Android Client ZSDK API Reference

Rev. 1.2
May 21, 2024

ID TECH
10721 Walker Street, Cypress, CA 90630-4720
Tel: (714) 761-6368 Fax (714) 761-8880
www.idtechproducts.com

ID TECH
10721 Walker St.
Cypress, CA 90630

Modification History

Rev	Date	Changes	By
1	05/24/2023	Initial Draft	IW
1.1	05/14/2024	Update APIs based on zsdk-client 1.84	VW
1.2	05/21/2024	Major style/organization pass.	CB

Table of Contents

1. Overview

This document provides an API reference for the ID TECH Android Client ZSDK.

2. Transaction Commands

The section below describes API methods for performing transactions.

2.1. ActivateTransactionAsync

```
fun Client.ActivateTransactionAsync(deviceId: String, timeout: UByte,
tlvData: List<UByte>, timeoutMilli: Long = 3000):
ActivateTransactionCommand
```

Asynchronously activate a transaction. This method responds with a result of **Activate Transaction (02-40)**. If auto-authentication is enabled, it responds with a result of **Contact Authenticate Transaction (60-11)**; if auto-completion is enabled, it responds with a result of **Complete Transaction (60-12)**.

Parameters

Name	Type	Description
deviceId	String	Device ID.
timeout	UByte	Timeout value in seconds.
tlvData	List<UByte>	Activate Command TLVs.
timeoutMilli	Long	The timeout time in milliseconds; default is 3s.

Returns

```
StartTransactionResponseData
```

Example

```
GlobalScope.launch(Dispatchers.IO) {
   Client.waitUntilConnected()
   val tlvData = listOf(0x9c, 0x01, 0x00, 0x9f, 0x02, 0x06, 0x00, 0x00
, 0x00, 0x00, 0x01, 0x00, 0xdf, 0xef, 0x37, 0x01, 0x07, 0xdf, 0xef, 0x
3c, 0x03, 0x01, 0x00, 0x60)
   val data = ArrayList<UByte>().also{
        for(tlv:Int in tlvData){
        it.add(tlv.toByte().toUByte())
       }
   }
   val request = ActivateTransactionRequestData(timeout = 30.toByte().
toUByte(),tlvData = data)
   val cmd = Client.ActivateTransactionAsync(connectedDeviceId!!, requ
est)
   val status = cmd.waitForCompletionWithTimeout(20000)
   outputInCoroutine("ActivateTransaction: ${status.name}")
}
```

2.2. ActivateCtlsTransactionAsync

```
fun Client.ActivateCtlsTransactionAsync(deviceId: String, timeout:
UByte, tlvData: Array<UByte>, timeoutMilli: Long = 3000):
ContactlessActivateTransactionCommand
```

Description

Start a CTLS transaction request with TLV parameters.

Returns

```
ContactlessActivateTransactionResult
```

Parameters

Name	Type	Description
deviceId	string	Device ID.
timeout	UByte	Timeout time.
tlvData	Array<UByte>	TLV Data.
timeoutMilli	Long	The timeout time in milliseconds; default is 3s.

Example

```
val timeout = 30.toUByte()
val tlv = arrayOf<UByte>(0xdfu, 0xefu, 0x3cu, 0x03u, 0x01u, 0x00u, 0x6
0u)
val cmd = Client.ActivateCtlsTransactionAsync(connectedDeviceId!!,time
out,tlv)
cmd.waitForCompletionWithTimeout(10000)
val status = cmd.getCommandStatus()
cmd.getResultData()?.apply {
    outputInCoroutine("ActivateCtlsTransaction: ${status.name}, " +
          "status : ${statusCode.toInt()}"+
          "errorCode : ${errorCode.toInt()}"+
          "capturedDataType : ${CapturedDataType.values()[capturedData
Type.toInt()]}"+
          "encryptionMode : ${EncryptionMode.values()[encryptionMode.t
oInt()]}"+
          "msrMsdEncryptionEnabled : $msrMsdEncryptionEnabled"+
          "emvEncryptionEnabled : $emvEncryptionEnabled"+
          "ksn : ${ksn?.toHexString()}"+
          "track1Data : ${track1Data?.toHexString()}"+
          "track2Data : ${track2Data?.toHexString()}"+
          "clearingRecord : ${clearingRecord?.toHexString()}"+
          "fullTlvData : ${fullTlvData?.toHexString()}")
}
```

2.3. StartEMVTransactionAsync

```
fun Client.StartEMVTransactionAsync(deviceId: String, amount: Double,
amountOther: Double, transType: UByte, transTimeout: Int, forceOnline:
Boolean, timeoutMilli: Long = 3000): StartEMVTransactionCommand
```

Start an MSR Transaction.

Parameters

Name	Type	Description
deviceId	String	Device ID.
amount	Long	Transaction amount.
amountOther	Long	Other transaction amount.
transType	UByte	Transaction type (Tag value 9C). Indicates the type of financial transaction. 0x00 = Purchase, 0x20 = Refund.
transTimeout	Int	The transTimeout of transaction timeout time.
forceOnline	Boolean	Allow normal processing to occur via Terminal action analysis
timeoutMilli	Long	The timeout time in milliseconds; default is 3s.

Returns

```
StartTransactionResponseData(var respType: TransactionResponseType,var
attribute: UShort,var cardData: List)
StartTransactionResponseData(var respType: TransactionResponseType,var
attribute: UShort,var cardData: List)
```

Example

```
GlobalScope.launch(Dispatchers.IO) {
    Client.waitUntilConnected()
    if (connectedDeviceId == null) Returns@launch
    val setaacmd = Client.SetAutoAuthenticateAsync(connectedDeviceId!!
, true)
    val setaa_rv: ClientCommandStatus = setaacmd.waitForCompletionWith
Timeout(1000)
    Log.d(TAG, "Enable Auto Authenticate: ${setaa_rv.name}")
    val setaccmd = Client.SetAutoCompleteAsync(connectedDeviceId!!, tr
ue)
    val setac_rv = setaccmd.waitForCompletionWithTimeout(1000)
    Log.d(TAG, "Enable Auto Complete: ${setac_rv.name}")
    val amount = 0.1
    val amountOther = 0.0
    val transType: UByte = 0u
    val transTimeout = 100
    val startTransCmd = Client.StartEMVTransactionAsync(connectedDevic
eId!!,
                amount, amountOther, transType, transTimeout, false,10
0000)
    startTransCmd.waitForCompletion()
```

```
    var startTransStatus = startTransCmd.getCommandStatus()
    Log.d(TAG, "Transaction Status: ${startTransStatus.name}")
    val resultData: StartTransactionResponseData? = startTransCmd.getR
esultData()
}
```

2.4. StartEMVTransactionExAsync

```
fun Client.StartEMVTransactionExAsync(deviceId: String, amount: Long,
amountOther: Long, transType: UByte, transTimeout: UByte, forceOnline:
Boolean, timeoutMilli: Long = 3000): StartEMVTransactionExCommand
```

Starts a new contact EMV L2 transaction (ICC + MSR) or an MSR-only transaction. This function goes through the processes below when a user inserts an EMV card into the reader:

1. Card power on
2. Card activation
3. Application selection
4. Initiate application processing
5. Get process options
6. Read records

Parameters

Name	Type	Description
deviceId	String	Device ID.
amount	Long	Transaction amount.
amountOther	Long	Other transaction amount.
transType	UByte	Transaction type (Tag value 9C). Indicates the type of financial transaction. 0x00 = Purchase, 0x20 = Refund.
transTimeout	UByte	The transTimeout of transaction timeout time.
forceOnline	Boolean	Allow normal processing to occur via Terminal action analysis
timeoutMilli	Long	The timeout time in milliseconds; default is 3s.

Returns

```
StartTransactionResponseData(var respType: TransactionResponseType,var
attribute: UShort,var cardData: List)
```

Example

```
GlobalScope.launch(Dispatchers.IO) {
    Client.waitUntilConnected()
    if (connectedDeviceId == null) Returns@launch
    val setaacmd = Client.SetAutoAuthenticateAsync(connectedDeviceId!!
, true)
    val setaa_rv: ClientCommandStatus = setaacmd.waitForCompletionWith
Timeout(1000)
    Log.d(TAG, "Enable Auto Authenticate: ${setaa_rv.name}")
    val setaccmd = Client.SetAutoCompleteAsync(connectedDeviceId!!, tr
```

```
ue)
    val setac_rv = setaccmd.waitForCompletionWithTimeout(1000)
    Log.d(TAG, "Enable Auto Complete: ${setac_rv.name}")
    val startTransCmd = Client.StartEMVTransactionExAsync(connectedDev
iceId!!,
                        amount, amountOther, transType, transTimeout,
                        forceOnline,100000)
    startTransCmd.waitForCompletion()
    var startTransStatus = startTransCmd.getCommandStatus()
    Log.d(TAG, "Transaction Status: ${startTransStatus.name}")
    val resultData: StartTransactionResponseData? = startTransCmd.getR
esultData()
}
```

2.5. StartTransactionAsync

```
fun Client.StartTransactionAsync(deviceId: String, amount: Double,
amountOther: Double, transType: UByte, transTimeout: UByte,
transInterfaces: UByte, timeoutMilli: Long = 3000):
StartTransactionCommand
```

Start a transaction.

Parameters

Name	Type	Description
deviceId	String	Device ID.
amount	Double	Transaction amount.
amountOther	Double	Other transaction amount.
transType	UByte	Transaction type (Tag value 9C). Indicates the type of financial transaction. 0x00 = Purchase, 0x20 = Refund.
transTimeout	UByte	The transTimeout of transaction timeout time.
transInterfaces	UByte	Transaction interface type. 1=MSR, 2=CTLS, 4=EMV.
timeoutMilli	Long	The timeout time in milliseconds; default is 3s.

Returns

```
StartTransactionResponseData(var respType: TransactionResponseType,var
attribute: UShort,var cardData: List)
```

Example

```
GlobalScope.launch(Dispatchers.IO) {
    Client.waitUntilConnected()
    if (connectedDeviceId == null) Returns@launch
    val setaacmd = Client.SetAutoAuthenticateAsync(connectedDeviceId!!
, true)
    val setaa_rv: ClientCommandStatus = setaacmd.waitForCompletionWith
Timeout(1000)
    Log.d(TAG, "Enable Auto Authenticate: ${setaa_rv.name}")
```

```
    val setaccmd = Client.SetAutoCompleteAsync(connectedDeviceId!!, tr
ue)
    val setac_rv = setaccmd.waitForCompletionWithTimeout(1000)
    Log.d(TAG, "Enable Auto Complete: ${setac_rv.name}")
    val startTransCmd = Client.StartTransactionAsync(connectedDeviceId
!!,
                                                     amount, amountOth
er, transType, transTimeout,
                                                     transInterfaceTyp
e.toUByte(), 100000)
    startTransCmd.waitForCompletion()
    var startTransStatus = startTransCmd.getCommandStatus()
    Log.d(TAG, "Transaction Status: ${startTransStatus.name}")
    val resultData: StartTransactionResponseData? = startTransCmd.getR
esultData()
}
```

2.6. StartTransactionExAsync

```
fun Client.StartTransactionExAsync(deviceId: String, amount: Long,
amountOther: Long, transType: UByte, transTimeout: UByte,
transInterfaces: UByte, timeoutMilli: Long = 3000):
StartTransactionExCommand
```

Asynchronously starts a transaction. This method responds with a result of Activate Transaction (02-40). If auto-authentication is enabled, it responds with a result of Contact Authenticate Transaction (60-11); if auto-completion is enabled, it responds with a result of Complete Transaction (60-12).

Note: The difference between **StartTransactionExAsync** and **StartTransactionAsync** is in their use of different types for the **amount** and **amountOther** parameters.

Parameters

Name	Type	Description
deviceId	String	Device ID.
amount	Long	Transaction amount.
amountOther	Long	Other transaction amount.
transType	UByte	Transaction type (Tag value 9C). Indicates the type of financial transaction. 0x00 = Purchase, 0x20 = Refund.
transTimeout	UByte	Transaction timeout time.
transInterfaces	UByte	Transaction interface type. 1=MSR, 2=CTLS, 4=EMV.
timeoutMilli	Long	The timeout time in milliseconds; default is 3s.

Returns

```
StartTransactionResponseData(var respType: TransactionResponseType,var
attribute: UShort,var cardData: List)
```

Example

```
GlobalScope.launch(Dispatchers.IO) {
    Client.waitUntilConnected()
    if (connectedDeviceId == null) Returns@launch
    val setaacmd = Client.SetAutoAuthenticateAsync(connectedDeviceId!!
, true)
    val setaa_rv: ClientCommandStatus = setaacmd.waitForCompletionWith
Timeout(1000)
    Log.d(TAG, "Enable Auto Authenticate: ${setaa_rv.name}")
    val setaccmd = Client.SetAutoCompleteAsync(connectedDeviceId!!, tr
ue)
    val setac_rv = setaccmd.waitForCompletionWithTimeout(1000)
    Log.d(TAG, "Enable Auto Complete: ${setac_rv.name}")
    val startTransCmd = Client.StartTransactionExAsync(connectedDevice
Id!!,
                                                       amount, amountOth
er, transType, transTimeout,
                                                       transInterfaceTyp
e.toUByte(), 100000)
    startTransCmd.waitForCompletion()
    var startTransStatus = startTransCmd.getCommandStatus()
    Log.d(TAG, "Transaction Status: ${startTransStatus.name}")
    val resultData: StartTransactionResponseData? = startTransCmd.getR
esultData()
}
```

2.7. StartTransactionDetailedExAsync

```
fun Client.StartTransactionDetailedExAsync(deviceId: String, amount:
Long, amountOther: Long, transType: UByte, transTimeout: UByte,
transInterfaces: UByte, enableEMVFallback: Boolean,
secondCommandTimeout: UByte, timeoutMilli: Long = 3000):
StartTransactionDetailedExCommand
```

Detailed version of **StartTransactionExAsync**. Note: This function reproduces **StartTransactionExAsync** and supports additional parameters: **enableFallbackSupport** and **secondCommandTimeout**.

