



iAuthenticate 2.0 SDK Manual

Version 1.0

Revision History

Version	Changed by	Description
1.0		Initial release corresponds to v1.0.0.0 SDK library

Table of Contents

1. Introduction.....	3
2. Supported External Accessory Protocols.....	3
3. Deviations from desktop PC/SC API.....	3
3.1 SCARD_AUTOALLOCATE.....	3
3.2 SCardFreeMemory	3
3.3 SCardGetStatusChange	3
3.4 SCardBeginTransaction & SCardEndTransaction	4
4. Version API	5
4.1 iAuth2GetLibVersion	5
5. Debug API	6
5.1 SetDebugLevel	6
5.2 IsDebugDataAvailable.....	7
5.3 RetrieveDebugData	8
6. Reader & Card Access API.....	9
6.1 SCardEstablishContext.....	9
6.2 SCardReleaseContext	10
6.3 SCardListReaders	11
6.4 SCardGetStatusChange	12
6.5 SCardConnect.....	13
6.6 SCardDisconnect	14
6.7 SCardReconnect	15
6.8 SCardTransmit.....	16
6.9 SCardStatus	17
6.10 SCardGetAttrib.....	18
6.11 SCardBeginTransaction	19
6.12 SCardEndTransaction	20
7. SDK Files.....	21
8. Sample Application.....	21
9. Checklist for Developers.....	23

1. Introduction

This document describes the iAuthenticate 2.0 SDK API, and provides some hints on how to use it effectively. This document is accurate as of version 1.0.0.0 of the SDK.

2. Supported External Accessory Protocols

The SDK supports the following External Accessory (EA) protocols:

`com.identiv.ccid-over-iap2`

`com.identiv.ccid-transport`

During development and testing, any one of the above protocol strings can be used depending on the device that is available. However it is recommend to use *com.identiv.ccid-transport* protocol during submission to App Store.

3. Deviations from desktop PC/SC API

The SDK is implemented as a PC/SC API around the proprietary driver supporting iAP2 protocol. This ensures that code that is proven to work on other platforms can be rapidly ported to iOS.

The PC/SC API supported by this SDK is a subset of the standard PC/SC. The implementation does not support the card database functionality defined by the PC/SC API. It supports only the transport API to communicate with reader and card. Some notable deviations from the desktop PC/SC API are listed below.

3.1 SCARD_AUTOALLOCATE

SCARD_AUTOALLOCATE is not supported by the SDK. Developers may have to make multiple calls to the respective API (once to get the size of buffer, and once again to get the data) to work around this limitation.

3.2 SCardFreeMemory

Since SCARD_AUTOALLOCATE is not supported, SCardFreeMemory() is also not supported.

3.3 SCardGetStatusChange

SCardGetStatusChange API polls the reader for card events every 500ms. If multiple card events happens within 500ms (like a card is quickly inserted and removed), it is possible that the event is not identified by the API.

The API does not power up the card. Hence ATR is not available in the SCARD_READERSTATE. Apps that need the ATR need to perform a SCardConnect followed by SCardStatus to retrieve the ATR. This implementation could change in future releases.

3.4 SCardBeginTransaction & SCardEndTransaction

Since the iOS app architecture does not allow shared libraries, there is no thread contention. Hence SCardBeginTransaction and SCardEndTransaction are just placeholder functions.

4. Version API

4.1 iAuth2GetLibVersion

Retrieves the library version.

Prototype:

```
LONG iAuth2GetLibVersionI(void);
```

Parameter(s):

None

Return Value:

Library version encoded as a LONG. It has 4 components, namely, major version, minor version, revision and sub-revision. Each component is one byte long.

Encoded as below:

(major version << 24) | (minor version << 16) | (revision << 8) | sub-revision

5. Debug API

The SDK implements a circular debug buffer where the debug messages are stored. App can retrieve them when needed. There is a provision for app to log its messages into the debug buffer and retrieve it too.

The debug buffer is statically allocated by the SDK library. The debug provision is not available in release version of library.

5.1 SetDebugLevel

Allows to set the granularity of debug messages from library.

Prototype:

```
void SetDebugLevel(int newLevel);
```

Parameter(s):

In/Out	Parameter	Description
[in]	newLevel	Granularity of debug message to take effect after the call

Return Value:

None

5.2 IsDebugEnabledDataAvailable

Allows to check for the presence of data in the circular debug buffer of SDK.

