IDENTIVE

Android CCID Library User Manual

Version 2.1

Document History

Date	Version	Description of Changes
3rd July 2012	1.0	Initial version
7th September 2012	2.0	Added documentation for 3 new API's – SCardReconnect, SCardGetStatusChange, SCardGetAttrib
16th February 2023	2.1	Added document for 1 new API - SCardListener

Contents

1.0 Legal Information

2.0 Licenses

3.0 Trademarks

Android is a trademark of Google Inc.

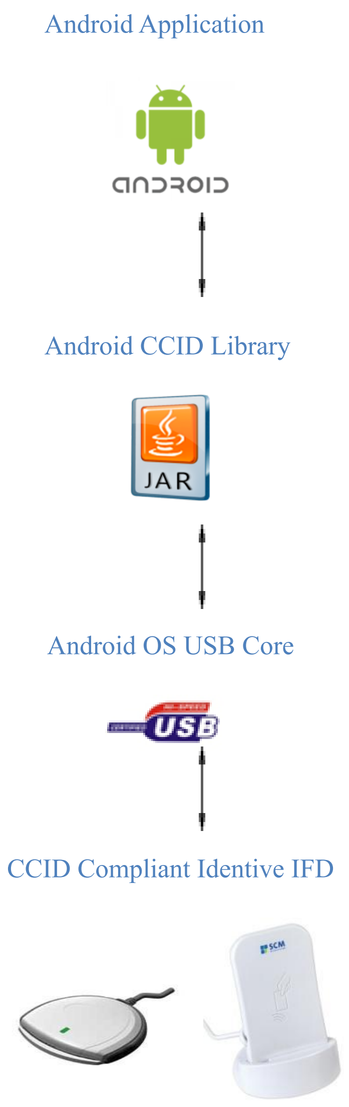
4.0 Introduction

Android CCID library serves as an interface between Android platform with USB host support and Identive CCID compliant USB smartcard readers. Android application developers will integrate this library as part of their Android application to communicate with Identive's CCID readers with VID 0x04E6 or 0x1FFA.

5.0 Terms and Abbreviations

Term/ Abbreviation	Description
ADK	Android Development Kit
API	Application Programming Interface
CCID	Chip Card Interface Device
ICC	Integrated Circuit Card
IFD	Interface Device
ISO	International Standardization Organization
JAR	Java Archive
PC/SC	Personal Computer / Smart Card interface
USB	Universal Serial Bus
VID	Vendor ID

