_setCurrentDevice, 957
- device_setMerchantRecord, 957
- device_setSDKWaitTime, 957
- device_setThreadStackSize, 958
- device_setTransactionExponent, 958
- device_startTransaction, 958
- device_transferFile, 960
- emv_activateTransaction, 960
- emv_allowFallback, 961
- emv_authenticateTransaction, 961
- emv_authenticateTransactionWithTimeout, 961
- emv_cancelTransaction, 962
- emv_completeTransaction, 962
- emv_getAutoAuthenticateTransaction, 963
- emv_getAutoCompleteTransaction, 963
- emv_getEMVConfigurationCheckValue, 963
- emv_getEMVKernelCheckValue, 964
- emv_getEMVKernelVersion, 964
- emv_getEMVKernelVersion_Len, 964
- emv_registerCallBk, 966
- emv_removeAllApplicationData, 966
- emv_removeAllCAPK, 966
- emv_removeAllCRL, 966
- emv_removeAllExceptions, 966
- emv_removeApplicationData, 967
- emv_removeCAPK, 967
- emv_removeCRL, 967
- emv_removeException, 968
- emv_removeTransactionLog, 968
- emv_retrieveAIDList, 968
- emv_retrieveApplicationData, 969
- emv_retrieveCAPKList, 970
- emv_retrieveCAPK, 969
- emv_retrieveCRL, 970
- emv_retrieveExceptionList, 971
- emv_retrieveExceptionLogStatus, 971
- emv_retrieveTerminalData, 972
- emv_retrieveTransactionLog, 972
- emv_retrieveTransactionLogStatus, 973
- emv_setApplicationData, 974
- emv_setApplicationDataTLV, 974
- emv_setAutoAuthenticateTransaction, 975
- emv_setAutoCompleteTransaction, 975
- emv_setCAPK, 975
- emv_setCRL, 976
- emv_setException, 976
- emv_setTerminalData, 978
- emv_startTransaction, 978
- ftpComm_callBack, 930
- httpComm_callBack, 930
- IN_OUT, 929
- IN, 929
- lcd_addItemToList, 979
- lcd_cancelSlideShow, 979
- lcd_captureSignature, 980
- lcd_clearDisplay, 980
- lcd_clearEventQueue, 980
- lcd_createInputField, 981
- lcd_createInputField_Len, 982
- lcd_createList, 983
- lcd_createList_Len, 984
- lcd_customDisplayMode, 986
- lcd_displayButton, 986
- lcd_displayButton_Len, 988
- lcd_displayParagraph, 989
- lcd_displayText, 991
- lcd_displayText_Len, 992
- lcd_getInputEvent, 993
- lcd_getInputEvent_Len, 995
- lcd_getInputFieldValue, 997
- lcd_getSelectedListItem, 998
- lcd_getSelectedListItem_Len, 998
- lcd_resetInitialState, 998
- lcd_setBackgroundImage, 998
- lcd_setDisplayImage, 999
- lcd_setForeBackColor, 999
- lcd_startSlideShow, 1000
- msr_cancelMSRSwipe, 1001
- msr_flushTrackData, 1001
- msr_registerCallBk, 1001
- msr_registerCallBkp, 1001
- msr_startMSRSwipe, 1001
- OUT, 930
- pCMR_callBack, 930
- pCSFS_callBack, 930
- pEMV_callBack, 930
- pLog_callback, 930
- pMSR_callBack, 931
- pMSR_callBackp, 931
- pMessageHotplug, 931
- pPIN_callBack, 931
- pReadDataLog, 931
- pSendDataLog, 931
- parseMSRData, 1002
- pin_getEncryptedOnlinePIN, 1002