Prototype:

```
Int IsDebugEnabledDataAvailable(void);
```

Parameter(s):

None

Return Values:

Value	Description
True	if there is data in debug buffer
False	If there is no data in debug buffer

5.3 RetrieveDebugData

Allows to pull the data from the circular debug buffer of SDK library.

Prototype:

```
int RetrieveDebugData(char *buffer, unsigned size);
```

Parameter(s):

In/Out	Parameter	Description
[in]	buffer	Pointer to destination buffer where debug data is to be copied
[in]	size	Size of destination buffer

Return Values:

Number of bytes copied into destination buffer

Notes:

Debug messages copied/retrieved are removed from the queue.

6. Reader & Card Access API

6.1 SCardEstablishContext

Creates an Application Context to the SDK. This must be the first SDK function called from app.

Prototype:

```
LONG SCardEstablishContext(DWORD dwScope,  
                           LPCVOID pvReserved1,  
                           LPCVOID pvReserved2,  
                           LPSCARDCONTEXT phContext);
```

Parameter(s):

In/Out	Parameter	Description
[in]	Dwscope	Scope of context establishment; parameters is validated but not used. Allowed values are SCARD_SCOPE_USER SCARD_SCOPE_SYSTEM
[in]	pvReserved1	Not used – should be NULL
[in]	pvReserved2	Not used – should be NULL
[out]	phContext	Buffer to accept the application context

Return Values:

Value	Description
SCARD_S_SUCCESS	If context is successfully established
SCARD_E_INVALID_PARAMETER	If one of the parameters is not valid

Notes:

This API shall be successful even if accessory is not plugged in.

6.2 SCardReleaseContext

Destroys a communication context previously established with SDK. This must be the last function called into SDK library before app terminates.

Prototype:

```
LONG SCardReleaseContext(SCARDCONTEXT hContext);
```

Parameter(s):

In/Out	Parameter	Description
[in]	hContext	Context to be closed/released

Return Values:

Value	Description
SCARD_S_SUCCESS	If context is valid and is successfully released
SCARD_E_INVALID_HANDLE	If the context provided is not valid

6.3 SCardListReaders

Returns a list of available readers on the system.

pvReserved1 has to be NULL.

mszReaders is pointer to buffer that is allocated by the app. If the app sets this as NULL, then the function shall return the size of buffer needed to allocate in pcchReaders.

SCARD_AUTOALLOCATE is not supported for pcchReaders.

Prototype:

```
LONG SCardListReaders(SCARDCONTEXT hContext,  
                      LPCSTR pvReserved1,  
                      LPSTR mszReaders,  
                      LPDWORD pcchReaders);
```

Parameter(s):

In/Out	Parameter	Description
[in]	hContext	Context to be used for this call
[in]	pvReserved	Set to NULL
[out]	mszReaders	Pointer to buffer that shall contain reader names upon return
[in out]	pcchReaders	Input shall be size of buffer Output shall be the size of valid bytes in buffer

Return Values:

Value	Description
SCARD_S_SUCCESS	If readers are successfully retrieved and filled in buffer
SCARD_E_INVALID_HANDLE	If the context provided is not valid
SCARD_E_INSUFFICIENT_BUFFER	If the reader name buffer is insufficient to copy all reader names
SCARD_E_NO_READERS	If no readers are detected

6.4 SCardGetStatusChange

Blocks execution until the current availability of the cards in a specific set of readers changes.

This function receives a list of structures containing reader names. It then blocks waiting for a change in state to occur for a maximum blocking time of `dwTimeout`, or forever if `INFINITE` is used.

The new event state shall be contained in `dwEventState`. A status change shall be card insertion or card removal event.

To cancel the ongoing call, use `SCardCancel` with the same context.