6.0 Software Design Overview



7.0 Minimum Requirements

1. USB host mode is supported in Android 3.1 and higher, hence the device should have Android 3.1 or above.

2. The Android device should support USB host mode. Please refer to the device technical specification for details.

8.0 Prerequisites

The Application developer should have the basic knowledge of the following

1. Java and Android Programming
2. Use of Eclipse for android

9.0 SDK Contents

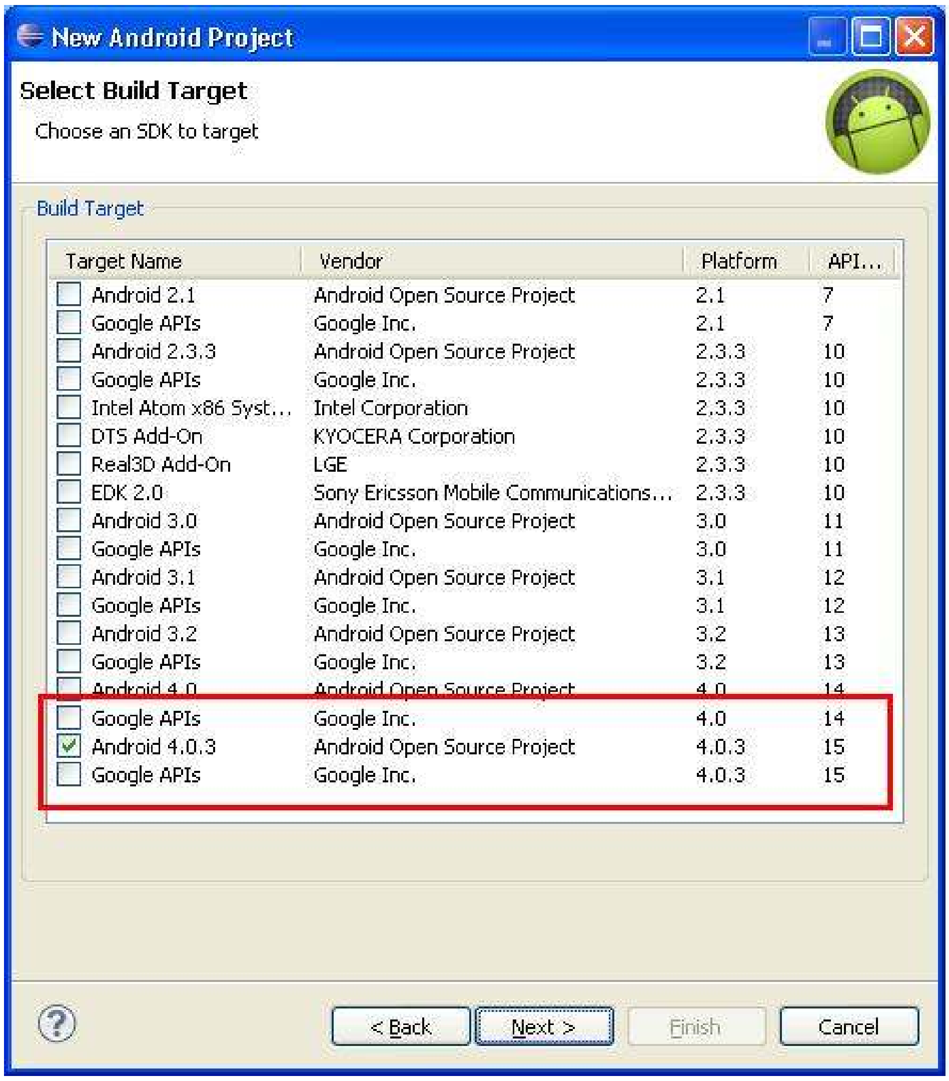
The SDK Package holds the following

1. Library Folder contains
   a. androidSCard-1.3.jar – to be used for the application development
2. Sample Application Folder contains
   a. IdentiveGetZv1.1 Eclipse Source Code – to be used for reference
   b. IdentiveGetZV1.1.apk
3. Android_CCID_Library_User_Manual.pdf

10.0 Adding JAR file in Eclipse

1. Open Eclipse IDE; create a new Android project from FILE NEW Android Project. Enter a project name e.g. “SampleSCard” and click NEXT

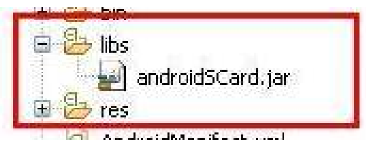
2. In the “Select Build Target” window, choose API version 13 and above (i.e.) Android 3.2 and above for USB support and click “Next” to provide a suitable package name and then click “Finish”.



	Target Name	Vendor	Platform	API...
☐	Android 2.1	Android Open Source Project	2.1	7
☐	Google APIs	Google Inc.	2.1	7
☐	Android 2.3.3	Android Open Source Project	2.3.3	10
☐	Google APIs	Google Inc.	2.3.3	10
☐	Intel Atom x86 Syst...	Intel Corporation	2.3.3	10
☐	DTS Add-On	KYOCERA Corporation	2.3.3	10
☐	Real3D Add-On	LGE	2.3.3	10
☐	EDK 2.0	Sony Ericsson Mobile Communications...	2.3.3	10
☐	Android 3.0	Android Open Source Project	3.0	11
☐	Google APIs	Google Inc.	3.0	11
☐	Android 3.1	Android Open Source Project	3.1	12
☐	Google APIs	Google Inc.	3.1	12
☐	Android 3.2	Android Open Source Project	3.2	13
☐	Google APIs	Google Inc.	3.2	13
☐	Android 4.0	Android Open Source Project	4.0	14
☐	Google APIs	Google Inc.	4.0	14
✓	Android 4.0.3	Android Open Source Project	4.0.3	15
☐	Google APIs	Google Inc.	4.0.3	15