Parameters

Name	Type	Description
deviceId	String	Device ID.
amount	Double	Transaction amount.
amountOther	Double	Other transaction amount.
transType	UByte	Transaction type (Tag value 9C). Indicates the type of financial transaction. 0x00 = Purchase, 0x20 = Refund.
transTimeout	UByte	The transTimeout of transaction timeout time.
transInterfaces	UByte	Transaction interface type. 1=MSR, 2=CTLS, 4=EMV.
timeoutMilli	Long	The timeout time in milliseconds; default is 3s.

enableFallbackSupport	Boolean	If true, fallback support is enabled.
secondCommandTimeout	UByte	A second timeout time.

Returns

```
StartTransactionResponseData(var respType: TransactionResponseType,var
attribute: UShort,var cardData: List)
```

Example

```
GlobalScope.launch(Dispatchers.IO) {
    Client.waitUntilConnected()
    if (connectedDeviceId == null) Returns@launch
    val setaacmd = Client.SetAutoAuthenticateAsync(connectedDeviceId!!
, true)
    val setaa_rv: ClientCommandStatus = setaacmd.waitForCompletionWith
Timeout(1000)
    Log.d(TAG, "Enable Auto Authenticate: ${setaa_rv.name}")
    val setaccmd = Client.SetAutoCompleteAsync(connectedDeviceId!!, tr
ue)
    val setac_rv = setaccmd.waitForCompletionWithTimeout(1000)
    Log.d(TAG, "Enable Auto Complete: ${setac_rv.name}")
    val startTransCmd = Client.StartTransactionDetailedExAsync(connect
edDeviceId!!,
                            amount, amountOther, transType, transTimeo
ut,
                            transInterfaceType.toUByte(),enableEMVFall
back,secondCommandTimeout,100000)
    startTransCmd.waitForCompletion()
    var startTransStatus = startTransCmd.getCommandStatus()
    Log.d(TAG, "Transaction Status: ${startTransStatus.name}")
    val resultData: StartTransactionResponseData? = startTransCmd.getR
esultData()
}
```

2.8. StartMSRTransAsync

```
fun Client.StartMSRTransAsync(deviceId: String, transTimeout: UShort,
timeoutMilli: Long = 3000): StartMSRTransCommand
```

Start an MSR transaction.

Parameters

Name	Type	Description
deviceId	String	Device ID.
transTimeout	UShort	Time for transaction timeout.
timeoutMilli	Long	The timeout time in milliseconds; default is 3s.

Returns

```
StartTransactionResponseData(var respType: TransactionResponseType,var
attribute: UShort,var cardData: List)
```

Example

```
GlobalScope.launch(Dispatchers.IO) {
    Client.waitUntilConnected()
    if (connectedDeviceId == null) Returns@launch
    val setaacmd = Client.SetAutoAuthenticateAsync(connectedDeviceId!!
, true)
    val setaa_rv: ClientCommandStatus = setaacmd.waitForCompletionWith
Timeout(1000)
    Log.d(TAG, "Enable Auto Authenticate: ${setaa_rv.name}")
    val setaccmd = Client.SetAutoCompleteAsync(connectedDeviceId!!, tr
ue)
    val setac_rv = setaccmd.waitForCompletionWithTimeout(1000)
    Log.d(TAG, "Enable Auto Complete: ${setac_rv.name}")
    val transTimeout: UShort = 100u
    val startTransCmd = Client.StartMSRTransAsync(connectedDeviceId!!,
                transTimeout, 100000)
    startTransCmd.waitForCompletion()
    var startTransStatus = startTransCmd.getCommandStatus()
    Log.d(TAG, "Transaction Status: ${startTransStatus.name}")
    val resultData: StartTransactionResponseData? = startTransCmd.getR
esultData()
}
```

2.9. SetTransactionExponentAsync

```
fun Client.SetTransactionExponentAsync(deviceId: String,
transactionExponent: UShort, timeoutMilli: Long = 3000):
SetTransactionExponentCommand
```

Set the transaction exponent.

Parameters

Name	Type	Description
deviceId	String	Device ID.
transactionExponent	UShort	Transaction exponent.
timeoutMilli	Long	The timeout time in milliseconds; default is 3s.

Returns

```
UShort
```

Example

```
val setTransExponentCmd = Client.SetTransactionExponentAsync(connected
DeviceId!!, 0u)
val settransexponent_rv: ClientCommandStatus = setTransExponentCmd.wai
tForCompletionWithTimeout(1000)
```

2.10. AuthenticateTransactionAsync

```
fun Client.AuthenticateTransactionAsync(deviceId: String, goOnline:
Boolean, online_timeout: Long, outputTagList: Array<ByteArray?>,
timeoutMilli: Long = 3000): AuthenticateTransactionCommand
```

Authenticates a transaction asynchronously.

Parameters

Name	Type	Description
deviceId	String	Device ID.
goOnline	Boolean	If true, force the transaction to go online.
online_timeout	Long	The waiting time for a host response when online.
outputTagList	Array<ByteArray?>	The output tag list.
timeoutMilli	Long	The timeout time in milliseconds; default is 3s.

Returns

```
AuthenticateTransactionResponseData(var statusCode: UShort, var
attribute: UShort, var cardData: List)
```

Example

```
var outputTag : ByteArray? = null
val outputTagList = arrayOf(outputTag)
val authenticateTransactionCommand = Client.AuthenticateTransactionAsy
nc(connectedDeviceId!!, false, 120, outputTagList)
val authTran_rv = authenticateTransactionCommand.waitForCompletionWith
Timeout(40000)
```

2.11. DisplayOnlineAuthResultAsync

```
fun Client.DisplayOnlineAuthResultAsync(deviceId: String, StatusCode:
Int, OnlineAuthCode: UByteArray, TransactionDate: UByteArray,
TransactionTime: UByteArray, timeoutMilli: Long = 3000):
DisplayOnlineAuthResultCommand
```

Display on-line authentication results.

Parameters

Name	Type	Description
deviceId	String	Device ID.
StatusCode	Int	00: OK, 01: NOT OK, 02: (ARC response 89 for Interac).
OnlineAuthCode	UByteArray	Used for authorization; can be empty to use defalut value.
TransactionDate	UByteArray	Binary-Coded Decimal (BCD) Code; YYMMDD format, can be empty to use reader date.
TransactionTime	UByteArray	Binary-Coded Decimal (BCD) Code; HHMMSS format, can be empty to use reader time.
timeoutMilli	Long	The timeout time in milliseconds; default is 3s.

Returns

```
StartTransactionResponseData(var respType: TransactionResponseType,var
attribute: UShort,var cardData: List)
```

Example

```
val cmd = Client.DisplayOnlineAuthResultAsync(connectedDeviceId!!, 0,
byteArrayOf(0x44, 0x45, 0x41, 0x44, 0x42, 0x45, 0x41, 0x46),
byteArrayOf(0x22, 0x06, 0x11), byteArrayOf(0x12, 0x34, 0x34))
cmd.waitForCompletionWithTimeout(20000)
```

2.12. RetrieveTransactionResultAsync

```
fun Client.RetrieveTransactionResultAsync(deviceId: String, tags:
ByteArray, timeoutMilli: Long = 3000): RetrieveTransactionCommand
```

Retrive transaction results.

Parameters

Name	Type	Description
deviceId	String	Device ID.
tags	ByteArray	Tags in the transaction result.
timeoutMilli	Long	The timeout time in milliseconds; default is 3s.

Returns

```
RetrieveTransactionResponseData(var statusCode: UShort, var attribute:
UShort, var cardData: List)
```

Example

```
val retrieveTransCmd = Client.RetrieveTransactionResultAsync(connected
DeviceId!!, byteArrayOf( 71 ), 10000)
val retrieveTransCmd_rv = retrieveTransCmd.waitForCompletionWithTimeou
t(10000)
```

2.13. CompleteTransactionAsync

```
fun Client.CompleteTransactionAsync(deviceId: String, wentOnline:
Boolean, authCode: ByteArray, issuerAuthData: UByteArray, scriptsData:
UByteArray, outputData: UByteArray, timeoutMilli: Long = 3000):
CompleteTransactionCommand
```

Complete a transaction asynchronously.

Parameters

Name	Type	Description
deviceId	String	Device ID.
wentOnline	Boolean	If true, the reader went online with the host device.
authCode	ByteArray	Authorization response code.
issuerAuthData	UByteArray	Issuer authentication data in TLV data format.
scriptsData	UByteArray	Scripts in TLV data format.
outputData	UByteArray	Output data list.
timeoutMilli	Long	The timeout time in milliseconds; default is 3s.

Returns

```
StartTransactionResponseData(var respType: TransactionResponseType,var
attribute: UShort,var cardData: List)
```

Example

```
val completeTransCmd = Client.CompleteTransactionAsync(connectedDevice
Id!!,
                                                       true, byteArray
Of(0x30, 0x30))
val completeTransCmd_rv = completeTransCmd.waitForCompletionWithTimeou
t(1000)
```

2.14. CancelTransactionAsync

```
fun Client.CancelTransactionAsync(deviceId: String, mode: UShort,
timeoutMilli: Long = 3000): CancelTransactionCommand
```

Cancel a transaction asynchronously.

Parameters

Name	Type	Description
deviceId	String	Device ID.
mode	UShort	0: cancel mode. 1: Reset Mode, reset transaction status without displaying anything.
timeoutMilli	Long	The timeout time in milliseconds; default is 3s.

Returns

```
UShort
```

Example

```
val cancelTransCmd = Client.CancelTransactionAsync(connectedDeviceId!!
)
val status: ClientCommandStatus = cancelTransCmd.waitForCompletionWith
Timeout(10000)
```

2.15. SetAutoCompleteAsync

```
fun Client.SetAutoCompleteAsync(deviceId: String, enable: Boolean,
timeoutMilli: Long = 3000): SetAutoCompleteCommand
```

Set transactions to automatically complete.

Parameters

Name	Type	Description
deviceId	String	Device ID.
enable	Boolean	If true, transaction autocompletes.
timeoutMilli	Long	The timeout time in milliseconds; default is 3s.

Returns

```
EmptyPayloadType()
```

Example

```
val setaccmd = Client.SetAutoCompleteAsync(connectedDeviceId!!, true)
val setac_rv: ClientCommandStatus = setaccmd.waitForCompletionWithTime
out(1000)
```

2.16. GetAutoCompleteAsync

```
fun Client.GetAutoCompleteAsync(deviceId: String, timeoutMilli: Long =
3000): GetAutoCompleteCommand
```

Asynchronous version of **GetAutoCompleteCheck** if the device is set to automatically do **ContactCompleteTransaction** in EMV Transactions.

Parameters

Name	Type	Description
deviceId	String	Device ID.
timeoutMilli	Long	The timeout time in milliseconds; default is 3s.

Returns

```
Boolean
```

Example

```
if (connectedDeviceId == null) {
     outputInCoroutine("Not connected to any device")
   } else {
     GlobalScope.launch(Dispatchers.IO) {
        Client.waitUntilConnected()
        val cmd = Client.GetAutoCompleteAsync(connectedDeviceId!!)
        val status = cmd.waitForCompletionWithTimeout(3000)
        outputInCoroutine("GetAutoComplete ${cmd.getResultData()} stat
us: ${status.name}")
   }
}
```

2.17. SetAutoAuthenticateAsync

```
fun Client.SetAutoAuthenticateAsync(deviceId: String, enable: Boolean,
timeoutMilli: Long = 3000): SetAutoAuthenticateCommand
```

Set transactions to automatically authenticate.

Parameters

Name	Type	Description
deviceId	String	Device ID.
enable	Boolean	If true, transaction autoauthenticates.
timeoutMilli	Long	The timeout time in milliseconds; default is 3s.

Returns

```
EmptyPayloadType()
```

Example

```
val setaacmd = Client.SetAutoAuthenticateAsync(connectedDeviceId!!, tr
ue)
val setaa_rv: ClientCommandStatus = setaacmd.waitForCompletionWithTime
out(1000)
```

2.18. GetAutoAuthenticateAsync

```
fun Client.GetAutoAuthenticateAsync(deviceId: String, timeoutMilli:
Long = 3000): GetAutoAuthenticateCommand
```

Asynchronous version of **GetAutoAuthenticateCheck** if the device is set to automatically do **ContactAuthenticateTransaction** in EMV transactions.

Parameters

Name	Type	Description
deviceId	String	Device ID.
timeoutMilli	Long	The timeout time in milliseconds; default is 3s.

Returns

```
Boolean
```

Example

```
if (connectedDeviceId == null) {
     outputInCoroutine("Not connected to any device")
   } else {
     GlobalScope.launch(Dispatchers.IO) {
        Client.waitUntilConnected()
        val cmd = Client.GetAutoAuthenticateAsync(connectedDeviceId!!)
        val status = cmd.waitForCompletionWithTimeout(3000)
        outputInCoroutine("GetAutoAuthenticate ${cmd.getResultData()}
status: ${status.name}")
   }
}
```

3. General Commands

The following section describes general commands.

3.1. MiscPingDeviceAsync

```
fun Client.MiscPingDeviceAsync(deviceId: String, timeoutMilli: Long =
3000): MiscPingDeviceCommand
```

Ping the reader.

Parameters

Name	Type	Description
deviceId	String	Device ID.
timeoutMilli	Long	The timeout time in milliseconds; default is 3s.

Returns

```
Status
```

Example

```
GlobalScope.launch(Dispatchers.IO) {
    Client.waitUntilConnected()
    val cmd = Client.MiscPingDeviceAsync(connectedDeviceId!!)
    val result = cmd.waitForCompletion()
    Log.d(TAG, "Ping status: ${result.name}")
}
```

3.2. GetSerialNumberAsync

```
fun Client.GetSerialNumberAsync(deviceId: String, timeoutMilli: Long =
3000): GetDeviceSerialNumberCommand
```

Get the reader's IDG serial number.