- pin_getPAN, 1003
- pin_promptCreditDebit, 1003
- pin_registerCallBk, 1003
- registerHotplugCallBk, 1004
- registerLogCallBk, 1004
- SDK_Version, 1004
- setAbsoluteLibraryPath, 1004
- v4Comm_callBack, 931
- ws_deleteSSLCert, 1004
- ws_getCertChainType, 1005
- ws_loadSSLCert, 1005
- ws_requestCSR, 1005
- ws_revokeSSLCert, 1006

- ws_updateRootCertificate, 1006
- libIDT_Vendi.h
 - comm_registerHTTPCallback, 695
 - comm_registerV4Callback, 695
 - config_getSerialNumber, 695
 - config_getSerialNumber_Len, 695
 - ctls_activateTransaction, 696
 - ctls_cancelTransaction, 697
 - ctls_getAllConfigurationGroups, 697
 - ctls_getConfigurationGroup, 698
 - ctls_registerCallBk, 698
 - ctls_registerCallBkp, 698
 - ctls_removeAllApplicationData, 698
 - ctls_removeAllCAPK, 698
 - ctls_removeApplicationData, 699
 - ctls_removeCAPK, 699
 - ctls_removeConfigurationGroup, 699
 - ctls_retrieveAIDList, 700
 - ctls_retrieveApplicationData, 700
 - ctls_retrieveCAPKList, 701
 - ctls_retrieveCAPK, 701
 - ctls_retrieveTerminalData, 702
 - ctls_setApplicationData, 702
 - ctls_setCAPK, 703
 - ctls_setConfigurationGroup, 703
 - ctls_setTerminalData, 704
 - ctls_startTransaction, 704
 - device_SendDataCommandNEO, 713
 - device_close, 706
 - device_controlUserInterface, 706
 - device_enablePassThrough, 707
 - device_getCurrentDeviceType, 708
 - device_getFirmwareVersion, 708
 - device_getFirmwareVersion_Len, 708
 - device_getIDGStatusCodeString, 709
 - device_getMerchantRecord, 710
 - device_getMerchantRecord_Len, 711
 - device_getSDKWaitTime, 711
 - device_getThreadStackSize, 711
 - device_getTransactionResults, 712
 - device_init, 712
 - device_isAttached, 712
 - device_isConnected, 713
 - device_pingDevice, 713
 - device_registerCameraCallBk, 713
 - device_registerCardStatusFrontSwitchCallBk, 713
 - device_setBurstMode, 714
 - device_setCurrentDevice, 715
 - device_setMerchantRecord, 715
 - device_setPollMode, 715
 - device_setSDKWaitTime, 716
 - device_setThreadStackSize, 716
 - emv_registerCallBk, 716
 - ftpComm_callBack, 693
 - httpComm_callBack, 693
 - IN_OUT, 693
 - IN, 692
 - msr_cancelMSRSwipe, 716
 - msr_registerCallBk, 717
 - msr_registerCallBkp, 717
 - msr_startMSRSwipe, 717
 - OUT, 693
 - pCMR_callBack, 693
 - pCSFS_callBack, 693
 - pEMV_callBack, 693
 - pMSR_callBack, 694
 - pMSR_callBackp, 694
 - pMessageHotplug, 694
 - pPIN_callBack, 694
 - pReadDataLog, 694
 - pSendDataLog, 694
 - parseMSRData, 717
 - pin_registerCallBk, 717
 - registerHotplugCallBk, 718
 - registerLogCallBk, 718
 - SDK_Version, 718
 - setAbsoluteLibraryPath, 718
 - v4Comm_callBack, 694
- loyalty_cancelTransaction
 - libIDT_Device.h, 265
 - libIDT_NEO2.h, 554