3. A new project with the name “SampleSCard” is created in the project Explorer.

4. Now Create a New Folder named “libs” in the project root and copy the “androidSCard.jar” to this folder

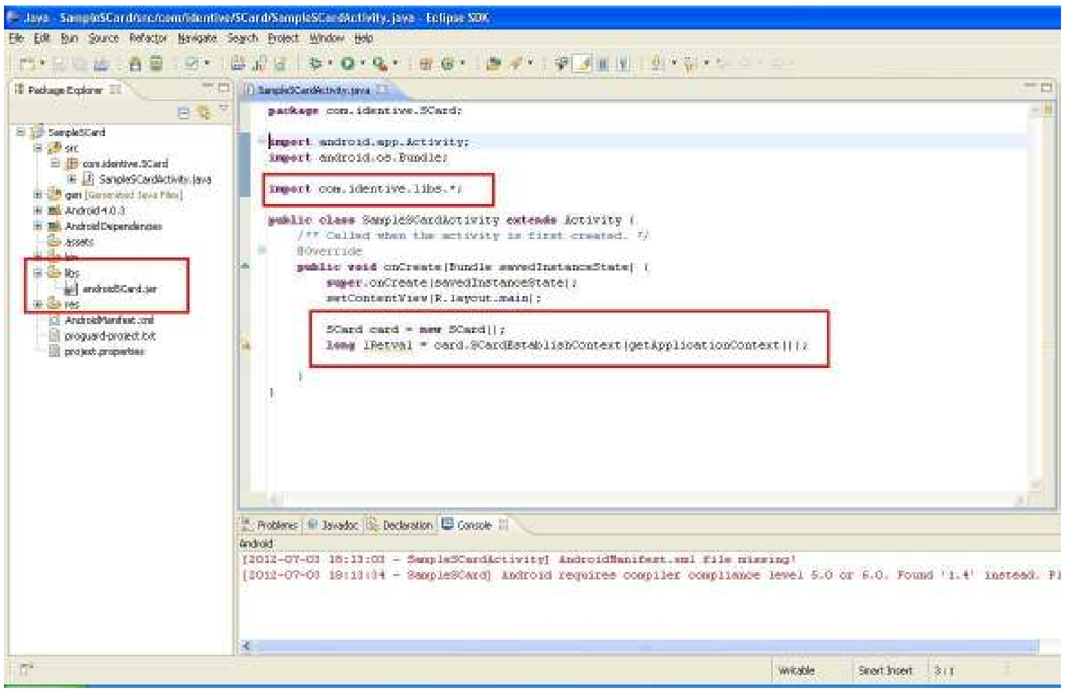


5. Import the library into the project.

```
import com.identive.libs.*;
```

6. Create an object for the class “SCard” to use the APIs.

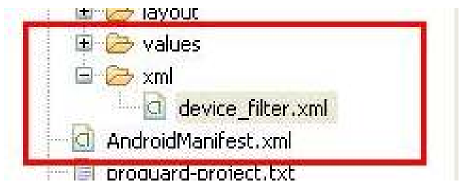
```
SCard card = new SCard();
long lRetval = card.SCardEstablishContext(getApplicationContext());
```

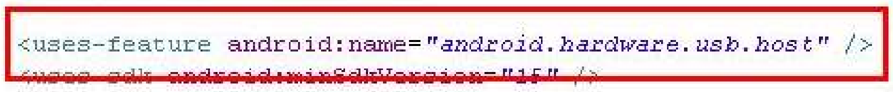


```
package com.identive.SCard;

import android.app.Activity;
import android.os.Bundle;

import com.identive.libs.*;

public class SampleSCardActivity extends Activity {
    /** Called when the activity is first created. */
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);

        SCard card = new SCard();
        long lRetval = card.SCardEstablishContext(getApplicationContext());
    }
}
```