Parameters

Name	Type	Description
deviceId	String	Device ID.
timeoutMilli	Long	The timeout time in milliseconds; default is 3s.

Example

```
val getsncmd = Client.GetSerialNumberAsync(connectedDeviceId!!)
val status: ClientCommandStatus = getsncmd.waitForCompletionWithTimeou
t(20000)
val serialNumberData: String? = getsncmd.getResultData()
```

3.3. GetDRSInfoAsync

```
fun Client.GetDRSInfoAsync(deviceId: String, timeoutMilli: Long =
3000): DRSInfoCommand
```

Get the reader's DRS information.

Parameters

Name	Type	Description
deviceId	String	Device ID.
timeoutMilli	Long	The timeout time in milliseconds; default is 3s.

Returns

```
DRSInfoResult(var statusCode: UByte, var ResponseData: List) class
DRSInfo(var DRSSource: Int, var DRSData: Int)
```

Example

```
val drsInfoCommand = Client.GetDRSInfoAsync(connectedDeviceId!!)
val getDRSInfoCmd_rv = getDRSInfosCmd.waitForCompletionWithTimeout(100
00)
```

3.4. SetConfigurationDefaultsAsync

```
fun Client.SetConfigurationDefaultsAsync(deviceId: String,
timeoutMilli: Long = 3000): SetConfigurationDefaultsCommand
```

Set configuration defaults.

Parameters

Name	Type	Description
deviceId	String	Device ID.
timeoutMilli	Long	The timeout time in milliseconds; default is 3s.

Returns

```
Status
```

Example

```
val cmd = Client.SetConfigurationDefaultsAsync(connectedDeviceId!!)
val status = cmd.waitForCompletionWithTimeout(10000)
```

3.5. GetKernelVersionAsync

```
fun Client.GetKernelVersionAsync(deviceId: String, timeoutMilli: Long
= 3000): GetKernelVersionCommand
```

Get the reader's EMV kernel version.

Parameters

Name	Type	Description
deviceId	String	Device ID.
timeoutMilli	Long	The timeout time in milliseconds; default is 3s.

Returns

```
Status
```

Example

```
GlobalScope.launch(Dispatchers.IO) {
    Client.waitUntilConnected()
    val cmd = Client.GetKernelVersionAsync(connectedDeviceId!!)
    cmd.waitForCompletion()
    val status = cmd.getCommandStatus().name
    val result = cmd.getResultData()
    outputInCoroutine("GetKernelVersion $status KernelVersion: $result
")
}
```

3.6. GetDeviceTreeVersionAsync

```
fun Client.GetDeviceTreeVersionAsync(deviceId: String, type:
DeviceTreeType, timeoutMilli: Long = 3000):
GetDeviceTreeVersionCommand
```

Get the reader's device tree version.

Parameters

Name	Type	Description
deviceId	String	Device ID.
type	String	The device type in the device tree.
timeoutMilli	Long	The timeout time in milliseconds; default is 3s.

Example

```
GlobalScope.launch(Dispatchers.IO) {
    Client.waitUntilConnected()
    val cmd = Client.GetDeviceTreeVersionAsync(connectedDeviceId!!,
        DeviceTreeType.GetVersionOfSecureDeviceTreeInK81)
    val status = cmd.waitForCompletionWithTimeout(20000)
    val result = cmd.getResultData()
    outputInCoroutine("SerializeGetDeviceTreeVersion ${status.name}; "
 +
        "DeviceTreeVersion: $result")
}
```

3.7. MiscReboot

```
fun Client.MiscReboot(deviceId: String, timeoutMilli: Long = 3000):
MiscRebootCommand
```

Reboot the reader.

Parameters

Name	Type	Description
deviceId	String	Device ID.
timeoutMilli	Long	The timeout time in milliseconds; default is 3s.

Returns

```
Status
```

Example

```
GlobalScope.launch(Dispatchers.IO) {
    Client.waitUntilConnected()
    val cmd = Client.MiscReboot(connectedDeviceId!!)
    val result = cmd.waitForCompletion()
    Log.d(TAG, "Reboot status: ${result.name}")
}
```

4. Miscellaneous Commands

The following section describes misellaneous commands.

4.1. StartRKIAsync

```
fun Client.StartRKIAsync(deviceId: String, timeoutMilli: Long = 3000):
PkiRKICommand
```

Start remote key injection.

Parameters

Name	Type	Description
deviceId	String	Device ID.
timeoutMilli	Long	The timeout time in milliseconds; default is 3s.

4.2. MiscDumpOutputLogAsync

```
fun Client.MiscDumpOutputLogAsync(deviceId: String, timeoutMilli: Long
= 3000): MiscDumpOutputLogCommand
```

Dump the output log.

Parameters

Name	Type	Description
deviceId	String	Device ID.
timeoutMilli	Long	The timeout time in milliseconds; default is 3s.

Returns

```
Status
```

Example

```
GlobalScope.launch(Dispatchers.IO) {
    Client.waitUntilConnected()
    val cmd = Client.MiscDumpOutputLogAsync(connectedDeviceId!!)
    val result = cmd.waitForCompletion()
    Log.d(TAG, "Reboot status: ${result.name}")
}
```

4.3. ReceiveGUINotificationsAsync

```
fun Client.ReceiveGUINotificationsAsync(deviceId: String,
timeoutMilli: Long = 3000): ReceiveGUINotificationsCommand
```

Receive GUI notifications.

Parameters

Name	Type	Description
deviceId	String	Device ID.
timeoutMilli	Long	The timeout time in milliseconds; default is 3s.

Returns

```
EmptyPayloadType()
```

Example

```
var receivingGUINotification = false
GlobalScope.launch(Dispatchers.IO) {
    Client.waitUntilConnected()
    val guiEventCmd = Client.ReceiveGUINotificationsAsync(connectedDev
iceId!!)
    receivingGUINotification = true
    while(receivingGUINotification) {
        guiEventCmd!!.waitForEventWithTimeout(Long.MAX_VALUE)
        val progressList = guiEventCmd!!.receiveCommandProgress()
        for (guiEvent in progressList) {
            if (guiEvent.Status == CommandResponseType.Executing) {
                val payload = guiEvent.Payload as PayloadData<GuiNotif
icationInfo>?
                val guiEventPalyoad:GuiNotificationInfo? = payload?.ge
tPayloadData()
                if (guiEventPalyoad != null) {
                    when (GUIResponse.from(guiEventPalyoad.responseTyp
e)) {
                        GUIResponse.ButtonEvent -> {
                            Log.d(TAG, "screen id/name: ${guiEventPaly
oad.screenID}/${guiEventPalyoad.screenName}")
                        }
                        GUIResponse.LCDMessage -> {
                            //output("LCD Code/Message: ${guiEventPaly
oad.lcdCode}/${guiEventPalyoad.lcdMessage}")
                            val responseMode: UByte = guiEventPalyoad.
lcdData.displayMode
                            //val message = guiEventPalyoad.lcdData
                            var led_id : UByte = 0u
                            var led_state : UByte = 0u
                            if(guiEventPalyoad.lcdData.displayMode==Mo
de.DISP_UPDATE_LED) {
                                led_id = guiEventPalyoad.lcdData.ledSt
```

```
ateInfo.ledId
                                led_state = guiEventPalyoad.lcdData.le
dStateInfo.ledState
                                Log.d(TAG, "led_id=$led_id")
                                Log.d(TAG, "led_state=$led_state")
                            }
                        }
                    }
                }
            } else if (guiEvent.Status == CommandResponseType.Started)
 {
                Log.d(TAG, "receive gui event command state: Started")
            }
        }
    }
    receivingGUINotification = false
    guiEventCmd?.cancelCommandAsync()
}
```

4.4. SendRawdataToReader

```
fun Client.SendRawdataToReader(deviceId: String, rawData: UByteArray,
timeoutMilli: Long = 30000): SendRawdataToReaderCommand
```

Send a raw command with attribution to the reader's K81 processor.

Parameters

Name	Type	Description
deviceId	String	Device ID.
rawData	UByteArray	The command including header and CRC.
timeoutMilli	Long	The timeout time in milliseconds; default is 3s.

Example

```
GlobalScope.launch(Dispatchers.IO) {
     Client.waitUntilConnected()
     val data = getV2FormatData(0x00.toUByte(), listOf<UByte>(),0x29).
toUByteArray()
     val cmd = Client.SendRawdataToReader(connectedDeviceId,data)
     val status = cmd.waitForCompletionWithTimeout(3000)
     output = "sendRawdataToReader: ${status.name}"
     }
```

4.5. MiscSendDataCommandNEOAsync

```
fun Client.MiscSendDataCommandNEOAsync(deviceId: String, cmd: UByte,
subCmd: UByte, data: UByteArray?, timeoutMilli: Long = 30000):
MiscSendDataCommandNEOCommand
```

Send a raw NEO command to the reader's K81 processor.

Parameters

Name	Type	Description
deviceId	String	Device ID.
cmd	UByte	Command number.
subCmd	UByte	Subcommand number.
timeoutMilli	Long	The timeout time in milliseconds; default is 3s.

Returns

```
DataResponseNEO
```

Example

```
GlobalScope.launch(Dispatchers.IO) {
    val miscSendDataCommandNEOcmd = Client.MiscSendDataCommandNEOAsync
(connectedDeviceId!!, 0x29u, 0x00u, null)
    while(true) {
        miscSendDataCommandNEOcmd.waitForEventWithTimeout(Long.MAX_VAL
UE)
        if (miscSendDataCommandNEOcmd.getCommandStatus() == Executing)
 {
            val progressList =
            miscSendDataCommandNEOcmd.receiveCommandProgress()
            val frame = (progressList.last().Payload as PayloadData<Da
taResponseNEO>).getPayloadData().responseFrames.last()
            outputInCoroutine(
                "Frame: ${progressList.size}    " +
                "statusCode: ${frame.statusCode}    " +
                "responseData: ${frame.data.toByteArray().toString(US_
ASCII)}")
        } else {
            val commandStatus = miscSendDataCommandNEOcmd.getCommandSt
atus()
            val frame = miscSendDataCommandNEOcmd.getResultData()?.res
ponseFrames?.last()
            outputInCoroutine("Final Status: ${commandStatus.name}
" +
                "statusCode: ${frame?.statusCode}    " +
                "responseData: ${frame?.data?.toByteArray()?.toString(
US_ASCII)}")
            break
        }
```

```
    }
}
```

Definition of **ResponseFrame** class:

```
class ResponseFrame(
    val statusCode: UByte,
    val data: UByteArray
)
```

Definition of **DataResponseNEO** class:

```
class DataResponseNEO(
    val responseFrames: List<ResponseFrame>
)
```

4.6. SendDataCommandNEOExtraAsync

```
fun Client.SendDataCommandNEOExtraAsync(deviceId: String, cmd: UByte,
subCmd: UByte, data: UByteArray?, attribution: UByte, timeoutMilli:
Long = 30000): SendDataCommandNEOExtraCommand
```

Send a raw command with attribution to the reader's K81 processor.

Parameters

Name	Type	Description
deviceId	String	Device ID.
cmd	UByte	Command number.
subCmd	UByte	Subcommand number.
data	UByteArray?	Command data.
attribution	UByte	Command attribution.
timeoutMilli	Long	The timeout time in milliseconds; default is 3s.

Returns

```
DataResponseNEO
```

Example

```
GlobalScope.launch(Dispatchers.IO) {
    Client.waitUntilConnected()
    val sendDataCommandNEOExtraAsync = Client.SendDataCommandNEOExtraA
sync(
        connectedDeviceId!!, 0x29u, 0x00u, null, 0x00u)
    while (true) {
        sendDataCommandNEOExtraAsync.waitForEventWithTimeout(Long.MAX_
VALUE)
        if (sendDataCommandNEOExtraAsync.getCommandStatus() == Executi
ng) {
            val progressList = sendDataCommandNEOExtraAsync.receiveCom
mandProgress()
```

```
            val frame = (progressList.last().Payload as PayloadData<Da
taResponseNEO>)
                .getPayloadData().responseFrames.last()
            outputInCoroutine(
                "Frame: ${progressList.size}    " +
                "statusCode: ${frame.statusCode}    " +
                "responseData: ${frame.data.toByteArray().toString(US_
ASCII)}")
        } else {
            val commandStatus = sendDataCommandNEOExtraAsync.getComman
dStatus()
            val frame = sendDataCommandNEOExtraAsync.getResultData()?.
responseFrames?.last()
            outputInCoroutine(
                "Final Status: ${commandStatus.name}    " +
                "statusCode: ${frame?.statusCode}    " +
                "responseData: ${frame?.data?.toByteArray()?.toString(
US_ASCII)}")
            break
        }
    }
}
```

Definition of **ResponseFrame** class:

```
class ResponseFrame(
    val statusCode: UByte,
    val data: UByteArray
)
```

Definition of **DataResponseNEO** class:

```
class DataResponseNEO(
    val responseFrames: List<ResponseFrame>
)
```

5. Reader Configuration Commands

The section below describes API methods for configuring AP6800 readers.

5.1. GetBootloaderVersionAsync

```
fun Client.GetBootloaderVersionAsync(deviceId: String, timeoutMilli:
Long = 3000): GetBootLoaderVersionCommand
```

Get the reader's bootloader version.

Parameters

Name	Type	Description
deviceId	String	Device ID.
timeoutMilli	Long	The timeout time in milliseconds; default is 3s.

Returns

The boot loader version in string format.

Example

```
GlobalScope.launch(Dispatchers.IO) {
     Client.waitUntilConnected()
     val cmd = Client.GetBootloaderVersionAsync(connectedDeviceId!!)
     cmd.waitForCompletion()
     val result = cmd.getResultData()
     Log.d(TAG, "GetBootloaderVersion: $result")
}
```

5.2. EnterBootLoaderProcessAsync

```
fun Client.EnterBootLoaderProcessAsync(deviceId: String, timeoutMilli:
Long = 3000): EnterBootLoaderProcessFromMainApplicationCommand
```

Enter the boot loader.