- loyalty_cancelTransactionSilent
 - libIDT_Device.h, 265
 - libIDT_NEO2.h, 554
- loyalty_registerCallBk
 - libIDT_Device.h, 266
 - libIDT_NEO2.h, 555
- loyalty_startTransaction
 - libIDT_Device.h, 266
 - libIDT_NEO2.h, 555
- msr_cancelMSRSwipe
 - libIDT_Augusta.h, 94
 - libIDT_Device.h, 268
 - libIDT_NEO2.h, 557
 - libIDT_SpectrumPro.h, 636
 - libIDT_UniPayI_V.h, 689
 - libIDT_VP3300_AJ.h, 768
 - libIDT_VP3300_BT.h, 820
 - libIDT_VP3300_COM.h, 871
 - libIDT_VP3300_USB.h, 923
 - libIDT_VP8800.h, 1001
 - libIDT_Vendi.h, 716
- msr_captureMode
 - libIDT_Augusta.h, 94
 - libIDT_Device.h, 268
- msr_clearMSRData
 - libIDT_Device.h, 268
 - libIDT_SpectrumPro.h, 636
- msr_disable
 - libIDT_Augusta.h, 94
 - libIDT_Device.h, 269
 - libIDT_SREDKey2.h, 656
- msr_flushTrackData
 - libIDT_Device.h, 269
 - libIDT_VP8800.h, 1001
- msr_getClearPANID

- libIDT_Augusta.h, 94
- libIDT_Device.h, 269
- libIDT_SREDKey2.h, 657
- msr_getExpirationMask
 - libIDT_Augusta.h, 95
 - libIDT_Device.h, 269
 - libIDT_SREDKey2.h, 657
- msr_getFunctionStatus
 - libIDT_Device.h, 270
 - libIDT_SREDKey2.h, 657
- msr_getKeyFormatForICCDUKPT
 - libIDT_Augusta.h, 95
 - libIDT_Device.h, 270
- msr_getKeyTypeForICCDUKPT
 - libIDT_Augusta.h, 96
 - libIDT_Device.h, 271
- msr_getMSRData
 - libIDT_Augusta.h, 96
 - libIDT_Device.h, 271
 - libIDT_SpectrumPro.h, 636
- msr_getSetting
 - libIDT_Augusta.h, 96
- msr_getSwipeForcedEncryptionOption
 - libIDT_Augusta.h, 98
 - libIDT_Device.h, 271
 - libIDT_SREDKey2.h, 658
- msr_getSwipeMaskOption
 - libIDT_Augusta.h, 98
 - libIDT_Device.h, 272
 - libIDT_SREDKey2.h, 658
- msr_registerCallBk
 - libIDT_Augusta.h, 98
 - libIDT_Device.h, 272
 - libIDT_L100.h, 349
 - libIDT_L80.h, 382
 - libIDT_MiniSmartII.h, 440
 - libIDT_NEO2.h, 557
 - libIDT_SREDKey2.h, 658
 - libIDT_SpectrumPro.h, 636
 - libIDT_UniPayl_V.h, 689
 - libIDT_VP3300_AJ.h, 768
 - libIDT_VP3300_BT.h, 820
 - libIDT_VP3300_COM.h, 871
 - libIDT_VP3300_USB.h, 923
 - libIDT_VP8800.h, 1001
 - libIDT_Vendi.h, 717
- msr_registerCallBkp
 - libIDT_Augusta.h, 99
 - libIDT_Device.h, 272
 - libIDT_L100.h, 349
 - libIDT_L80.h, 382
 - libIDT_MiniSmartII.h, 440
 - libIDT_NEO2.h, 557
 - libIDT_SREDKey2.h, 658
 - libIDT_SpectrumPro.h, 637
 - libIDT_UniPayl_V.h, 689
 - libIDT_VP3300_AJ.h, 769
 - libIDT_VP3300_BT.h, 821
- libIDT_VP3300_COM.h, 872
- libIDT_VP3300_USB.h, 924
- libIDT_VP8800.h, 1001
- libIDT_Vendi.h, 717
- msr_setClearPANID
 - libIDT_Augusta.h, 99
 - libIDT_Device.h, 272
 - libIDT_SREDKey2.h, 659
- msr_setExpirationMask
 - libIDT_Augusta.h, 99
 - libIDT_Device.h, 273
 - libIDT_SREDKey2.h, 659
- msr_setKeyFormatForICCDUKPT
 - libIDT_Augusta.h, 99
 - libIDT_Device.h, 273