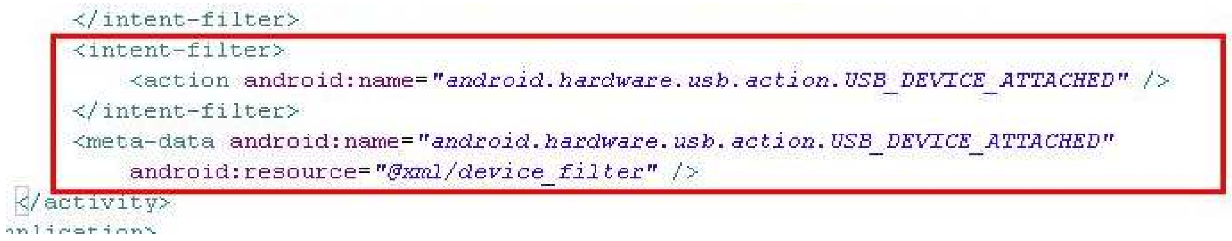
```
Android
[2012-07-03 18:13:03 - SampleSCardActivity] AndroidManifest.xml file missing!
[2012-07-03 18:13:04 - SampleSCard] Android requires compiler compliance level 5.0 or 6.0. Found '1.4' instead. Pl
```

7. Include the following details in the application Manifest for auto launch of application on device arrival. Please refer to the sample application code for detailed information.

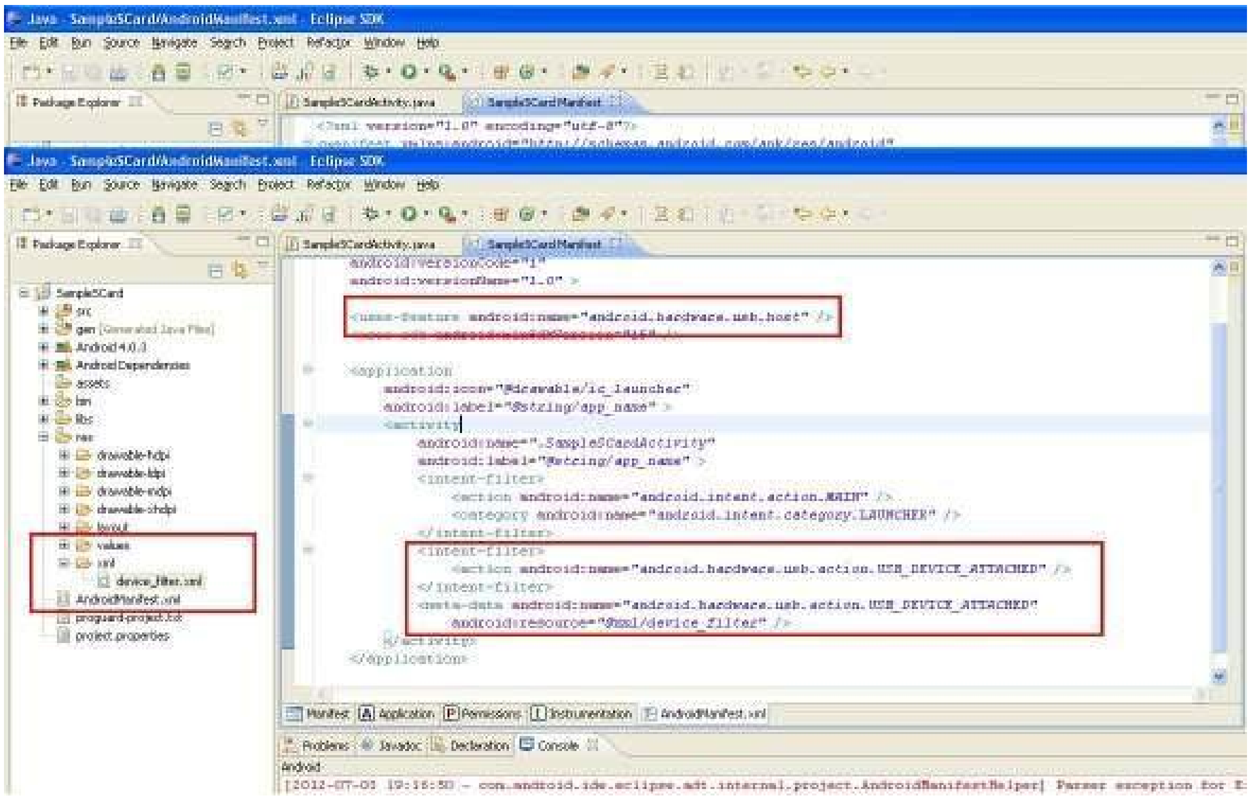




```
<uses-feature android:name="android.hardware.usb.host" />
```



```
</intent-filter>
<intent-filter>
    <action android:name="android.hardware.usb.action.USB_DEVICE_ATTACHED" />
</intent-filter>
<meta-data android:name="android.hardware.usb.action.USB_DEVICE_ATTACHED"
    android:resource="@xml/device_filter" />
</activity>
```



```
    android:versionCode="1"
    android:versionName="1.0" >

    <uses-feature android:name="android.hardware.usb.host" />

    <application
        android:icon="@drawable/ic_launcher"
        android:label="@string/app_name" >
        <activity
            android:name=".SampleSCardActivity"
            android:label="@string/app_name" >
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />
                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
            <intent-filter>
                <action android:name="android.hardware.usb.action.USB_DEVICE_ATTACHED" />
            </intent-filter>
            <meta-data android:name="android.hardware.usb.action.USB_DEVICE_ATTACHED"
                android:resource="@xml/device_filter" />
        </activity>
    </application>
```

```
Android
[2012-07-03 19:16:50 - com.android.ide.eclipse.adt.internal.project.AndroidManifestHelper] Parser exception for E
```