Prototype:

```
LONG SCardGetStatusChange(SCARDCONTEXT hContext,  
                           DWORD dwTimeout,  
                           SCARD_READERSTATE rgReaderStates,  
                           DWORD cReaders);
```

Parameter(s):

In/Out	Parameter	Description
[in]	hContext	Context to be used for this call
[in]	dwTimeout	Maximum waiting time in milliseconds, or <code>INFINITE</code>
[in]	rgReaderStates	Array of structures containing reader names and state
[in]	cReaders	Count of structures in array

Return Values:

Value	Description
SCARD_S_SUCCESS	When there is event change within timeout specified
SCARD_E_INVALID_HANDLE	If the context provided is not valid
SCARD_E_TIMEOUT	When timeout has expired without event change
SCARD_E_INVALID_PARAMETER	If one of the parameters is not valid
SCARD_E_UNKNOWN_READER	If reader name provided in <code>SCARD_READSTATE</code> structure does not match any of the readers

6.5 SCardConnect

Establishes a connection/session to the card inserted into reader specified.

Prototype:

```
LONG SCardConnect( SCARDCONTEXT hContext,  
                   LPCSTR szReader,  
                   DWORD dwShareMode,  
                   DWORD dwPreferredProtocols,  
                   LPSCARDHANDLE phCard,  
                   LPDWORD pdwActiveProtocol);
```

Parameter(s):

In/Out	Parameter	Description
[in]	hContext	Context to be used for this call
[in]	szReader	Reader name to connect to
[in]	dwShareMode	Parameter is unused; connection is always shared
[in]	dwPreferredProtocols	Desired protocol to communicate with card
[out]	phCard	Handle to this card connection
[out]	pdwActiveProtocol	Protocol established with card

Return Values:

Value	Description
SCARD_S_SUCCESS	When connection/session is established with card
SCARD_E_INVALID_HANDLE	If the context provided is not valid
SCARD_E_INVALID_PARAMETER	If one of the parameters is not valid
SCARD_E_READER_UNAVAILABLE	If reader name provided does not match any of the identified readers
SCARD_E_NO_READERS_AVAILABLE	If the accessory was removed
SCARD_E_UNSUPPORTED_FEATURE	If SCARD_PROTOCOL_DIRECT is attempted

6.6 SCardDisconnect

Terminates a connection made through SCardConnect().

Prototype:

```
LONG SCardDisconnect(SCARDHANDLE hCard,  
                     DWORD dwDisposition);
```

Parameter(s):

In/Out	Parameter	Description
[in]	hCard	Connection returned by SCardConnect()
[in]	dwDisposition	Reader function to execute SCARD_LEAVE_CARD – do nothing SCARD_RESET_CARD – reset the card SCARD_UNPOWER_CARD – power down the card SCARD_EJECT_CARD – power down and eject the card

Return Values:

Value	Description
SCARD_S_SUCCESS	When connection/session was terminated successfully
SCARD_E_INVALID_HANDLE	If the card handle provided is not valid
SCARD_E_INVALID_PARAMETER	If dwDisposition parameter is not valid

6.7 SCardReconnect

Reestablishes a connection to a reader that was previously connected to using SCardConnect().

This can be used to switch the communication protocol previously established the card.

Prototype:

```
LONG SCardReConnect(SCARDHANDLE hCard,  
                    DWORD dwShareMode,  
                    DWORD dwPreferredProtocols,  
                    DWORD dwInitialization,  
                    LPDWORD pdwActiveProtocol);
```

Parameter(s):

In/Out	Parameter	Description
[in]	hCard	Handle from previous call to SCardConnect()
[in]	dwShareMode	Parameter is unused; connection is always shared
[in]	dwPreferredProtocols	Desired protocol to communicate with card
[in]	dwInitialization	Desired action on card/reader SCARD_LEAVE_CARD – do nothing SCARD_RESET_CARD – reset the card (warm reset) SCARD_UNPOWER_CARD – power down the card (cold reset) SCARD_EJECT_CARD – power down and eject the card
[out]	pdwActiveProtocol	Protocol established with card

Return Values:

Value	Description
SCARD_S_SUCCESS	When connection/session was terminated successfully
SCARD_E_INVALID_HANDLE	If the card handle provided is not valid

Notes:

This is just a placeholder and is not implemented as of v1.0.0.0 of library.

6.8 SCardTransmit

Sends an APDU to the smart card contained in the reader connected to by SCardConnect().

The card responses are stored in pbRecvBuffer and its length in pcbRecvLength.