- msr_setKeyTypeForICCDUKPT
 - libIDT_Augusta.h, 100
 - libIDT_Device.h, 273
- msr_setSetting
 - libIDT_Augusta.h, 100
- msr_setSwipeForcedEncryptionOption
 - libIDT_Augusta.h, 101
 - libIDT_Device.h, 274
 - libIDT_SREDKey2.h, 659
- msr_setSwipeMaskOption
 - libIDT_Augusta.h, 101
 - libIDT_Device.h, 274
 - libIDT_SREDKey2.h, 660
- msr_startMSRSwipe
 - libIDT_Augusta.h, 101
 - libIDT_Device.h, 275
 - libIDT_NEO2.h, 557
 - libIDT_SpectrumPro.h, 637
 - libIDT_UniPayl_V.h, 689
 - libIDT_VP3300_AJ.h, 769
 - libIDT_VP3300_BT.h, 821
 - libIDT_VP3300_COM.h, 872
 - libIDT_VP3300_USB.h, 924
 - libIDT_VP8800.h, 1001
 - libIDT_Vendi.h, 717
- OUT
 - libIDT_Augusta.h, 47
 - libIDT_Device.h, 112
 - libIDT_KioskIII.h, 299
 - libIDT_L100.h, 327
 - libIDT_L80.h, 360
 - libIDT_MiniSmartII.h, 394
 - libIDT_NEO2.h, 448
 - libIDT_PipReader.h, 568
 - libIDT_SREDKey2.h, 643
 - libIDT_SpectrumPro.h, 595
 - libIDT_UniPayl_V.h, 664
 - libIDT_VP3300_AJ.h, 722
 - libIDT_VP3300_BT.h, 773
 - libIDT_VP3300_COM.h, 825
 - libIDT_VP3300_USB.h, 877
 - libIDT_VP8800.h, 930
 - libIDT_Vendi.h, 693

- pCMR_callBack
 - libIDT_Augusta.h, [47](#)
 - libIDT_Device.h, [113](#)
 - libIDT_KioskIII.h, [299](#)
 - libIDT_L100.h, [327](#)
 - libIDT_L80.h, [360](#)
 - libIDT_MiniSmartII.h, [394](#)
 - libIDT_NEO2.h, [448](#)
 - libIDT_PipReader.h, [569](#)
 - libIDT_SREDKey2.h, [643](#)
 - libIDT_SpectrumPro.h, [596](#)
 - libIDT_UniPayI_V.h, [664](#)
 - libIDT_VP3300_AJ.h, [722](#)
 - libIDT_VP3300_BT.h, [774](#)
 - libIDT_VP3300_COM.h, [826](#)
 - libIDT_VP3300_USB.h, [878](#)
 - libIDT_VP8800.h, [930](#)
 - libIDT_Vendi.h, [693](#)
- pCSFS_callBack
 - libIDT_Augusta.h, [48](#)
 - libIDT_Device.h, [113](#)
 - libIDT_KioskIII.h, [299](#)
 - libIDT_L100.h, [327](#)
 - libIDT_L80.h, [360](#)
 - libIDT_MiniSmartII.h, [394](#)
 - libIDT_NEO2.h, [448](#)
 - libIDT_PipReader.h, [569](#)
 - libIDT_SREDKey2.h, [643](#)
 - libIDT_SpectrumPro.h, [596](#)
 - libIDT_UniPayI_V.h, [664](#)
 - libIDT_VP3300_AJ.h, [722](#)
 - libIDT_VP3300_BT.h, [774](#)
 - libIDT_VP3300_COM.h, [826](#)
 - libIDT_VP3300_USB.h, [878](#)
 - libIDT_VP8800.h, [930](#)
 - libIDT_Vendi.h, [693](#)
- pEMV_callBack
 - libIDT_Augusta.h, [48](#)
 - libIDT_Device.h, [113](#)
 - libIDT_KioskIII.h, [300](#)
 - libIDT_L100.h, [327](#)
 - libIDT_L80.h, [360](#)
 - libIDT_MiniSmartII.h, [394](#)
 - libIDT_NEO2.h, [448](#)
 - libIDT_PipReader.h, [569](#)
 - libIDT_SREDKey2.h, [643](#)
 - libIDT_SpectrumPro.h, [596](#)
 - libIDT_UniPayI_V.h, [665](#)
 - libIDT_VP3300_AJ.h, [722](#)
 - libIDT_VP3300_BT.h, [774](#)
 - libIDT_VP3300_COM.h, [826](#)
 - libIDT_VP3300_USB.h, [878](#)
 - libIDT_VP8800.h, [930](#)