11.0 List of API's supported by the library

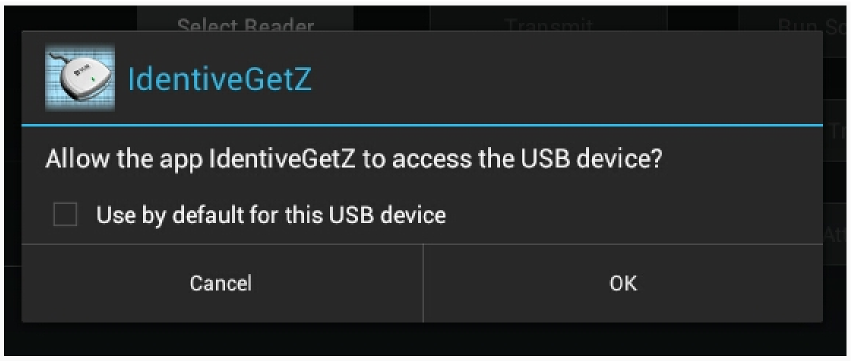
Current list of API's supported by the library. Please refer to the "WinDefs.java" class for values of input parameter definitions e.g. SCARD_PROTOCOL_T0, SCARD_PROTOCOL_T1 etc.

1. USBRequestPermission
2. SCardEstablishContext
3. SCardListReaders
4. SCardConnect
5. SCardStatus
6. SCardTransmit
7. SCardControl
8. SCardDisconnect
9. SCardReleaseContext
10. SCardReconnect
11. SCardGetStatusChange
12. SCardGetAttrib
13. SCardListener

11.1 USBRequestPermission

Applications must get user permission to access USB devices connected to an Android host. The **USBRequestPermission** API helps the application developer to get access rights for Identive's CCID device connected to the host. If the user does not grant access SCardListReaders will fail.

Note*: The application developer should take care of calling this function at appropriate location such as onCreate() function of the activity so that he gets the user authentication before proceeding with other Scard API calls. This function will pop up a dialog as shown below where the user has to grant the access. If the application calls any Scard API's before the user grants permission then it will fail.*



IdentiveGetZ

Allow the app IdentiveGetZ to access the USB device?

☐ Use by default for this USB device

Syntax:
LONG USBRequestPermission (
__in CONTEXT context
);

Parameters:
context [in]
The application context

Return value:
Please refer to the "List of Error Codes" section for details on return values.