Prototype:

```
LONG SCardTransmit( SCARDHANDLE hCard,  
                    SCARD_IO_REQUEST *pioSendPci,  
                    LPCBYTE pbSendBuffer,  
                    DWORD cbSendLength,  
                    SCARD_IO_REQUEST *pioRecvPci,  
                    LPBYTE pbRecvBuffer,  
                    LPDWORD pcbRecvLength);
```

Parameter(s):

In/Out	Parameter	Description
[in]	hCard	Handle from previous call to SCardConnect()
[in]	pioSendPci	Parameter is unused; set to NULL
[in]	pbSendBuffer	APDU to send to card
[in]	cbSendLength	Length of APDU
[in out]	pioRecvPci	Parameter is unused; set to NULL
[out]	pbRecvBuffer	Buffer to receive response from card
[in out]	pcbRecvLength	Length of response buffer at input; Actual length of response from card at outputd

Return Values:

Value	Description
SCARD_S_SUCCESS	When connection/session was terminated successfully
SCARD_E_INVALID_HANDLE	If the card handle provided is not valid
SCARD_E_INSUFFICIENT_BUFFER	cbRecvLength was not sufficient to copy response data from card
SCARD_E_INVALID_PARAMETER	If cbSendLength is less than 4 bytes
SCARD_E_COMM_DATA_LOST	Error exchanging data with accessory

6.9 SCardStatus

Returns the current status of the reader whose handle is passed as parameter.

Prototype:

```
LONG SCardStatus(SCARDHANDLE hCard,  
                 LPSTR szReaderName,  
                 LPDWORD pcchReaderLen,  
                 LPDWORD pdwState,  
                 LPDWORD pdwProtocol,  
                 LPBYTE pbAtr,  
                 LPDWORD pcbAtrLen);
```

Parameter(s):

In/Out	Parameter	Description
[in]	hCard	Handle from previous call to SCardConnect()
[out]	szReaderName	Buffer to copy reader name that corresponds to card handle
[in out]	pcchReaderLen	Length of reader name buffer on input; Number of bytes in reader name on output
[out]	pdwState	Buffer to return card state
[out]	pdwProtocol	Buffer to return active protocol
[out]	pbAtr	Buffer to return ATR of card
[in out]	pcbAtrLen	Length of ATR buffer on input; Number of bytes of ATR of card on output

Return Values:

Value	Description
SCARD_S_SUCCESS	When connection/session was terminated successfully
SCARD_E_INVALID_HANDLE	If the card handle provided is not valid
SCARD_E_INSUFFICIENT_BUFFER	If szReaderName is not NULL, and pcchReaderLen was too short to copy reader name, OR If pbATR is not NULL, and pbAtrLen was too short to copy ATR

6.10 SCardGetAttrib

Retrieve the attributes specified by parameter.

Prototype:

```
LONG SCardGetAttrib(SCARDHANDLE hCard,  
                    DWORD dwAttrId,  
                    LPBYTE pbAttr,  
                    LPDWORD pcbAttrLen);
```

Parameter(s):

In/Out	Parameter	Description
[in]	hCard	Handle from previous call to SCardConnect()
[in]	dwAttrId	Attribute ID to retrieve
[out]	pbAttr	Buffer to return attribute
d[in out]	pcbAttrLen	Length of attribute buffer on input; Number of bytes of attributes on output

Return Values:

Value	Description
SCARD_S_SUCCESS	When connection/session was terminated successfully
SCARD_E_INVALID_HANDLE	If the card handle provided is not valid

Notes:

This is just a placeholder and is not implemented as of v1.0.0.0 of library.

6.11 SCardBeginTransaction

Begin a transaction with card; provides exclusive access temporarily until a call to SCardEndTransaction(), or card is removed or reset.

Prototype:

```
LONG SCardBeginTransaction(SCARDHANDLE hCard );
```

Parameter(s):

In/Out	Parameter	Description
[in]	hCard	Handle from previous call to SCardConnect()

Return Values:

Value	Description
SCARD_S_SUCCESS	When connection/session was terminated successfully
SCARD_E_INVALID_HANDLE	If the card handle provided is not valid

Notes:

This is just a placeholder and is not implemented as of v1.0.0.0 of library.

6.12 SCardEndTransaction

Close/terminate a transaction with card; temporary exclusive access is relinquished.