 - libIDT_Vendi.h, [693](#)
- pFW_callBack
 - libIDT_Augusta.h, [48](#)
 - libIDT_Device.h, [113](#)
 - libIDT_L100.h, [327](#)
- libIDT_L80.h, [361](#)
- libIDT_NEO2.h, [449](#)
- libIDT_SREDKey2.h, [644](#)
- pLCD_callBack
 - libIDT_Device.h, [113](#)
 - libIDT_NEO2.h, [449](#)
 - libIDT_SREDKey2.h, [644](#)
- pLog_callback
 - libIDT_Device.h, [113](#)
 - libIDT_VP8800.h, [930](#)
- pMSR_callBack
 - libIDT_Augusta.h, [48](#)
 - libIDT_Device.h, [114](#)
 - libIDT_KioskIII.h, [300](#)
 - libIDT_L100.h, [328](#)
 - libIDT_L80.h, [361](#)
 - libIDT_MiniSmartII.h, [395](#)
 - libIDT_NEO2.h, [449](#)
 - libIDT_PipReader.h, [569](#)
 - libIDT_SREDKey2.h, [644](#)
 - libIDT_SpectrumPro.h, [596](#)
 - libIDT_UniPayI_V.h, [665](#)
 - libIDT_VP3300_AJ.h, [723](#)
 - libIDT_VP3300_BT.h, [774](#)
 - libIDT_VP3300_COM.h, [826](#)
 - libIDT_VP3300_USB.h, [878](#)
 - libIDT_VP8800.h, [931](#)
 - libIDT_Vendi.h, [694](#)
- pMSR_callBackp
 - libIDT_Augusta.h, [48](#)
 - libIDT_Device.h, [114](#)
 - libIDT_KioskIII.h, [300](#)
 - libIDT_L100.h, [328](#)
 - libIDT_L80.h, [361](#)
 - libIDT_MiniSmartII.h, [395](#)
 - libIDT_NEO2.h, [449](#)
 - libIDT_PipReader.h, [569](#)
 - libIDT_SREDKey2.h, [644](#)
 - libIDT_SpectrumPro.h, [596](#)
 - libIDT_UniPayI_V.h, [665](#)
 - libIDT_VP3300_AJ.h, [723](#)
 - libIDT_VP3300_BT.h, [775](#)
 - libIDT_VP3300_COM.h, [827](#)
 - libIDT_VP3300_USB.h, [878](#)
 - libIDT_VP8800.h, [931](#)
 - libIDT_Vendi.h, [694](#)
- pMessageHotplug
 - libIDT_Augusta.h, [48](#)
 - libIDT_Device.h, [113](#)
 - libIDT_KioskIII.h, [300](#)
 - libIDT_L100.h, [328](#)
 - libIDT_L80.h, [361](#)
 - libIDT_MiniSmartII.h, [395](#)
 - libIDT_NEO2.h, [449](#)
 - libIDT_PipReader.h, [569](#)
 - libIDT_SREDKey2.h, [644](#)
 - libIDT_SpectrumPro.h, [596](#)
 - libIDT_UniPayI_V.h, [665](#)

- libIDT_VP3300_AJ.h, [723](#)
- libIDT_VP3300_BT.h, [774](#)
- libIDT_VP3300_COM.h, [826](#)
- libIDT_VP3300_USB.h, [878](#)
- libIDT_VP8800.h, [931](#)
- libIDT_Vendi.h, [694](#)
- pPIN_callBack
 - libIDT_Augusta.h, [48](#)
 - libIDT_Device.h, [114](#)
 - libIDT_KioskIII.h, [300](#)
 - libIDT_L100.h, [328](#)
 - libIDT_L80.h, [361](#)
 - libIDT_MiniSmartII.h, [395](#)
 - libIDT_NEO2.h, [449](#)
 - libIDT_PipReader.h, [570](#)
 - libIDT_SREDKey2.h, [644](#)
 - libIDT_SpectrumPro.h, [597](#)
 - libIDT_UniPayI_V.h, [665](#)
 - libIDT_VP3300_AJ.h, [723](#)
 - libIDT_VP3300_BT.h, [775](#)
 - libIDT_VP3300_COM.h, [827](#)
 - libIDT_VP3300_USB.h, [879](#)
 - libIDT_VP8800.h, [931](#)
 - libIDT_Vendi.h, [694](#)
- pRKI_callBack
 - libIDT_Device.h, [114](#)
 - libIDT_NEO2.h, [450](#)
 - libIDT_VP3300_AJ.h, [723](#)
 - libIDT_VP3300_BT.h, [775](#)