Parameters

Name	Type	Description
deviceId	String	Device ID.
timeoutMilli	Long	The timeout time in milliseconds; default is 3s.

Example

```
val cmd = Client.EnterBootLoaderProcessAsync(connectedDeviceId!!)
val status: ClientCommandStatus = cmd.waitForCompletionWithTimeout(100
00)
```

Returns

```
EmptyPayloadType()
```

5.3. GetServerFeaturesAsync

```
fun Client.GetServerFeaturesAsync(timeoutMilli: Long = 3000):
GetServerInfoCommand
```

Get the reader's server information command.

Parameters

Name	Type	Description
timeoutMilli	Long	The timeout time in milliseconds; default is 3s.

Example

```
val cmd = Client.GetServerFeaturesAsync()
val status = cmd.waitForCompletionWithTimeout(3000)
```

5.4. SetAllowFallbackAsync

```
fun Client.SetAllowFallbackAsync(deviceId: String, allowFallback:
Boolean, timeoutMilli: Long = 3000): SetAllowFallbackCommand
```

Asynchronous version of SetAllowFallback; if set to **True**, the reader falls back to MSR.

Parameters

Name	Type	Description
deviceId	String	Device ID.
allowFallBack	Boolean	If true, allow the reader to fall back to MSR.
timeoutMilli	Long	The timeout time in milliseconds; default is 3s.

Example

```
if (connectedDeviceId == null) {
      outputInCoroutine("Not connected to any device")
   } else {
     GlobalScope.launch(Dispatchers.IO) {
         Client.waitUntilConnected()
         val cmd = Client.SetAllowFallbackAsync(connectedDeviceId!!,tr
ue)
         val status = cmd.waitForCompletionWithTimeout(3000)
         outputInCoroutine("SetAllowFallback status: ${status.name}")
    }
}
```

5.5. SetCertificationRevocationListAsync

```
fun Client.SetCertificationRevocationListAsync(deviceId: String, crls:
Array<String>, timeoutMilli: Long = 3000):
ContactSetCertificationRevocationListCommand
```

Set the reader's certificate revocation list.

Parameters

Name	Type	Description
deviceId	String	Device ID.
crls	Array<String>	Array of certifications to revoke.
timeoutMilli	Long	The timeout time in milliseconds; default is 3s.

Returns

```
UByte
```

Example

```
val crls = arrayOf("a00000000350014455", "a000000004fe092355")
val cmd = Client.SetCertificationRevocationListAsync( connectedDeviceI
d!!, crls)
cmd.waitForCompletionWithTimeout(20000)
```

5.6. RetrieveCertificationRevocationListAsync

```
fun Client.RetrieveCertificationRevocationListAsync(deviceId: String,
timeoutMilli: Long = 3000):
ContactRetrieveCertificationRevocationListCommand
```

Get the reader's certificate revocation list.

Parameters

Name	Type	Description
deviceId	String	Device ID.
timeoutMilli	Long	The timeout time in milliseconds; default is 3s.

Returns

```
EMVCRLResult(var statusCode: UByte, var crls: Array)
```

Example

```
val ContactRetrieveCertificationRevocationListCommand = Client.Retriev
eCertificationRevocationListAsync(connectedDeviceId!!)
cmd.waitForCompletionWithTimeout(20000)
```

5.7. RemoveCertificationRevocationListAsync

```
fun Client.RemoveCertificationRevocationListAsync(deviceId: String,
timeoutMilli: Long = 3000):
ContactRemoveCertificationRevocationListCommand
```

Remove the reader's certification revocation list.

Parameters

Name	Type	Description
deviceId	String	Device ID.
timeoutMilli	Long	The timeout time in milliseconds; default is 3s.

Returns

```
EMVCRLResult(var statusCode: UByte, var crls: Array)
```

Example

```
val ContactRemoveCertificationRevocationListCommand = Client.RemoveCer
tificationRevocationListAsync(connectedDeviceId!!)
cmd.waitForCompletionWithTimeout(20000)
```

5.8. GetDevicesAsync

```
fun Client.GetDevicesAsync(timeoutMilli: Long = 3000):
GetDevicesCommand
```

Get all commands used on the reader.

Parameters

Name	Type	Description
timeoutMilli	Long	The timeout time in milliseconds; default is 3s.

Example

```
val cmd: GetDevicesCommand = Client.GetDevicesAsync()
val status: ClientCommandStatus = cmd.waitForCompletionWithTimeout(300
0)
val devices: MutableList<String> = cmd.devices
cmd.devices.forEachIndexed { i, info ->
    Log.d(TAG,
        "device [$i]    deviceId: $info") }
```

5.9. GetDeviceStatusAsync

```
fun Client.GetDeviceStatusAsync(deviceId: String, timeoutMilli: Long =
3000): GetDeviceStatusCommand
```

Get the reader's activation status and lock status.

Parameters

Name	Type	Description
deviceId	String	Device ID.
timeoutMilli	Long	The timeout time in milliseconds; default is 3s.

Returns

```
DeviceStatus
```

Example

```
GlobalScope.launch(Dispatchers.IO) {
    Client.waitUntilConnected()
    val cmd = Client.GetDeviceStatusAsync(connectedDeviceId!!)
    cmd.waitForCompletionWithTimeout(3000)
    val result: DeviceStatus?  = cmd.getResultData()
}
```

5.10. GetKernelCheckValueAsync

```
fun Client.GetKernelCheckValueAsync(deviceId: String, timeoutMilli:
Long = 3000): GetKernelCheckValueCommand
```

Get the reade's kernel check value.

Parameters

Name	Type	Description
deviceId	String	Device ID.
timeoutMilli	Long	The timeout time in milliseconds; default is 3s.

Returns

```
Status
```

Example

```
val cmd = Client.GetKernelCheckValueAsync(connectedDeviceId!!)
cmd.waitForCompletionWithTimeout(20000)
val status = cmd.getCommandStatus().name
val result = cmd.getResultData()
```

5.11. ListenUnexpectedInputRawdataFromReader

```
fun Client.ListenUnexpectedInputRawdataFromReader(deviceId: String,
timeoutMilli: Long = 3000):
ListenUnexpectedInputRawdataFromReaderCommand
```

Listener for all raw commands from the reader's K81 processor.

Parameters

Name	Type	Description
deviceId	String	Device ID.
timeoutMilli	Long	The timeout time in milliseconds; default is 3s.

Returns

```
UByteArray
```

Example

```
                if (connectedDeviceId == null) {
                        outputInCoroutine("Not connected to any device
")
                    } else {
                        receivingRawdataFromReader = false
                        GlobalScope.launch(Dispatchers.IO) {
                            Client.waitUntilConnected()
                            listenInputRawdataFromReaderCommand?.cance
lCommandAsync()?.waitForCompletionWithTimeout(5000)
                            listenInputRawdataFromReaderCommand = Clie
nt.ListenInputRawdataFromReader(
                                connectedDeviceId!!
                            )
                            receivingRawdataFromReader = true
                            var previousProgressListSize = 0
                            while(receivingRawdataFromReader){
                                listenInputRawdataFromReaderCommand!!.
waitForEventWithTimeout(Long.MAX_VALUE,previousProgressListSize)
                                val progressList = listenInputRawdataF
romReaderCommand!!.receiveCommandProgress()
                                val size = progressList.size
                                for (index in previousProgressListSize
 until size) {
                                    val rowData = progressList[index]
                                    if (rowData.Status == CommandRespo
nseType.Executing) {
                                        val payloadData = rowData.Payl
oad as PayloadData<UByteArray>
                                        outputInCoroutine("ListenInput
RawdataFromReader ${payloadData.getPayloadData().toHexString()}")
                                    }
                                }
```

```
                                previousProgressListSize = size
                            }
                        }
                    }
```

5.12. AutoResendListenInputRawdataFromReader

```
fun Client.AutoResendListenInputRawdataFromReader(deviceId: String,
timeoutMilli: Long = 3000):
AutoResendListenUnexpectedInputRawdataFromReaderCommand
```

An automatic version of **ListenInputRawdataFromReader** that automatically listens when ZSDK-client reconnects to ZSDK-server.

Parameters

Name	Type	Description
deviceId	String	Device ID.
timeoutMilli	Long	The timeout time in milliseconds; default is 3s.

Returns

```
UByteArray
```

Example

```
                if (connectedDeviceId == null) {
                    outputInCoroutine("Not connected to any device
")
                } else {
                    GlobalScope.launch(Dispatchers.IO) {
                        autoReceivingRawdata = false
                        Client.waitUntilConnected()
                        autoReconnectRawDataEventCmd?.cancelComman
dAsync()?.waitForCompletion()
                        autoReconnectRawDataEventCmd = Client.Auto
ResendListenInputRawdataFromReader(
                            connectedDeviceId!!
                        )
                        autoReceivingRawdata = true
                        var previousProgressListSize = 0
                        while(autoReceivingRawdata){
                            autoReconnectRawDataEventCmd!!.waitFor
EventWithTimeout(Long.MAX_VALUE,previousProgressListSize)
                            val progressList = autoReconnectRawDat
aEventCmd!!.receiveCommandProgress()
                            val size = progressList.size
                            for (index in previousProgressListSize
 until size) {
                                val rowData = progressList[index]
                                if (rowData.Status == CommandRespo
nseType.Executing) {
```

```
                                        val payloadData = rowData.Payl
oad as PayloadData<UByteArray>
                                        outputInCoroutine("AutoResendL
istenInputRawdataFromReader ${payloadData.getPayloadData().toHexString
()}")
                                    }
                                }
                                previousProgressListSize = size
                            }
                        }
                    }
```

5.13. ApplyConciergePackageAsync

```
fun Client.ApplyConciergePackageAsync(deviceId: String, bytes:
ByteArray, mode: ConciergePackageApplyMode, timeoutMilli: Long = 3000,
ignorePidVidCheck: Boolean = false): ConciergePackageApplyCommand
```

Applies a Concierge package.

Parameters

Name	Type	Description
deviceId	String	Device ID.
bytes	ByteArray	The package information.
Mode	ConceirgePackageApplyMode	Puts the reader in the mode to apply packages.
timeoutMilli	Long	The timeout time in milliseconds; default is 3s.
ignorePidVidCheck	Boolean	If true, the reader ignores checks when applying packages.

Returns

```
EmptyPayloadType()
```

Example

```
val bytes = readAsset(context, filename)
GlobalScope.launch(Dispatchers.IO) {
    Client.waitUntilConnected()
    val applyPackageCmd = Client.ApplyConciergePackageAsync(connectedD
eviceId!!, bytes, ConciergePackageApplyMode.Auto)
    while (!applyPackageCmd.isCompleted()) {
        Log.d(TAG, "!applyPackageCmd.isCompleted()")
        val receivedProgressList = applyPackageCmd.receiveCommandProgr
ess()
        if (receivedProgressList.isNotEmpty()) {
            Log.d(TAG, "receivedProgressList.isNotEmpty()")
            val progress = receivedProgressList.last()
            when (progress.Status) {
                CommandResponseType.Started,
                CommandResponseType.Executing -> {
```

```
                    Log.d(TAG, "CommandResponseType.${progress.Status.
name}")
                    if (progress.Progress != null) {
                        Log.d(TAG, "progress.Progress != null")
                        val innerProgress = progress.Progress
                        Log.d(TAG, "COMMAND PROGRESS: ${innerProgress!
!.CommandName}  ${innerProgress.ProgressCurrent}/${innerProgress.Progr
essFull}")
                        if (innerProgress.Stages!!.isNotEmpty()) {
                            Log.d(TAG, "innerProgress.Stages!!.isNotEm
pty()")
                            val stage = innerProgress.Stages.last()
                            Log.d(TAG, "STAGE[${stage.StageId}] ${stag
e.StageName} ${stage.StageProgressCurrent}/${stage.StageProgressFull}
${stage.StageState}")
                        }
                    }
                } else -> {}
            }
        }
        applyPackageCmd.waitForEventWithTimeout(1000)
    }
    if (applyPackageCmd.getCommandStatus() == Completed) {
        Log.d(TAG, "Completed!")
    } else {
        val errInfo = applyPackageCmd.getErrorInformation()
        Log.d(TAG, applyPackageCmd.getCommandStatus().name)
        if (errInfo != null) {
            Log.d(TAG, "Error Message: ${errInfo.ErrorMessage}  Error
Code: ${errInfo.ErrorCode}")
            //todo: add function -- GetErrorCodeStringRepresentation(e
rrorCode)
        } else {
            Log.d(TAG, "no error details")
        }
    }
}
```

6. Key Management Commands

The following section describes key management commands.

6.1. GetAllCAPublicRIDs

```
fun Client.GetAllCAPublicRIDs(deviceId: String, timeoutMilli: Long =
3000): GetAllCAPublicRIDsCommand
```

Retrieve all the reader's CA public RIDs.

Parameters

Name	Type	Description
deviceId	String	Device ID.
timeoutMilli	Long	The timeout time in milliseconds; default is 3s.

Example

```
val getAllCAPublicRIDs = GetAllCAPublicRIDs(deviceId,timeoutMilli)
val getAllCAPublicRIDsStatus = getAllCAPublicRIDs.waitForCompletionWit
hTimeout(20000)
```

6.2. SetCtlsCAPublicKeyAsync

```
fun Client.SetCtlsCAPublicKeyAsync(deviceId: String, keyId: String,
capk: ByteArray, timeoutMilli: Long = 3000): SetCAPublicKeyCommand
```

Sets the CTLS CAPK.

Parameters

Name	Type	Description
deviceId	String	Device ID.
keyId	String	Key ID to set.
capk	ByteArray	The CAPK to set.
timeoutMilli	Long	The timeout time in milliseconds; default is 3s.