Prototype:

```
LONG SCardEndTransaction(SCARDHANDLE hCard,  
                          DWORD dwDisposition);
```

Parameter(s):

In/Out	Parameter	Description
[in]	hCard	Handle from previous call to SCardConnect()
[in]	dwDisposition	Desired action on card/reader SCARD_LEAVE_CARD – do nothing SCARD_RESET_CARD – reset the card (warm reset) SCARD_UNPOWER_CARD – power down the card (cold reset) SCARD_EJECT_CARD – power down and eject the card

Return Values:

Value	Description
SCARD_S_SUCCESS	When connection/session was terminated successfully
SCARD_E_INVALID_HANDLE	If the card handle provided is not valid

Notes:

This is just a placeholder and is not implemented as of v1.0.0.0 of library.

7. SDK Files

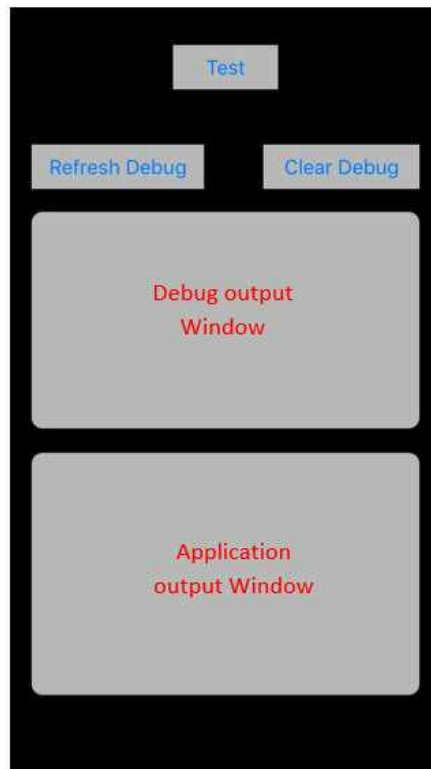
The SDK comprises of the following folders

Folder	Description
/sdk/inc	Include files of SDK
/sdk/lib	Lib file of SDK
/iAuth2demo	Sample Application that uses SDK

8. Sample Application

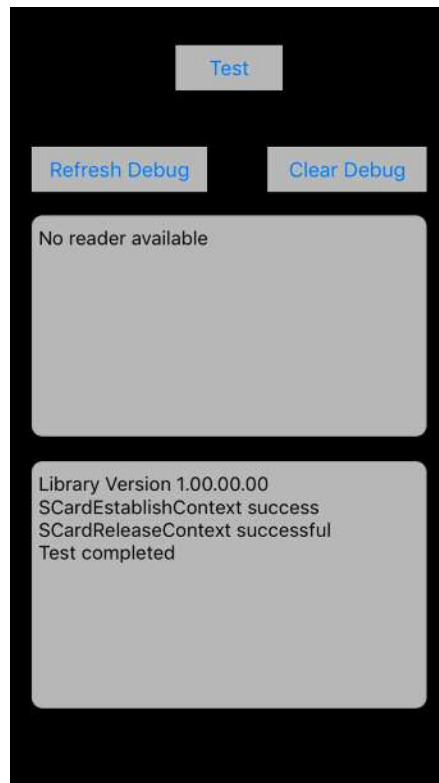
The sample application provides a brief demonstration of how to use the SDK by making calls into the SDK library. Open the iAuth2demo project in XCode and build it. Connect the iOS device to host to download the app to the iOS device.

App opening screen is shown below:



It has two text windows: a top one for displaying debug output, and a bottom one to display application output messages. The “Refresh Debug” button executes `RetrieveDebugData()` and updates Debug output Window. The “Clear Debug” button clears the Debug output Window. The “Test” button shall make a series of calls into the SDK library and displays the results in Application output window.

The following screenshot shows the results of executing the application without the accessory plugged in:



The debug window shows the error message from SDK. Enabling debug messages in SDK allows to determine where the issue happened.

The sample app has the debug level set to `LOG_LEVEL_ERROR`, which is the default in the SDK library. Uncommenting the `SetDebugLevel(LOG_LEVEL_ALL)` shall show more descriptive messages in debug windows from SDK.

The screenshots below show the output when app is executed with accessory plugged in.



9. Checklist for Developers

Include the SDK header files in application	
Include the SDK library files during linker	
Ensure that the EA protocol strings (refer Section 2 above) are specified as <i>UISupportedEAPProtocols</i> string array in the Info.plist file for the application	
It is not recommended to call the API from main thread (or UI thread) of the app. Since external accessory communication can take a while, it would lock up the app resulting in app being terminated by iOS.	