 - libIDT_VP3300_COM.h, [827](#)
 - libIDT_VP3300_USB.h, [879](#)
- pReadDataLog
 - libIDT_Augusta.h, [49](#)
 - libIDT_Device.h, [114](#)
 - libIDT_KioskIII.h, [300](#)
 - libIDT_L100.h, [328](#)
 - libIDT_L80.h, [361](#)
 - libIDT_MiniSmartII.h, [395](#)
 - libIDT_NEO2.h, [449](#)
 - libIDT_PipReader.h, [570](#)
 - libIDT_SREDKey2.h, [644](#)
 - libIDT_SpectrumPro.h, [597](#)
 - libIDT_UniPayI_V.h, [665](#)
 - libIDT_VP3300_AJ.h, [723](#)
 - libIDT_VP3300_BT.h, [775](#)
 - libIDT_VP3300_COM.h, [827](#)
 - libIDT_VP3300_USB.h, [879](#)
 - libIDT_VP8800.h, [931](#)
 - libIDT_Vendi.h, [694](#)
- pSendDataLog
 - libIDT_Augusta.h, [49](#)
 - libIDT_Device.h, [114](#)
 - libIDT_KioskIII.h, [300](#)
 - libIDT_L100.h, [328](#)
 - libIDT_L80.h, [361](#)
 - libIDT_MiniSmartII.h, [395](#)
 - libIDT_NEO2.h, [450](#)
 - libIDT_PipReader.h, [570](#)
- libIDT_SREDKey2.h, [644](#)
- libIDT_SpectrumPro.h, [597](#)
- libIDT_UniPayI_V.h, [665](#)
- libIDT_VP3300_AJ.h, [723](#)
- libIDT_VP3300_BT.h, [775](#)
- libIDT_VP3300_COM.h, [827](#)
- libIDT_VP3300_USB.h, [879](#)
- libIDT_VP8800.h, [931](#)
- libIDT_Vendi.h, [694](#)
- pWN_callBack
 - libIDT_Device.h, [114](#)
 - libIDT_NEO2.h, [450](#)
 - libIDT_VP3300_AJ.h, [723](#)
 - libIDT_VP3300_BT.h, [775](#)
 - libIDT_VP3300_COM.h, [827](#)
 - libIDT_VP3300_USB.h, [879](#)
- pWP_callBack
 - libIDT_Device.h, [114](#)
 - libIDT_NEO2.h, [450](#)
 - libIDT_VP3300_AJ.h, [724](#)
 - libIDT_VP3300_BT.h, [775](#)
 - libIDT_VP3300_COM.h, [827](#)
 - libIDT_VP3300_USB.h, [879](#)
- parseMSRData
 - libIDT_Augusta.h, [102](#)
 - libIDT_Device.h, [275](#)
 - libIDT_KioskIII.h, [323](#)
 - libIDT_NEO2.h, [558](#)
 - libIDT_PipReader.h, [591](#)
 - libIDT_SpectrumPro.h, [637](#)
 - libIDT_UniPayI_V.h, [689](#)
 - libIDT_VP3300_AJ.h, [769](#)
 - libIDT_VP3300_BT.h, [821](#)
 - libIDT_VP3300_COM.h, [872](#)
 - libIDT_VP3300_USB.h, [924](#)
 - libIDT_VP8800.h, [1002](#)
 - libIDT_Vendi.h, [717](#)
- parsePINBlockData
 - libIDT_Device.h, [275](#)
 - libIDT_SpectrumPro.h, [637](#)
- parsePINData
 - libIDT_Device.h, [276](#)
 - libIDT_SpectrumPro.h, [638](#)
- pin_cancelPINEntry
 - libIDT_Augusta.h, [102](#)
 - libIDT_Device.h, [276](#)
 - libIDT_NEO2.h, [558](#)
 - libIDT_SpectrumPro.h, [638](#)
- pin_capturePin
 - libIDT_Device.h, [276](#)
 - libIDT_NEO2.h, [558](#)
- pin_capturePinExt
 - libIDT_Device.h, [277](#)
 - libIDT_NEO2.h, [559](#)
- pin_getEncryptedOnlinePIN
 - libIDT_Device.h, [278](#)
 - libIDT_VP8800.h, [1002](#)
- pin_getEncryptedPIN

- libIDT_Device.h, 279
- libIDT_L100.h, 349
- libIDT_L80.h, 382
- pin_getFunctionKey
 - libIDT_Device.h, 279
 - libIDT_L100.h, 349
 - libIDT_L80.h, 383