Returns

```
UByte
```

Example

```
GlobalScope.launch(Dispatchers.IO) {
    val keyId = "a00000000350"
    val capk = "0101b769775668cacb5d22a647d1d993141edab7237b0001000180
00d11197590057b84196c2f4d11a8f3c05408f422a35d702f90106ea5b019bb28ae607
aa9cdebcd0d81a38d48c7ebb0062d287369ec0c42124246ac30d80cd602ab7238d5108
4ded4698162c59d25eac1e66255b4db2352526ef0982c3b8ad3d1cce85b01db5788e75
e09f44be7361366def9d1e1317b05e5d0ff5290f88a0db47"
```

```
    Client.waitUntilConnected()
    val cmd = Client.SetCtlsCAPublicKeyAsync(connectedDeviceId!!, keyI
d, capk.decodeHex())
    val status = cmd.waitForCompletionWithTimeout(20000)
    outputInCoroutine("SetCtlsCAPublicKey: ${status.name}")
}
```

6.3. RemoveCtlsCAPublicKeyAsync

```
fun Client.RemoveCtlsCAPublicKeyAsync(deviceId: String, keyId: String,
timeoutMilli: Long = 3000): DeleteCAPublicKeyCommand
```

Remove the reader's CTLS CAPK.

Parameters

Name	Type	Description
deviceId	String	Device ID.
timeoutMilli	Long	The timeout time in milliseconds; default is 3s.

Returns

```
UByte
```

Example

```
GlobalScope.launch(Dispatchers.IO) {
    val keyId = "a00000000350"
    Client.waitUntilConnected()
    val cmd = Client.RemoveCtlsCAPublicKeyAsync(connectedDeviceId!!, k
eyId)
    val status = cmd.waitForCompletionWithTimeout(20000)
    outputInCoroutine("RemoveCtlsCAPublicKey: ${status.name}")
}
```

6.4. RetrieveCtlsCAPublicKeyAsync

```
fun Client.RetrieveCtlsCAPublicKeyAsync(deviceId: String, keyId:
String, timeoutMilli: Long = 3000): GetCAPublicKeyCommand
```

Retrieve the reader's CTLS CAPK.

Parameters

Name	Type	Description
deviceId	String	Device ID.
timeoutMilli	Long	The timeout time in milliseconds; default is 3s.

Returns

```
UByte
```

Example

```
GlobalScope.launch(Dispatchers.IO) {
    val keyId = "a00000000350"
    Client.waitUntilConnected()
    val cmd = Client.RetrieveCtlsCAPublicKeyAsync(connectedDeviceId!!,
 keyId)
    val status = cmd.waitForCompletionWithTimeout(20000)
    val result = cmd.getResultData()
    outputInCoroutine("RetrieveCtlsCAPublicKey: ${status.name}; CtlsCA
PublicKey: ${result?.toHexString()}")
}
```

6.5. SetEMVCAPublicKeyAsync

```
fun Client.SetEMVCAPublicKeyAsync(deviceId: String, keyid: String,
capk: ByteArray, timeoutMilli: Long = 3000):
ContactSetCAPublicKeyCommand
```

Set the EMV CA public key.

Parameters

Name	Type	Description
deviceId	String	Device ID.
keyid	String	The key ID to set.
capk	ByteArray	The CAPK to set.
timeoutMilli	Long	The timeout time in milliseconds; default is 3s.

Returns

```
UByte
```

Example

```
val cmd = Client.SetEMVCAPublicKeyAsync(
    connectedDeviceId!!,
    "a00000000350",
    "0101b769775668cacb5d22a647d1d993141edab7237b000100018000d11197590
057b84196c2f4d11a8f3c05408f422a35d702f90106ea5b019bb28ae607aa9cdebcd0d
81a38d48c7ebb0062d287369ec0c42124246ac30d80cd602ab7238d51084ded4698162
c59d25eac1e66255b4db2352526ef0982c3b8ad3d1cce85b01db5788e75e09f44be736
1366def9d1e1317b05e5d0ff5290f88a0db47".decodeHex()
)
val status: ClientCommandStatus = cmd.waitForCompletionWithTimeout(200
00)
```

6.6. RetrieveCtlsCAPublicKeyIDsList

```
suspend fun Client.RetrieveCtlsCAPublicKeyIDsList(deviceId: String,
timeoutMilli: Long = 3000): CAPublicKeyIDsResult
```

Retrieve the reader's CTLS CAPK ID list.

Parameters

Name	Type	Description
deviceId	String	Device ID.
timeoutMilli	Long	The timeout time in milliseconds; default is 3s.

Returns

```
CAPublicKeyIDsResult
```

Example

```
GlobalScope.launch(Dispatchers.IO) {
    val res = Client.RetrieveCtlsCAPublicKeyIDsList(connectedDeviceId!
!)
    res.caPublicKeyIDs.forEach{
            outputInCoroutine("RetrieveCtlsCAPublicKeyIDsList CtlsCAPu
blicKeyID: $it")
    }
}
```

6.7. RemoveAllCtlsCAPublicKeysAsync

```
fun Client.RemoveAllCtlsCAPublicKeysAsync(deviceId: String,
timeoutMilli: Long = 3000): DeleteAllCAPublicKeysCommand
```

Remove all CTLS CAPKs.

Parameters

Name	Type	Description
deviceId	String	Device ID.
timeoutMilli	Long	The timeout time in milliseconds; default is 3s.

Returns

```
UByte
```

Example

```
GlobalScope.launch(Dispatchers.IO) {
    Client.waitUntilConnected()
    val cmd = Client.RemoveAllCtlsCAPublicKeysAsync(connectedDeviceId!
!)
    val status = cmd.waitForCompletionWithTimeout(20000)
    outputInCoroutine("RemoveAllCtlsCAPublicKeys: ${status.name}")
}
```

6.8. RetrieveEMVCAPublicKeyListAsync

```
fun Client.RetrieveEMVCAPublicKeyListAsync(deviceId: String,
timeoutMilli: Long = 3000): ContactRetrieveCAPublicKeyListCommand
```

Retrieve the reader's EMV CA public key list.

Parameters

Name	Type	Description
deviceId	String	Device ID.
timeoutMilli	Long	The timeout time in milliseconds; default is 3s.

Returns

```
EMVCAPublicKeyIDsResult(var statusCode: UByte,var caPublicKeyIDs:
Array)
```

Example

```
val cmd = Client.RetrieveEMVCAPublicKeyListAsync(connectedDeviceId!!)
val status: ClientCommandStatus = cmd.waitForCompletionWithTimeout(200
00)
val result: EMVCAPublicKeyIDsResult = cmd.getResultData()
```

6.9. RemoveEMVCAPublicKeyAsync

```
fun Client.RemoveEMVCAPublicKeyAsync(deviceId: String, aid: String,
timeoutMilli: Long = 3000): ContactRemoveCAPublicKeyCommand
```

Remove the reader's EMV CA public key.

Parameters

Name	Type	Description
deviceId	String	Device ID.
aid	String	The CA public key to remove.
timeoutMilli	Long	The timeout time in milliseconds; default is 3s.

Returns

```
UByte
```

Example

```
val cmd = Client.RemoveEMVCAPublicKeyAsync(connectedDeviceId!!, "a0000
0000350")
val status: ClientCommandStatus = cmd.waitForCompletionWithTimeout(200
00)
```

6.10. ListCAPublicKeyIDs

```
fun Client.ListCAPublicKeyIDs(deviceId: String, rid: String,
timeoutMilli: Long = 3000): ListCAPublicKeyIDsCommand
```

List the reader's CA public key RIDs.

Parameters

Name	Type	Description
deviceId	String	Device ID.
rid	String	The RID on the device.
timeoutMilli	Long	The timeout time in milliseconds; default is 3s.

Example

```
val listCAPublicKeyIDs = ListCAPublicKeyIDs(deviceId, it, timeoutMilli
)
val status = listCAPublicKeyIDs.waitForCompletionWithTimeout(20000)
```

7. Configurable AID and Group Commands

The following section describes AID and group configuration commands.

7.1. SetCtlsAppDataAsync

```
fun Client.SetCtlsAppDataAsync(deviceId: String, tlv: String,
timeoutMilli: Long = 3000): CtlsSetApplicationDataCommand
```

Set CTLS application data.

Parameters

Name	Type	Description
deviceId	String	Device ID.
tlv	String	The TLV to set.
timeoutMilli	Long	The timeout time in milliseconds; default is 3s.

Returns

```
UByte
```

Example

```
val cmd = Client.SetCtlsAppDataAsync(
    connectedDeviceId!!, "DFEE2D01B09F0607A0000001523010DFEE2E0110DFEE
4B0101"
)
val status: ClientCommandStatus = cmd.waitForCompletionWithTimeout(300
0)
```

7.2. GetCtlsAppDataAsync

```
fun Client.GetCtlsAppDataAsync(deviceId: String, aid: String,
timeoutMilli: Long = 3000): RetrieveCtlsApplicationDataCommand
```

Retrieve the reader's CTLS application data.

Parameters

Name	Type	Description
deviceId	String	Device ID.
aid	String	The AIDs in the configuration.
timeoutMilli	Long	The timeout time in milliseconds; default is 3s.

Returns

```
UByteArray
```

Example

```
val cmd = Client.GetCtlsAppDataAsync(connectedDeviceId!!, "A0000001523
010")
val status: ClientCommandStatus = cmd.waitForCompletionWithTimeout(200
00)
val result: UByteArray = cmd.getResultData()
```

7.3. RemoveCtlsAppDataAsync

```
fun Client.RemoveCtlsAppDataAsync(deviceId: String, aid: String,
timeoutMilli: Long = 3000): CtlsRemoveApplicationDataCommand
```

Remove the reader's CTLS application data.

Parameters

Name	Type	Description
deviceId	String	Device ID.
timeoutMilli	Long	The timeout time in milliseconds; default is 3s.

Returns

```
UByte
```

Example

```
val cmd = Client.RemoveCtlsAppDataAsync(connectedDeviceId!!, "A0000001
523010")
val status: ClientCommandStatus = cmd.waitForCompletionWithTimeout(200
00)
```

7.4. GetAllCtlsConfigurationGroupsAsync

```
fun Client.GetAllCtlsConfigurationGroupsAsync(deviceId: String,
timeoutMilli: Long = 3000): GetAllGroupsCommand
```

Get all the reader's CTLS configuration groups.

Parameters

Name	Type	Description
deviceId	String	Device ID.
timeoutMilli	Long	The timeout time in milliseconds; default is 3s.

Example

```
val cmd = Client.GetAllCtlsConfigurationGroupsAsync(connectedDeviceId!
!,20000)
val cmd_rv = cmd.waitForCompletionWithTimeout(20000)
```

7.5. GetCtlsAIDListAsync

```
fun Client.GetCtlsAIDListAsync(deviceId: String, timeoutMilli: Long =
3000): RetrieveCtlsAIDListCommand
```

Retrieve the reader's CTLS AID list.

Parameters

Name	Type	Description
deviceId	String	Device ID.
timeoutMilli	Long	The timeout time in milliseconds; default is 3s.

Returns

```
AIDListResponse(val aidList: List)
```

Example

```
val cmd = Client.GetCtlsAIDListAsync(connectedDeviceId!!)
val status: ClientCommandStatus = cmd.waitForCompletionWithTimeout(200
00)
val result: AIDListResponse = cmd.getResultData()
```

7.6. SetCtlsTerminalDataAsync

```
fun Client.SetCtlsTerminalDataAsync(deviceId: String,
terminalDataBytes: ByteArray, timeoutMilli: Long = 3000):
SetCtlsTerminalDataCommand
```

Set EMV terminal data.

Parameters

Name	Type	Description
deviceId	String	Device ID.
terminalDataBytes	ByteArray	The EMV terminal data to set.
timeoutMilli	Long	The timeout time in milliseconds; default is 3s.

Returns

```
EMVTerminalDataResult(var statusCode: UByte, var data: ByteArray)
```

Example

```
val cmd = Client.RemoveEMVTerminalDataAsync(connectedDeviceId!!)
cmd.waitForCompletionWithTimeout(20000)
```

7.7. GetCtlsTerminalDataAsync

```
fun Client.GetCtlsTerminalDataAsync(deviceId: String, timeoutMilli:
Long = 3000): GetCtlsTerminalDataCommand
```

Retrieve the reader's EMV terminal data.

Parameters

Name	Type	Description
deviceId	String	Device ID.
timeoutMilli	Long	The timeout time in milliseconds; default is 3s.

Returns

```
EMVTerminalDataResult(var statusCode: UByte, var data: ByteArray)
```

Example

```
val cmd = Client.RetrieveEMVTerminalDataAsync(connectedDeviceId!!)
cmd.waitForCompletionWithTimeout(20000)
val status: ClientCommandStatus = cmd.getCommandStatus()
val result: EMVTerminalDataResult? = cmd.getResultData()
```

7.8. GetCtlsConfigurationGroupAsync

```
fun Client.GetCtlsConfigurationGroupAsync(deviceId: String,
groupNumber: UShort, timeoutMilli: Long = 3000):
GetConfigurableGroupCommand
```

Get the reader's CTLS configuration group.

Parameters

Name	Type	Description
deviceId	String	Device ID.
groupNumber	UShort	The group number of the configuration.
timeoutMilli	Long	The timeout time in milliseconds; default is 3s.

Example

```
val cmd = Client.GetCtlsConfigurationGroupAsync(connectedDeviceId!!,0u
)
val cmd_rv = cmd.waitForCompletionWithTimeout(20000)
```

7.9. SetTLVConfigurationAsync

```
fun Client.SetTLVConfigurationAsync(deviceId: String, configuration:
ByteArray, timeoutMilli: Long = 3000): SetTLVConfigurationCommand
```

Set or change the values of specified Tag Length Value (TLV) data objects in the reader. This command can set parameters for Auto Poll and Poll on Demand Modes. When the reader receives this command, it extracts the TLV-encoded parameters from the data portion of the command and saves them to the default TLV Group in non-volatile memory. If a TLV data object is incorrectly formatted, the reader stops processing the object. A single command may contain more than one TLV data object.