11.2 SCardEstablishContext

The **SCardEstablishContext** function establishes the handle to the USB Service using the UsbManager API.

Syntax:

```
LONG SCardEstablishContext (
_in   CONTEXT context
);
```

Parameters:

context [in]

The application context

Return value:

Please refer to the “List of Error Codes” section for details on return values.

11.3 SCardListReaders

The SCardListReaders function provides the list of Identive readers as an array list.

Syntax:

```
LONG SCardListReaders (
 _in CONTEXT context,
_out ARRAYLIST<STRING> deviceList
);
```

Parameters:

context[in]

The application context or the base context

deviceList[out]

An array list of String values containing the names of connected Identive readers

Return value:

Please refer to the “List of Error Codes” section for details on return values.

11.4 SCardConnect

The **SCardConnect** function establishes a connection between the calling application and a smart card contained by a specific reader. In order to communicate with the card we have to connect using "SCARD_SHARE_EXCLUSIVE" mode and if we want to communicate with the reader (when card is not needed) we have to use SCARD_SHARE_DIRECT. This mode is usually used for sending escape IOCTL's to the reader.

Syntax:

```
LONG SCardConnect (
  _in  STRING   szReader,
  _in  INT      nMode,
  _in  INT      nPreferredProtocols
);
```

Parameters:

szReader [in]

The name of the reader that contains the target card.

nMode[in]

A flag that indicates whether other applications may form connections to the card.

Value	Meaning
SCARD_SHARE_EXCLUSIVE	This application is not willing to share the card with other applications.
SCARD_SHARE_DIRECT	This application is allocating the reader for its private use, and will be controlling it directly. No other applications are allowed access to it.

nPreferredProtocols [in]

A bitmask of acceptable protocols for the connection. Possible values may be combined with the **OR** operation.

Value	Meaning
SCARD_PROTOCOL_T0	*T=0* is an acceptable protocol.
SCARD_PROTOCOL_T1	*T=1* is an acceptable protocol.
SCARD_PROTOCOL_Tx	If the protocol of the card is not known this value can be used so that this is the OR of above two protocol values.
SCARD_PROTOCOL_UNDEFINED	Should be used in case of "SCARD_SHARE_DIRECT" connection

Return value:
Please refer to the "List of Error Codes" section for details on return values.

11.5 SCardStatus

The **SCardStatus** function provides the current status of a *smart card* in a *reader*. You can call it any time after a successful call to **SCardConnect** and before a successful call to **SCardDisconnect**. It does not affect the *state* of the reader. Please refer to the section "Class definition" for details on SCARDSTATE class.

Syntax:
```
LONG SCardStatus (
  _out      SCARDSTATE cardstate
);
```

Parameters:
cardstate [in]

An Object of the class SCARDSTATE

Return value:
Please refer to the "List of Error Codes" section for details on return values.

11.6 SCardTransmit

The **SCardTransmit** function sends the command to the *smart card* and expects to receive data back from the card. If the command involves chaining then it is automatically taken care by the library and the final output is given back to the application. Please refer section "Class definition" for details on SCARDIOBUFFER class.

Syntax:

```
LONG SCardTransmit (
  _in/out       SCARDIOBUFFER transmit
);
```

Parameters:

transmit [in/out]

An Object of the class SCARDIOBUFFER

Return value:

Please refer to the "List of Error Codes" section for details on return values.

11.7 SCardControl

The **SCardControl** function can be used to send escape IOCTL to the reader. You can call it any time after a successful call to **SCardConnect** and before a successful call to **SCardDisconnect**. For a list of escape IOCTL codes and return values please refer to the respective reader's user manual.

Syntax:

```
LONG SCardControl (
  _in/out       SCARDIOBUFFER transmit
);
```

Parameters:

transmit [in/out]

An Object of the class SCARDIOBUFFER

Return value:

Please refer to the "List of Error Codes" section for details on return values.

11.8 SCardDisconnect

The **SCardDisconnect** function terminates a connection previously opened between the calling application and a *smart card* in the target *reader*.