- pin_getPAN
 - libIDT_Device.h, 280
 - libIDT_VP8800.h, 1003
- pin_getPIN
 - libIDT_Device.h, 281
 - libIDT_SpectrumPro.h, 638
- pin_getPanEntry
 - libIDT_Device.h, 280
 - libIDT_NEO2.h, 560
- pin_inputFromPrompt
 - libIDT_Device.h, 281
 - libIDT_NEO2.h, 561
- pin_promptCreditDebit
 - libIDT_Device.h, 282
 - libIDT_VP8800.h, 1003
- pin_promptForAmount
 - libIDT_Device.h, 283
- pin_promptForAmountInput
 - libIDT_Device.h, 283
 - libIDT_L100.h, 350
 - libIDT_L80.h, 383
- pin_promptForKeyInput
 - libIDT_Device.h, 286
 - libIDT_L100.h, 353
 - libIDT_L80.h, 386
- pin_promptForNumericKey
 - libIDT_Device.h, 289
 - libIDT_NEO2.h, 562
- pin_promptForNumericKeyWithSwipe
 - libIDT_Device.h, 290
 - libIDT_NEO2.h, 562
- pin_registerCallBk
 - libIDT_Augusta.h, 102
 - libIDT_Device.h, 291
 - libIDT_KioskIII.h, 323
 - libIDT_L100.h, 356
 - libIDT_L80.h, 389
 - libIDT_MiniSmartII.h, 440
 - libIDT_NEO2.h, 563
 - libIDT_PipReader.h, 591
 - libIDT_SREDKey2.h, 660
 - libIDT_SpectrumPro.h, 639
 - libIDT_UniPayI_V.h, 690
 - libIDT_VP3300_AJ.h, 769
 - libIDT_VP3300_BT.h, 821
 - libIDT_VP3300_COM.h, 872
 - libIDT_VP3300_USB.h, 924
 - libIDT_VP8800.h, 1003
 - libIDT_Vendi.h, 717
- pin_sendBeep
 - libIDT_Device.h, 291
- libIDT_L100.h, 356
- libIDT_L80.h, 389
- pin_setKeyValues
 - libIDT_Device.h, 292
 - libIDT_L100.h, 357
 - libIDT_L80.h, 390
 - libIDT_NEO2.h, 563
- registerHotplugCallBk
 - libIDT_Augusta.h, 102
 - libIDT_Device.h, 292
 - libIDT_KioskIII.h, 323
 - libIDT_L100.h, 357
 - libIDT_L80.h, 390
 - libIDT_MiniSmartII.h, 440
 - libIDT_NEO2.h, 564
 - libIDT_PipReader.h, 591
 - libIDT_SREDKey2.h, 660
 - libIDT_SpectrumPro.h, 639
 - libIDT_UniPayI_V.h, 690
 - libIDT_VP3300_AJ.h, 769
 - libIDT_VP3300_BT.h, 821
 - libIDT_VP3300_COM.h, 872
 - libIDT_VP3300_USB.h, 924
 - libIDT_VP8800.h, 1004
 - libIDT_Vendi.h, 718
- registerLogCallBk
 - libIDT_Augusta.h, 103
 - libIDT_Device.h, 292
 - libIDT_KioskIII.h, 323
 - libIDT_L100.h, 357
 - libIDT_L80.h, 390
 - libIDT_MiniSmartII.h, 440
 - libIDT_NEO2.h, 564
 - libIDT_PipReader.h, 591
 - libIDT_SREDKey2.h, 660
 - libIDT_SpectrumPro.h, 639
 - libIDT_UniPayI_V.h, 690
 - libIDT_VP3300_AJ.h, 770
 - libIDT_VP3300_BT.h, 822
 - libIDT_VP3300_COM.h, 873
 - libIDT_VP3300_USB.h, 925
 - libIDT_VP8800.h, 1004
 - libIDT_Vendi.h, 718
- rs232_device_init
 - libIDT_Device.h, 292
 - libIDT_KioskIII.h, 324
 - libIDT_MiniSmartII.h, 440
 - libIDT_NEO2.h, 564
 - libIDT_PipReader.h, 591
 - libIDT_SREDKey2.h, 660
 - libIDT_SpectrumPro.h, 639
 - libIDT_VP3300_COM.h, 873
- SDK_Version
 - libIDT_Augusta.h, 103
 - libIDT_Device.h, 294
 - libIDT_KioskIII.h, 324
 - libIDT_L100.h, 357

- libIDT_L80.h, [390](#)