Parameters

Name	Type	Description
deviceId	String	Device ID.
configuration	ByteArray	The TLV configuration to set.
timeoutMilli	Long	The timeout time in milliseconds; default is 3s.

Example

```
GlobalScope.launch(Dispatchers.IO) {
    Client.waitUntilConnected()
    val cmd = Client.SetTLVConfigurationAsync(connectedDeviceId!!,arra
yOf(0xDF.toByte(), 0xEE.toByte(), 0x38.toByte(), 0x01.toByte(), 0x00.t
oByte()))
    val status = cmd.waitForCompletionWithTimeout(10000)
    outputInCoroutine("SetTLVConfiguration: ${status.name}")
}
```

7.10. SetEMVApplicationDataAsync

```
fun Client.SetEMVApplicationDataAsync(deviceId: String, aid: String,
appdata: ByteArray, timeoutMilli: Long = 3000):
ContactSetApplicationDataCommand
```

Set EMV application data.

Parameters

Name	Type	Description
deviceId	String	Device ID.
aid	String	AID to set.
appdata	ByteArray	App data to set.
timeoutMilli	Long	The timeout time in milliseconds; default is 3s.

Returns

```
UByte
```

Example

```
val cmd = Client.SetEMVApplicationDataAsync(
    connectedDeviceId!!,
    "a00000002501",
    "5F5701005F2A0208409F090200015F3601029F1B0400003A98DF25039F3704DF2
8039F0802DFEE150101DF13050000000000DF14050000000000DF15050000000000DF1
80100DF170400002710DF190100".decodeHex()
)
val status: ClientCommandStatus = cmd.waitForCompletionWithTimeout(300
0)
```

7.11. RetrieveEmvAIDListAsync

```
fun Client.RetrieveEmvAIDListAsync(deviceId: String, timeoutMilli:
Long = 3000): ContactRetrieveAIDListCommand
```

Retrieve the reader's EMV AID list.

Parameters

Name	Type	Description
deviceId	String	Device ID.
timeoutMilli	Long	The timeout time in milliseconds; default is 3s.

Returns

```
EMVAIDListResult(var statusCode: UByte,var list: Array)
```

Example

```
val cmd = Client.RetrieveEmvAIDListAsync(connectedDeviceId!!)
val status: ClientCommandStatus = cmd.waitForCompletionWithTimeout(200
00)
val result: EMVAIDListResult = cmd.getResultData()
```

7.12. RetrieveEMVApplicationDataAsync

```
fun Client.RetrieveEMVApplicationDataAsync(deviceId: String, aid:
String, timeoutMilli: Long = 3000):
ContactRetrieveApplicationDataCommand
```

Retrieve the reader's EMV application data.

Parameters

Name	Type	Description
deviceId	String	Device ID.
aid	String	The application data to receive.
timeoutMilli	Long	The timeout time in milliseconds; default is 3s.

Returns

```
EMVAppDataResult(var statusCode: UByte,var data: ByteArray)
```

Example

```
val cmd = Client.RetrieveEMVApplicationDataAsync(connectedDeviceId!!,
"a00000002501")
val status: ClientCommandStatus = cmd.waitForCompletionWithTimeout(200
00)
val result: EMVAppDataResult = cmd.getResultData()
```

7.13. RemoveEMVApplicationDataAsync

```
fun Client.RemoveEMVApplicationDataAsync(deviceId: String, aid:
String, timeoutMilli: Long = 3000):
ContactRemoveApplicationDataCommand
```

Remove the reader's EMV application data.

Parameters

Name	Type	Description
deviceId	String	Device ID.
aid	String	The AID to remove.
timeoutMilli	Long	The timeout time in milliseconds; default is 3s.

Returns

```
UByte
```

Example

```
val cmd = Client.RemoveEMVApplicationDataAsync(connectedDeviceId!!, "a
00000002501")
val status: ClientCommandStatus = cmd.waitForCompletionWithTimeout(200
00)
```

7.14. SetEMVTerminalMajorConfigurationAsync

```
fun Client.SetEMVTerminalMajorConfigurationAsync(deviceId: String, id:
Int, timeoutMilli: Long = 3000): ContactSetICSIdentificationCommand
```

Set EMV terminal major configuration.

Parameters

Name	Type	Description
deviceId	String	Device ID.
id	Int	The terminal configuration to set.
timeoutMilli	Long	The timeout time in milliseconds; default is 3s.

Returns

```
UByte
```

Example

```
val cmd = Client.SetEMVTerminalMajorConfigurationAsync(connectedDevice
Id!!, 6)
val status: ClientCommandStatus = cmd.waitForCompletionWithTimeout(300
0)
```

7.15. SetEMVTerminalDataAsync

```
fun Client.SetEMVTerminalDataAsync(deviceId: String,
terminalDataBytes: ByteArray, timeoutMilli: Long = 3000):
ContactSetTerminalDataCommand
```

Set EMV terminal data.

Parameters

Name	Type	Description
deviceId	String	Device ID.
terminalDataBytes	ByteArray	The EMV terminal data to set.
timeoutMilli	Long	The timeout time in milliseconds; default is 3s.

Returns

```
UByte
```

Example

```
val data6C = "9f330360f8c89f3501229f4005f000f0a001df110101df260101df27
0100dfee1e08f09c3cf0c29e96005f3601029f1a0208409f1e085465726d696e616c9f
150212349f160f303030303030303030303030303030309f1c0838373635343332319f4e
22313037323120576 16c6b65722053742e20437970726573732c204341202c5553412e
df1002656edfee150101dfee160100dfee170105dfee180180dfee1f0180dfee1b0830
30ff3031ff3531dfee20013cdfee21010adfee2203323c3c".decodeHex()
val cmd = Client.SetEMVTerminalDataAsync(connectedDeviceId!!, data6C)
val status: ClientCommandStatus = cmd.waitForCompletionWithTimeout(300
0)
```

7.16. RetrieveEMVTerminalDataAsync

```
fun Client.RetrieveEMVTerminalDataAsync(deviceId: String,
timeoutMilli: Long = 3000): ContactRetrieveTerminalDataCommand
```

Retrieve the reader's EMV terminal data.

Parameters

Name	Type	Description
deviceId	String	Device ID.
timeoutMilli	Long	The timeout time in milliseconds; default is 3s.

Returns

```
EMVTerminalDataResult(var statusCode: UByte, var data: ByteArray)
```

Example

```
val cmd = Client.RetrieveEMVTerminalDataAsync(connectedDeviceId!!)
cmd.waitForCompletionWithTimeout(20000)
val status: ClientCommandStatus = cmd.getCommandStatus()
val result: EMVTerminalDataResult? = cmd.getResultData()
```

7.17. RemoveEMVTerminalDataAsync

```
fun Client.RemoveEMVTerminalDataAsync(deviceId: String, timeoutMilli:
Long = 3000): ContactRemoveTerminalDataCommand
```

Remove the reader's EMV terminal data.

Parameters

Name	Type	Description
deviceId	String	Device ID.
timeoutMilli	Long	The timeout time in milliseconds; default is 3s.

Returns

```
EMVTerminalDataResult(var statusCode: UByte, var data: ByteArray)
```

Example

```
val cmd = Client.RemoveEMVTerminalDataAsync(connectedDeviceId!!)
cmd.waitForCompletionWithTimeout(20000)
```

8. Module Versioning Commands

The following section describes module versioning commands.

8.1. GetFirmwareVersionAsync

```
fun Client.GetFirmwareVersionAsync(deviceId: String, timeoutMilli:
Long = 3000): GetDeviceFirmwareVersionCommand
```

Get the reader's firmware version.

Parameters

Name	Type	Description
deviceId	String	Device ID.
timeoutMilli	Long	The timeout time in milliseconds; default is 3s.

Example

```
val getFwVerCmd = Client.GetFirmwareVersionAsync(connectedDeviceId!!)
val status: ClientCommandStatus = getFwVerCmd.waitForCompletionWithTim
eout(20000)
val firmwareVersionData: String? = getFwVerCmd.getResultData()
```

8.2. GetHardwareInformationAsync

```
fun Client.GetHardwareInformationAsync(deviceId: String, timeoutMilli:
Long = 3000): GetHardwareInformationCommand
```

Get the reader's hardware information.

Parameters

Name	Type	Description
deviceId	String	Device ID.
timeoutMilli	Long	The timeout time in milliseconds; default is 3s.

Example

```
GlobalScope.launch(Dispatchers.IO) {
    Client.waitUntilConnected()
    val cmd = Client.GetHardwareInformationAsync(connectedDeviceId!!)
    val status = cmd.waitForCompletionWithTimeout(20000)
    val result: String = cmd.getResultData()?: ""
    outputInCoroutine("GetHardwareInformation: ${status.name}; Hardwar
eInformation: $result")
}
```

8.3. GetModuleVersionsAsync

```
fun Client.GetModuleVersionsAsync(deviceId: String, timeoutMilli: Long
= 3000): GetDeviceModuleVersionsCommand
```

Get the reader's module version.

Parameters

Name	Type	Description
deviceId	String	Device ID.
timeoutMilli	Long	The timeout time in milliseconds; default is 3s.

Example

```
GlobalScope.launch(Dispatchers.IO) {
    Client.waitUntilConnected()
    val cmd = Client.GetModuleVersionsAsync(connectedDeviceId!!)
    val status = cmd.waitForCompletionWithTimeout(20000)
    val result = cmd.getResultData()
    outputInCoroutine("GetModuleVersions ${status.name}; " +
        "ModuleVersions: $result")
}
```

8.4. GetDeviceVIDPIDAsync

```
fun Client.GetDeviceVIDPIDAsync(deviceId: String, timeoutMilli: Long =
3000): GetDeviceVIDPIDCommand
```

Get the reader's VID and PID.

Parameters

Name	Type	Description
deviceId	String	Device ID.
timeoutMilli	Long	The timeout time in milliseconds; default is 3s.

Example

```
GlobalScope.launch(Dispatchers.IO) {
    Client.waitUntilConnected()
    val cmd = Client.GetDeviceVIDPIDAsync(connectedDeviceId!!)
    val status = cmd.waitForCompletionWithTimeout(20000)
    val result = cmd.getResultData()
    outputInCoroutine("GetDeviceVIDPID: ${status.name}; " +
        "VID: ${result?.VID}; PID: ${result?.PID}")
}
```

8.5. GetContactCardModuleInformationAsync

```
fun Client.GetContactCardModuleInformationAsync(deviceId: String,
timeoutMilli: Long = 3000): GetContactCardModuleInformationCommand
```

Get the reader's contact card module information.

Parameters

Name	Type	Description
deviceId	String	Device ID.
timeoutMilli	Long	The timeout time in milliseconds; default is 3s.

Example

```
GlobalScope.launch(Dispatchers.IO) {
    Client.waitUntilConnected()
    val cmd = Client.GetContactCardModuleInformationAsync(connectedDev
iceId!!)
    val status = cmd.waitForCompletionWithTimeout(20000)
    val result = cmd.getResultData()
    outputInCoroutine("GetContactCardModuleInformation ${status.name};
 " +
        "ifmID: ${result?.ifmId}; " +
        "ifmHardwareId: ${result?.ifmHardwareId}; " +
        "ifmSoftwareId: ${result?.ifmSoftwareId}")
}
```

9. Real-Time Clock (RTC) Commands

The following section describes real time clock commands.

9.1. GetDeviceTimeAsync

```
fun Client.GetDeviceTimeAsync(deviceId: String, timeoutMilli: Long =
3000): GetDeviceTimeCommand
```

Get the reader's device time.

Parameters

Name	Type	Description
deviceId	String	Device ID.
timeoutMilli	Long	The timeout time in milliseconds; default is 3s.

Returns

```
Array
```

Example

```
val getdtcmd = Client.GetDeviceTimeAsync(connectedDeviceId!!)
val status: ClientCommandStatus = getdtcmd.waitForCompletionWithTimeou
t(20000)
val dt: Array<Int>? = getdtcmd.getResultData()
if (dt != null && dt.size == 6) {
    val result = "${"%02x".format(dt[0])}-" +
                 "${"%02x".format(dt[1])}-" +
                 "${"%02x".format(dt[2])}T" +
                 "${"%02x".format(dt[3])}:" +
                 "${"%02x".format(dt[4])}:" +
                 "${"%02x".format(dt[5])}")
}
```

9.2. SetRebootTime

```
suspend fun Client.SetRebootTime(deviceId: String, hh: Int, mm: Int,
timeoutMilli: Long = 3000): ClientCommandStatus
```

Set reboot time to 00:00 in UTC.

Parameters

Name	Type	Description
deviceId	String	Device ID.
hh	Int	The hour to set.
mm	Int	The minutes to set.
timeoutMilli	Long	The timeout time in milliseconds; default is 3s.

Returns

```
Status
```

Example

```
GlobalScope.launch(Dispatchers.IO) {
     val hour = 0
     val minute = 0
     val status Client.SetRebootTime(deviceId, hour, minute)
     Log.d(TAG, "SetRebootTime: status: $status")
}
```

9.3. GetRebootTime

```
suspend fun Client.GetRebootTime(deviceId: String, timeoutMilli: Long
= 3000): UByteArray?
```

Get the reader's reboot time.

Parameters

Name	Type	Description
deviceId	String	Device ID.
timeoutMilli	Long	The timeout time in milliseconds; default is 3s.

Returns

The UByte array of reboot time. The size should be two bytes. The first is hours, second is minutes.

Example

```
GlobalScope.launch(Dispatchers.IO) {
     val rebootTime = Client.GetRebootTime(deviceId)
     if (rebootTime == null) {
         Log.d(TAG, "getK81RebootTime: Can't get reboot time.")
     } else {
         val hh = rebootTime[0]
         val mm = rebootTime[1]
         Log.d(TAG, "getK81RebootTime: reboot at [${"%02x".format(hh.t
oInt())}:${"%02x".format(mm.toInt())}] in UTC.")
     }
}
```

9.4. GetNextRebootTime

```
fun Client.GetNextRebootTime(): String?
```

Get the reader's next reboot time.