Syntax:

```
LONG SCardDisconnect (
 __in  INT nDisposition
);
```

Parameters:

nDisposition [in]

Action to take on the card in the connected reader on close.

Value	Meaning
SCARD_LEAVE_CARD	Do not do anything special.
SCARD_RESET_CARD	Reset the card.
SCARD_UNPOWER_CARD	Power down the card.

Return value:

Please refer to the “List of Error Codes” section for details on return values.

11.9 SCardReleaseContext

The **SCardEstablishContext** function releases the handle to the USB Service.

Syntax:

```
LONG SCardReleaseContext ();
```

Return value:

Please refer to the “List of Error Codes” section for details on return values.

11.10 SCardReconnect

The **SCardReconnect** function reestablishes an existing connection between the calling application and a *smart card*.

Syntax:

```
LONG SCardReconnect(
        __in     INT      nMode,
        __in     INT      nPreferredProtocols,
        __in     INT      nInitialization
   );
```

Parameters:

nMode [in]

Flag that indicates whether other applications may form connections to this card.

Value	**Meaning**
SCARD_SHARE_EXCLUSIVE	This application will not share this card with other applications.

nPreferredProtocols [in]

Bitmask of acceptable protocols for this connection. Possible values may be combined with the **OR** operation. The value of this parameter should include the current protocol. Attempting to reconnect with a protocol other than the current protocol will result in an error.

Value	**Meaning**
SCARD_PROTOCOL_T0	*T=0* is an acceptable protocol.
SCARD_PROTOCOL_T1	*T=1* is an acceptable protocol.
SCARD_PROTOCOL_Tx	If the protocol of the card is not known this value can be used so that this is the OR of above two protocol values.

nInitialization [in]

Type of initialization that should be performed on the card.

Value	Meaning
SCARD_LEAVE_CARD	Do not do anything special on reconnect.
SCARD_RESET_CARD	Reset the card (Warm Reset).
SCARD_UNPOWER_CARD	Power down the card and reset it (Cold Reset).

Return value:
Please refer to the "List of Error Codes" section for details on return values.

11.11 SCardGetStatusChange

The **SCardGetStatusChange** function blocks execution until the current state of the card in a specific reader changes.

The caller supplies maximum amount of time (in milliseconds) or WinDefs.INFINITE to wait infinitely for the state change to occur. The SCARD_READERSTATE array can hold only one value, since only one reader per instance is supported currently. The object to the SCARD_READERSTATE class (*rgReaderStates*) carries the current state of the reader in *nCurrentState* member. The function returns when there is a change of state, having filled in the *nEventState* member of *rgReaderStates* appropriately.

Syntax:
```
LONG SCardGetStatusChange (
  __in    LONG  lTimeout,
  __in/out SCARD_READERSTATE [] rgReaderStates,
  __in    INT      nReaders
);
```

Parameters:

lTimeout [in]

The maximum amount of time in milliseconds to wait for a state change to occur. A value of zero causes the function to return immediately. A value of WinDefs.INFINITE causes this function to wait infinitely for the state change to occur.

rgReaderStates [in, out]

SCARD_READERSTATE objects that specify the reader to watch, and that receive the result.

nReaders [in]

The number of elements in the *rgReaderStates* array. Currently we support only one reader hence this value should always be set to one.

Return value:

Please refer to the "List of Error Codes" section for details on return values.

11.12 SCardGetAttrib

The **SCardGetAttrib** function retrieves the current reader attributes for the given handle. It does not affect the state of the reader, driver, or card.

Syntax:

```
LONG SCardGetAttrib (
  __in/out        SCARDATTRIBUTE attribute
);
```

Parameters:

attribute [in]

An Object of the class SCARDATTRIBUTE

Return value:

Please refer to the "List of Error Codes" section for details on return values.

11.13 SCardListener

The **SCardListener** function sets a SCardListener object that handles card insertion/removal events.