- libIDT_MiniSmartII.h, [441](#)
- libIDT_NEO2.h, [566](#)
- libIDT_PipReader.h, [592](#)
- libIDT_SREDKey2.h, [661](#)
- libIDT_SpectrumPro.h, [640](#)
- libIDT_UniPayI_V.h, [690](#)
- libIDT_VP3300_AJ.h, [770](#)
- libIDT_VP3300_BT.h, [822](#)
- libIDT_VP3300_COM.h, [873](#)
- libIDT_VP3300_USB.h, [925](#)
- libIDT_VP8800.h, [1004](#)
- libIDT_Vendi.h, [718](#)
- set_open_com_port_timeout
 - libIDT_Device.h, [294](#)
 - libIDT_NEO2.h, [566](#)
- setAbsoluteLibraryPath
 - libIDT_Augusta.h, [103](#)
 - libIDT_Device.h, [294](#)
 - libIDT_KioskIII.h, [324](#)
 - libIDT_L100.h, [358](#)
 - libIDT_L80.h, [391](#)
 - libIDT_MiniSmartII.h, [441](#)
 - libIDT_NEO2.h, [566](#)
 - libIDT_PipReader.h, [592](#)
 - libIDT_SREDKey2.h, [661](#)
 - libIDT_SpectrumPro.h, [640](#)
 - libIDT_UniPayI_V.h, [690](#)
 - libIDT_VP3300_AJ.h, [770](#)
 - libIDT_VP3300_BT.h, [822](#)
 - libIDT_VP3300_COM.h, [874](#)
 - libIDT_VP3300_USB.h, [925](#)
 - libIDT_VP8800.h, [1004](#)
 - libIDT_Vendi.h, [718](#)
- Source_C/libIDT_Augusta.h, [44](#)
- Source_C/libIDT_Device.h, [103](#)
- Source_C/libIDT_KioskIII.h, [297](#)
- Source_C/libIDT_L100.h, [325](#)
- Source_C/libIDT_L80.h, [358](#)
- Source_C/libIDT_MiniSmartII.h, [391](#)
- Source_C/libIDT_NEO2.h, [442](#)
- Source_C/libIDT_PipReader.h, [566](#)
- Source_C/libIDT_SREDKey2.h, [641](#)
- Source_C/libIDT_SpectrumPro.h, [593](#)
- Source_C/libIDT_UniPayI_V.h, [662](#)
- Source_C/libIDT_VP3300_AJ.h, [718](#)
- Source_C/libIDT_VP3300_BT.h, [770](#)
- Source_C/libIDT_VP3300_COM.h, [822](#)
- Source_C/libIDT_VP3300_USB.h, [874](#)
- Source_C/libIDT_VP8800.h, [925](#)
- Source_C/libIDT_Vendi.h, [691](#)
- v4Comm_callBack
 - libIDT_Augusta.h, [49](#)
 - libIDT_Device.h, [115](#)
 - libIDT_KioskIII.h, [301](#)
 - libIDT_L100.h, [328](#)
 - libIDT_L80.h, [362](#)
 - libIDT_MiniSmartII.h, [395](#)
 - libIDT_NEO2.h, [450](#)
 - libIDT_PipReader.h, [570](#)
 - libIDT_SREDKey2.h, [645](#)
 - libIDT_SpectrumPro.h, [597](#)
 - libIDT_UniPayI_V.h, [666](#)
 - libIDT_VP3300_AJ.h, [724](#)
 - libIDT_VP3300_BT.h, [775](#)
 - libIDT_VP3300_COM.h, [827](#)
 - libIDT_VP3300_USB.h, [879](#)
 - libIDT_VP8800.h, [931](#)
 - libIDT_Vendi.h, [694](#)
- ws_deleteSSLCert
 - libIDT_Device.h, [295](#)
 - libIDT_VP8800.h, [1004](#)
- ws_getCertChainType
 - libIDT_Device.h, [295](#)
 - libIDT_VP8800.h, [1005](#)
- ws_loadSSLCert
 - libIDT_Device.h, [295](#)
 - libIDT_VP8800.h, [1005](#)
- ws_requestCSR
 - libIDT_Device.h, [296](#)
 - libIDT_VP8800.h, [1005](#)
- ws_revokeSSLCert
 - libIDT_Device.h, [296](#)
 - libIDT_VP8800.h, [1006](#)
- ws_updateRootCertificate
 - libIDT_Device.h, [296](#)
 - libIDT_VP8800.h, [1006](#)