Parameters

None.

Returns

Reboot time.

Example

```
val result: String = Client.GetNextRebootTime()
```

9.5. GetUpTimeSeconds

```
suspend fun Client.GetUpTimeSeconds(deviceId: String, timeoutMilli:
Long = 3000): Int?
GetUpTimeSeconds
```

Get the reader's uptime in seconds.

Parameters

Name	Type	Description
deviceId	String	Device ID.
timeoutMilli	Long	The timeout time in milliseconds; default is 3s.

Returns

The integer of uptime. Time unit is second.

Example

```
GlobalScope.launch(Dispatchers.IO) {
     val uptimeSeconds = Client.GetUpTimeSeconds(deviceId)
     Log.d(TAG, "GetUpTimeSeconds: $uptimeSeconds")
}
```

9.6. Set24HRSelfCheckTimeAsync

```
fun Client.Set24HRSelfCheckTimeAsync(deviceId: String, hours: UByte,
minutes: UByte, timeoutMilli: Long = 3000):
Set24HRSelfCheckTimeCommand
```

Set the reader's 24-hour self-check time.

Parameters

Name	Type	Description
deviceId	String	Device ID.
hours	UByte	Self-check time hour.
minutes	UByte	Self-check time minutes.
timeoutMilli	Long	The timeout time in milliseconds; default is 3s.

9.7. Get24HRSelfCheckTimeAsync

```
fun Client.Get24HRSelfCheckTimeAsync(deviceId: String, timeoutMilli:
Long = 3000): Get24HRSelfCheckTimeCommand
```

Get the reader's 24-hour self-check time.

Parameters

Name	Type	Description
deviceId	String	Device ID.
timeoutMilli	Long	The timeout time in milliseconds; default is 3s.

10. PINpad Commands

The following section describes PINpad and related commands.

10.1. CapturePINExtAsync

```
fun Client.CapturePINExtAsync(deviceId: String, data:
DisplayMessageAndGetEncryptedPINData, timeoutMilli: Long = 3000):
DisplayMessageAndGetEncryptedPINCommand
```

Display a message and get the encrypted PIN.

Parameters

Name	Type	Description
deviceId	String	Device ID.
data	DisplayMessageAndGetEncryptedPINData	The PIN capture.
timeoutMilli	Long	The timeout time in milliseconds; default is 3s.

Returns

```
PINCaptureResponseData
```

Example

```
GlobalScope.launch(Dispatchers.IO) {
    Client.waitUntilConnected()
    val data = DisplayMessageAndGetEncryptedPINData(
                    40u,
                    DisplayMessageAndGetEncryptedPINAndKeyType.DUKPTtoE
ncriptPINexternalPlaintextPAN,
                    "4111111111111111",
                    4u,
                    4u,
                    "ID TECH",
                    "PLEASE ENTER PIN",
                    "",
                    ""
                )
    val cmd = Client.CapturePINExtAsync(connectedDeviceId!!, data)
    val status = cmd.waitForCompletionWithTimeout(5000)
    val result: PINCaptureResponseData = cmd.getResultData()
}
```

Definition of **DisplayMessageAndGetEncryptedPINData** class:

```
class DisplayMessageAndGetEncryptedPINData(
    val timeout: UByte,
    val pinAndKeyType: UByte,
    val pan: String,
    val pinMinLen: UByte,
    val pinMaxLen: UByte,
    val promptMessageLine1: String, // 1st LCD Message
    val promptMessageLine2: String, // 1st LCD Message
    val verifyMessageLine1: String, // 2nd LCD Messageval verifyMessag
eLine2: String // 2nd LCD Message
)
```

10.2. GetFunctionKeyAsync

```
fun Client.GetFunctionKeyAsync(deviceId: String, timeoutMilli: Long =
3000): GetFunctionKeyCommand
```

Capture a single keypress value from a reader.

Parameters

Name	Type	Description
deviceId	String	Device ID.
timeoutMilli	Long	The timeout time in milliseconds; default is 3s.

Example

```
GlobalScope.launch(Dispatchers.IO) { Client.waitUntilConnected() val
cmd = Client.GetFunctionKeyAsync(connectedDeviceId!!) val status =
cmd.waitForCompletionWithTimeout(15000) val result =
cmd.getResultData() Log.d(TAG, "GetFunctionKeyAsync state:
${result?.displayMessageAndGetEncryptedPINAndKeyType}") Log.d(TAG,
"GetFunctionKeyAsync functionKey: ${result?.functionKey}") }
```

Returns

```
FunctionKeyResponseData
```

Definition of **FunctionKeyResponseData** class:

```
class FunctionKeyResponseData(
    val displayMessageAndGetEncryptedPINAndKeyType: UByte,
    val functionKey: UByteArray
)
```

10.3. InputFromPromptAsync

```
fun Client.InputFromPromptAsync(deviceId: String, data:
DisplaySpecificMessageAndGetNumericKeyData, timeoutMilli: Long =
3000): DisplaySpecificMessageAndGetNumericKeyCommand
```

Display a specific message and get the numeric key.

Parameters

Name	Type	Description
deviceId	String	Device ID.
data	DisplaySpecificMessageAndGetNumericKeyData	Key data input at the prompt.
timeoutMilli	Long	The timeout time in milliseconds; default is 3s.

Returns

```
DisplayMessageAndGetNumericKeyResult
```

Example

```
GlobalScope.launch(Dispatchers.IO) {
    Client.waitUntilConnected()
    val data = DisplaySpecificMessageAndGetNumericKeyData(
                    DisplaySpecificMessageAndGetNumericKeyDisplayFlag.d
isplayNumericForNumericKeyOnLCD,
                    4u,
                    4u,
                    15u,
                    1u,
                    DisplaySpecificMessageAndGetNumericKeyDisplayLangua
ge.English,
                    DisplaySpecificMessageAndGetNumericKeyScenarioID.en
terIPAddress,
                    "",
                    40u
                )
    val cmd = Client.InputFromPromptAsync(connectedDeviceId!!, data)
    val status = cmd.waitForCompletionWithTimeout(5000)
    val result: DisplayMessageAndGetNumericKeyResult = cmd.getResultDa
ta()
}
```

Definition of **DisplaySpecificMessageAndGetNumericKeyData** class:

```
class DisplaySpecificMessageAndGetNumericKeyData(
    val displayFlag: UByte,
    val preCleartextDisplay: UByte,
    val postCleartextDisplay: UByte,
    val keyMaxLen: UByte,
    val keyMinLen: UByte,
    val displayLanguage: UByte,
    val scenarioID: UByte,
    val defaultString: String,
    val timeout: UShort
)
```

10.4. CancelPINEntryAsync

```
fun Client.CancelPINEntryAsync(deviceId: String, timeoutMilli: Long =
3000): CancelPINEntryCommand
```

Cancel PIN entry.

Parameters

Name	Type	Description
deviceId	String	Device ID.
timeoutMilli	Long	The timeout time in milliseconds; default is 3s.

Returns

```
Status code
```

Example

```
GlobalScope.launch(Dispatchers.IO) {
    Client.waitUntilConnected()
    val cmd = Client.CancelPINEntryAsync(connectedDeviceId!!)
    val status = cmd.waitForCompletionWithTimeout(5000)
    val statusCode: UByte = cmd.getResultData()
}
```

10.5. PromptForAmountAsync

```
fun Client.PromptForAmountAsync(deviceId: String, data:
DisplayMessageAndGetAmountRequest, timeoutMilli: Long = 3000):
DisplayMessageAndGetAmountCommand
```

Capture a single keypress value from a reader.

Parameters

Name	Type	Description
deviceId	String	Device ID.
data	DisplayMessageAndGetResult	Data of the request.
timeoutMilli	Long	The timeout time in milliseconds; default is 3s.

Returns

```
DisplayMessageAndGetAmountResult
```

Definition of **DisplayMessageAndGetAmountResult** class:

```
class DisplayMessageAndGetAmountResult( val state: UByte, val
amountValue: UByteArray )
```

Example

```
GlobalScope.launch(Dispatchers.IO) {
    Client.waitUntilConnected()
    val data = DisplayMessageAndGetAmountRequest(
        40u,
        DisplayMessageAndGetNumericKeyFunctionFlag
            .displayStarSignforNumericKeyOnLCD,
        16u, 1u, "Enter Amt.",
        "7d23bc58d7d9555d33ebf4cd8e7a19f867de213a82bc44dc94ac77b58b10e
97a240f37480035bf7b08986c46f023e584f674dc60a2fe5ff6b0ee630cb4891c81d10
a33edc5dc5cf3c7a409726fa96aeddfe63c4be87464a28bbf48a90fc5497ce7b0629d2
717ccbb44c9c8a46b736e483d015679b060d7b1dea38e5966b50ab07100e736a12cb39
2acf32427b9607cc3d46ebc35464a6d27b7d7864a4d80fb93350c5b3b793e912aaca9f
5dbbec8a8d571ea10654f94d6d19d0f4a54dd1da6ae71195785d495c67aabd3bc05a45
c83da547fd74c46bbc4011b1c8af8b44c4bb185108465a08a31ab8ac69d47b09ca147d
d1427651497f6b19cfa9bdb41e99ec0".hexToUByteArray()
    )
    val cmd = Client.PromptForAmountAsync(connectedDeviceId!!, data)
    val status = cmd.waitForCompletionWithTimeout(15000)
    val result = cmd.getResultData()
    outputInCoroutine("PromptForAmount: ${status.name}; " +
        "state: ${result?.state}; amountValue: ${result?.amountValue}"
)
}
```

10.6. PromptForNumericKeyAsync

```
fun Client.PromptForNumericKeyAsync(deviceId: String, data:
DisplayMessageAndGetNumericKeyData, timeoutMilli: Long = 3000):
DisplayMessageAndGetNumericKeyCommand
```

Display a prompt message and numeric key.

Parameters

Name	Type	Description
deviceId	String	Device ID.
data	DisplayMessageAndGetNumericKeyData	The display message and numeric key to display.
timeoutMilli	Long	The timeout time in milliseconds; default is 3s.

Returns

```
DisplayMessageAndGetNumericKeyResult
```

Example

```
GlobalScope.launch(Dispatchers.IO) {
    Client.waitUntilConnected()
    val data = DisplayMessageAndGetNumericKeyData(
                    DisplayMessageAndGetNumericKeyFunctionFlag.displayS
tarSignforNumericKeyOnLCD,
                    16u,
                    1u,
                    "Enter Num. Key",
                    "67E889DC42150BD5E57BF221AAEB68707440FA1612AC78D64E
5F04DB081E6ADE1C67B65DA52E0F1B51B464057AE8B2274980A57C4858C51BE371569B
DD2F6D5B7FDB34FA2FE577E5E5CDB3CB74F3C5D70F3C12EA7AF7A78C6CD7A5D8861B32
5958E35F3C8F90D013F854A000A90FD83FABA0F227ACB14783DE8C71CDF5DAD120F17B
2771B51BAEE989DC444613A5F9FB88A255CA8ADE69DBF1EF78DC51A3044687B07ABE15
CB07154AFFADBCC178CF995A6DBD3D1AD07ED93EAFF7B23F95C582857344D8520CD1FE
EA2B73A8CC8141D0CE78B643767014D26559C55D8AC45A110C3A6BAB8E817E16A4FF52
325E9D3E69B6E686A7C188DBB673E3B3CC2AD91D3E".hexToUByteArray(),
                    40u
                )
    val cmd = Client.PromptForNumericKeyAsync(connectedDeviceId!!, dat
a)
    val status = cmd.waitForCompletionWithTimeout(5000)
    val result: DisplayMessageAndGetNumericKeyResult = cmd.getResultDa
ta()
}
Definition of DisplayMessageAndGetNumericKeyData:
class DisplayMessageAndGetNumericKeyData(
    val functionFlag: UByte,
    val keyMaxLen: UByte,
```

```
    val keyMinLen: UByte,
    val displayMessage: String,
    val displayMessageSignature: UByteArray,
    val timeout: UShort
)
```

10.7. PromptForNumericKeyWithSwipeAsync

```
fun Client.PromptForNumericKeyWithSwipeAsync(deviceId: String, data:
DisplayMultiLineMessageAndGetNumericKeyData, timeoutMilli: Long =
3000): DisplayMultiLineMessageAndGetNumericKeyCommand
```

Display a multi-line message and numeric key.

Parameters

Name	Type	Description
deviceId	String	Device ID.
data	DisplayMultiLineMessageAndGetNumericKeyData	The display message and numeric key to display.
timeoutMilli	Long	The timeout time in milliseconds; default is 3s.

Returns

```
DisplayMultiLineMessageAndGetNumericKeyResult
```

Example

```
GlobalScope.launch(Dispatchers.IO) {
    Client.waitUntilConnected()
    val data = DisplayMultiLineMessageAndGetNumericKeyData(
                    DisplayMultiLineMessageAndGetNumericKeyFunctionFlag
.enableMSRdisplayNumericFirstAndThenMaskedByStarSignOnLCD,
                    16u,
                    1u,
                    "Please Swipe Or",
                    "Key Employee ID",
                    "b6d80e9d7d5686c7010215e0bacf95dc7f1f5a38d806ac9fb8
1bfb2368fe2f8a036a4f5ac85e81c5d0313b6e5472cfed9b6e8fa02b95e2427693e0d0
d271e9a3b1134f3f4c363e906e45f03a52937c59375ecb17b75565da9298639bf501ef
be884618af13a10da8eadb73cdb74406d15c224fbc8e685b1a1c672664f1a8e089ad2c
e67c69e84b5b6b5b94774105e871ef1e2a153591b19fbd9b3e5c9d45a44740339b09e2
50fd362030a726dd9a41784ed108e5b96c1b91164439890c7f2616293c97308f0eb7e4
3b2ce2ac8e09febc1a5b81a9568722b19a3b703249734298244bc5b47d32750956e567
27e45a8bfa9c17ca7803245f8c8ee361864fb9e89e".hexToUByteArray(),
                    40u
                )
     val cmd = Client.PromptForNumericKeyWithSwipeAsync(connectedDevic
eId!!, data)
     val status = cmd.waitForCompletionWithTimeout(5000)
```

```
     val result: DisplayMultiLineMessageAndGetNumericKeyResult = cmd.g
etResultData()
}
```

Definition of **DisplayMultiLineMessageAndGetNumericKeyData** class:

```
class DisplayMultiLineMessageAndGetNumericKeyData(
    val functionFlag: UByte,
    val keyMaxLen: UByte,
    val keyMinLen: UByte,
    val displayMessageLine1: String,
    val displayMessageLine2: String,
    val displayMessageSignature: UByteArray,
    val timeout: UShort
)
```

10.8. GetPANEntryAsync

```
fun Client.GetPANEntryAsync(deviceId: String, data: GetPANData,
timeoutMilli: Long = 3000): GetPANCommand
```

Get PAN

Parameters

Name	Type	Description
deviceId	String	Device ID.
data	GetPANData	The PAN data.
timeoutMilli	Long	The timeout time in milliseconds; default is 3s.

Returns

```
GetPANResult
```

Example

```
GlobalScope.launch(Dispatchers.IO) {
    Client.waitUntilConnected()
    val data = GetPANData(
                    false, false, false,
                    false, false, true, false,
                    15u
                )
    val cmd = Client.GetPANEntryAsync(connectedDeviceId!!, data)
    val status = cmd.waitForCompletionWithTimeout(5000)
    val result: GetPANResult = cmd.getResultData()
}
```

Definition of **GetPANData** class:

```
class GetPANData(
    val requestMod10Check: Boolean,
    val requestZIP: Boolean,
    val requestADR: Boolean,
    val requestExpData: Boolean,
    val requestCSC: Boolean,
    val PANOnly: Boolean,
    val withKeySlotSelection: Boolean,
    val timeout: UShort
    /* legacy C SDK does not consider DEK KeySlot */
)
```

11. Passthrough Commands

The section below describes passthrough commands.

11.1. EnableL100PassThroughAsync

```
fun Client.EnableL100PassThroughAsync(deviceId: String, enabled:
Boolean, timeoutMilli: Long = 3000): SetPinPadPassThroughModeCommand
```

Enable L100 Pass-Through Mode.

Parameters

Name	Type	Description
deviceId	String	Device ID.
enabled	boolean	If true, L100 Passthrough Mode is enabled.
timeoutMilli	Long	The timeout time in milliseconds; default is 3s.

Example

```
if (connectedDeviceId == null) {
      outputInCoroutine("Not connected to any device")
   } else {
     GlobalScope.launch(Dispatchers.IO) {
         Client.waitUntilConnected()
         val cmd = Client.EnableL100PassThroughAsync(connectedDeviceId
!!,true)
         val status = cmd.waitForCompletionWithTimeout(3000)
         outputInCoroutine("EnableL100PassThrough status: ${status.nam
e}")
    }
}
```

11.2. GetL100PassThroughModeAsync

```
fun Client.GetL100PassThroughModeAsync(deviceId: String, timeoutMilli:
Long = 3000): RetrievePinPadPassThroughModeCommand
```

Enable L100 Pass-Through Mode.

Parameters

Name	Type	Description
deviceId	String	Device ID.
timeoutMilli	Long	The timeout time in milliseconds; default is 3s.

Example

```
if (connectedDeviceId == null) {
      outputInCoroutine("Not connected to any device")
   } else {
     GlobalScope.launch(Dispatchers.IO) {
         Client.waitUntilConnected()
         val cmd = Client.GetL100PassThroughModeAsync(connectedDeviceI
d!!)
         val status = cmd.waitForCompletionWithTimeout(3000)
         outputInCoroutine("GetL100PassThroughMode status: ${status.na
me}, enable: ${cmd.getResultData()}")
    }
}
```

11.3. AndroidRespondToK81Async

```
fun Client.AndroidRespondToK81Async(deviceId: String, cmd: UByte,
subCmd: UByte, data: UByteArray?, timeoutMilli: Long = 3000):
MiscSendDataCommandNEOCommand
```

Sends an Android response to the reader's K81 processor.

Parameters

Name	Type	Description
deviceId	String	Device ID.
cmd	UByte	Subcommand.
timeoutMilli	Long	The timeout time in milliseconds; default is 3s.

Returns

```
DataResponseNEO(val responseFrames: List) class ResponseFrame(val
statusCode: UByte, val data: UByteArray)
```

Example

```
val DISP_LANG_MENU = "08".hexToByteArray()
val displayContent = 1
val displayContentLength = displayContent.size.toHexWithLsbAndMsb().he
xToByteArray()
val DisplayLangMenuResCommand = DISP_LANG_MENU + displayContentLength
+ displayContent
Client.AndroidRespondToK81Async(
    connectedDeviceId!!,
    0x61u,
    0x01u,
    DisplayLangMenuResCommand.toUByteArray()
)
```

11.4. EnableAutoPollModeAsync

```
fun Client.EnableAutoPollModeAsync(deviceId: String, enable: Boolean,
timeoutMilli: Long = 3000): EnableAutoPollModeCommand
```

Enable Auto Poll mode.

Parameters

Name	Type	Description
deviceId	String	Device ID.
timeoutMilli	Long	The timeout time in milliseconds; default is 3s.
enable	Boolean	If true, auto poll mode is enabled.

Returns

```
Status
```

Example

```
GlobalScope.launch(Dispatchers.IO) {
    Client.waitUntilConnected()
    val cmd = Client.EnableAutoPollModeAsync(connectedDeviceId!!)
    cmd.waitForCompletion()
    val result = cmd.getResultData()
    outputInCoroutine("EnableAutoPollMode(false) statusCode: : $result
")
}
```

11.5. EnablePassThroughModeAsync

```
fun Client.EnablePassThroughModeAsync(deviceId: String, enable:
Boolean, timeoutMilli: Long = 3000): EnablePassThroughModeCommand
```

Enable Pass-Through mode.

Parameters

Name	Type	Description
deviceId	String	Device ID.
Enable	Boolean	If true, passthrough mode is enabled.
timeoutMilli	Long	The timeout time in milliseconds; default is 3s.

Returns

```
Status
```

Example

```
GlobalScope.launch(Dispatchers.IO) {
    Client.waitUntilConnected()
    val cmd = Client.EnablePassThroughModeAsync(connectedDeviceId!!,fa
lse)
    cmd.waitForCompletion()
    val result = cmd.getResultData()
    outputInCoroutine("EnablePassThroughModeAsync(false) statusCode: :
 $result")
}
```

12. FeliCa Commands

The section below describes API methods related to FeliCa cards.

12.1. FelicaAuthenticateAsync

```
fun Client.FelicaAuthenticateAsync(deviceId: String, array: ByteArray,
timeoutMilli: Long = 3000): FelicaAuthenticateCommand
```

Authenticate a FeliCa card transaction.

Parameters

Name	Type	Description
deviceId	String	Device ID.
array	ByteArray	FeliCa authentication key.
timeoutMilli	Long	The timeout time in milliseconds; default is 3s.

Returns

```
Status
```

Example

```
val cmd = Client.FelicaAuthenticateAsync(connectedDeviceId!!,"00000000
000000000000000000000000".hexToByteArray())
val status = cmd.waitForCompletionWithTimeout(20000)
```

12.2. FelicaPollAsync

```
fun Client.FelicaPollAsync(deviceId: String, timeout: UShort,
timeoutMilli: Long = 3000): FelicaPollCommand
```

Poll for a FeliCa card.

Parameters

Name	Type	Description
deviceId	String	Device ID.
timeout	UShort	Timeout for autopolling.
timeoutMilli	Long	The timeout time in milliseconds; default is 3s.

Returns

```
Status
```

Example

```
val cmd = Client.FelicaPollAsync(connectedDeviceId!!, 30u,10000)
val cmd_rv = cmd.waitForCompletionWithTimeout(10000)
```

12.3. FelicaReadAsync

```
fun Client.FelicaReadAsync(deviceId: String, data: FelicaReadRequest,
timeoutMilli: Long = 3000): FelicaReadCommand
```

Read a FeliCa card.

Parameters

Name	Type	Description
deviceId	String	Device ID.
Data	FeliCaReadRequest	Read a FeliCa card.
timeoutMilli	Long	The timeout time in milliseconds; default is 3s.

Returns

```
Status
```

Example

```
val cmd = Client.FelicaReadAsync(connectedDeviceId!!,
                        FelicaReadRequest(blockCount = 1u,blockList =
"8000".hexToUByteArray(), serviceCodeList = "0b00".hexToUByteArray()),
 10000)
val cmd_rv = cmd.waitForCompletionWithTimeout(10000)
```

12.4. FelicaReadWithMacAsync

```
fun Client.FelicaReadWithMacAsync(deviceId: String, data:
FelicaReadWithMacRequest, timeoutMilli: Long = 3000):
FelicaReadWithMacCommand
```

FeliCa Read With Mac

Parameters

Name	Type	Description
deviceId	String	Device ID.
Data	FeliCaReadWithMacRequest	Read a FeliCa card with a MAC address.
timeoutMilli	Long	The timeout time in milliseconds; default is 3s.

Returns

```
Status
```

Example

```
val cmd = Client.FelicaReadWithMacAsync(connectedDeviceId!!,FelicaRead
WithMacRequest(blockCount = 1u,blockList = "8000".hexToUByteArray()),
10000)
val cmd_rv = cmd.waitForCompletionWithTimeout(10000)
```

12.5. FelicaWriteAsync

```
fun Client.FelicaWriteAsync(deviceId: String, data:
FelicaWriteRequest, timeoutMilli: Long = 3000): FelicaWriteCommand
```

Write to a FeliCa card.

Parameters

Name	Type	Description
deviceId	String	Device ID.
data	FelicaWriteRequest	Write to a FeliCa card.
timeoutMilli	Long	The timeout time in milliseconds; default is 3s.

Returns

```
Status
```

Example

```
val cmd = Client.FelicaWriteAsync(connectedDeviceId!!,
                        FelicaWriteRequest(blockCount = 1u,
                            blockList = "8000".hexToUByteArray(),
                            serviceCodeList = "0900".hexToUByteArray()
,
                            blockData = "00000000000000000000000000000
000".hexToUByteArray()), 10000)
val cmd_rv = cmd.waitForCompletionWithTimeout(10000)
```

12.6. FelicaWriteWithMacAsync

```
fun Client.FelicaWriteWithMacAsync(deviceId: String, data:
FelicaWriteWithMacRequest, timeoutMilli: Long = 3000):
FelicaWriteWithMacCommand
```

Write to a FeliCa card with MAC address.

Parameters

Name	Type	Description
deviceId	String	Device ID.
data	FelicaWriteWithMacRequest	Write to a FeliCa card with the MAC address.
timeoutMilli	Long	The timeout time in milliseconds; default is 3s.

Returns

```
Status
```

Example

```
val cmd = Client.FelicaWriteWithMacAsync(connectedDeviceId!!,FelicaWri
teWithMacRequest(blockNumber = 1u,blockData = "00000000000000000000000
000000000".hexToUByteArray()), 10000)
val cmd_rv = cmd.waitForCompletionWithTimeout(10000)
```

13. ICC Commands

The section below describes ICC commands.

13.1. ICCExchangeAPDUAsync

```
fun Client.ICCExchangeAPDUAsync(deviceId: String, apdu: ByteArray,
timeoutMilli: Long = 3000): ICCExchangeAPDUCommand
```

Exchange APDU data.

Parameters

Name	Type	Description
deviceId	String	Device ID.
apdu	ByteArray	APDU data.
timeoutMilli	Long	The timeout time in milliseconds; default is 3s.

Returns

APDU data.

Example

```
val apdu = "2000a404000e315041592e5359532e444446303100".hexToByteArray
()
val cmd = Client.ICCExchangeAPDUAsync(connectedDeviceId!!,apdu, 10000)
val cmd_rv = cmd.waitForCompletionWithTimeout(10000)
```

13.2. ICCGetReaderStatusAsync

```
fun Client.ICCGetReaderStatusAsync(deviceId: String, timeoutMilli:
Long = 3000): GetReaderStatusCommand
```

Get the reader's ready status.

Parameters

Name	Type	Description
deviceId	String	Device ID.
timeoutMilli	Long	The timeout time in milliseconds; default is 3s.

Returns

```
class GetICCReaderStatusResponseData(var isCardSeated:Boolean,var
isFrontSwitchDetected:Boolean,var isICCPowered:Boolean,var RFU:
UShort)
```

Example

```
val getReaderStatusTransCmd = Client.ICCGetReaderStatusAsync(connected
DeviceId!!, 10000)
val getReaderStatusTransCmd_rv = getReaderStatusTransCmd.waitForComple
tionWithTimeout(10000)
```

13.3. ICCPowerOffAsync

```
fun Client.ICCPowerOffAsync(deviceId: String, timeoutMilli: Long =
3000): ICCPowerOffCommand
```

Turn ICC power off.

Parameters

Name	Type	Description
deviceId	String	Device ID.
timeoutMilli	Long	The timeout time in milliseconds; default is 3s.

Example

```
val iccPowerOffCmd = Client.ICCPowerOffAsync(connectedDeviceId!!, 1000
0)
val iccPowerOffCmd_rv = iccPowerOffCmd.waitForCompletionWithTimeout(10
000)
```

13.4. ICCPowerOnAsync

```
fun Client.ICCPowerOnAsync(deviceId: String, timeoutMilli: Long =
3000): ICCPowerOnCommand
```

Turn ICC power on.

Parameters

Name	Type	Description
deviceId	String	Device ID.
timeoutMilli	Long	The timeout time in milliseconds; default is 3s.

Returns

```
Array
```

Example

```
val iccPowerOnCmd = Client.ICCPowerOnAsync(connectedDeviceId!!, 10000)
val iccPowerOnCmd_rv = iccPowerOnCmd.waitForCompletionWithTimeout(1000
0)
```