Syntax:

```
VOID SCardGetAttrib (
  __in/out        SCARDLISTENER listener
);
```

Parameters:

attribute [in]

An Object of a class that implement the interface SCARDLISTENER

12.0 Class Definition

SCardIOBuffer

```
public class SCardIOBuffer{
            __in byte[] abyInBuffer;
            __in int nInBufferSize;
            __out byte[] abyOutBuffer;
            __in int nOutBufferSize;
           __out int nBytesReturned;
}
```

byte[] abyInBuffer

Buffer of data sent to the card/reader

int nInBufferSize

The command length, in bytes, of the abyInBuffer parameter.

byte[] abyOutBuffer

Buffer of data returned from card/reader

int nOutBufferSize

The Max length, in bytes, of the abyOutBuffer parameter

int nBytesReturned

The actual length, in bytes, of the data returned from the reader.

SCardState

```
public class  SCardState{
        __out String szReader;
        __out int nState;
        __out int nProtocol;
        __out byte[] abyATR;
        __out int nATRlen;
}
```

String szReader
Name by which the currently connected reader is known

int nState
Current *state* of the smart card in the reader. Upon success, it receives one of the following state indicators

Value	Meaning
SCARD_ABSENT	There is no card in the reader.
SCARD_PRESENT	There is a card in the reader, but it has not been moved into position for use.
SCARD_SWALLOWED	There is a card in the reader in position for use. The card is not powered.
SCARD_POWERED	Power is being provided to the card, but the reader driver is unaware of the mode of the card.
SCARD_NEGOTIABLE	The card has been reset and is awaiting PTS negotiation.
SCARD_SPECIFIC	The card has been reset and specific *communication protocols* have been established.

int nProtocol

Current protocol, if any. The returned value is meaningful only if the returned value of nState is SCARD_SPECIFICMODE.

Value	Meaning
SCARD_PROTOCOL_RAW	The Raw Transfer protocol is in use.
SCARD_PROTOCOL_T0	The ISO 7816/3 *T=0* protocol is in use.
SCARD_PROTOCOL_T1	The ISO 7816/3 *T=1* protocol is in use.

byte[] abyATR

A 32-byte buffer that receives the *ATR string* from the currently inserted card, if available.

int nATRlen

On input, supplies the length of the abyATR buffer. On output, receives the number of bytes in the ATR string (32 bytes maximum)

SCARD_READERSTATE

```
public class SCARD_READERSTATE{
            __in String         szReader;
            __in byte []        pvUserData;
            __in int            nCurrentState;
            __out int           nEventState;
            __out int           nAtr;
            __out byte []       abyAtr = new byte[36];
}
```

String szReader

Name by which the currently connected reader is known

byte [] pvUserData

Not used by the smart card subsystem. This member is used by the application.

int nCurrentState;

Current *state* of the reader, as seen by the application. This field can take on any of the following values, in combination, as a bitmask.

Value	Meaning
SCARD_STATE_UNAWARE	The application is unaware of the current *state*, and would like to know. The use of this value results in an immediate return of the current state of the reader.
SCARD_STATE_UNAVAILABLE	The application expects that this reader is not available for use. The use of this value results in an immediate return of the current state of the reader causing a state change.
SCARD_STATE_EMPTY	The application expects that there is no card in the reader. The use of this value results in return if there is a state change to SCARD_STATE_PRESENT.
SCARD_STATE_PRESENT	The application expects that there is a card in the reader. The use of this value results in return if there is a state change to SCARD_STATE_ EMPTY.

int nEventState;

Current *state* of the *reader*, as known by the *smart card resource manager*. This field can take on any of the following values, in combination, as a bitmask.

Value	Meaning
SCARD_STATE_CHANGED	There is a difference between the state believed by the application, and the state known by the resource manager. When this bit is set, the application may assume a significant state change has occurred on this reader.
SCARD_STATE_UNKNOWN	The given reader name is not recognized by the resource manager.
SCARD_STATE_UNAVAILABLE	The actual state of this reader is not available.
SCARD_STATE_EMPTY	There is no card in the reader.
SCARD_STATE_PRESENT	There is a card in the reader.

int nAtr;

Number of bytes in the returned ATR.

byte [] abyAtr

ATR of the inserted card, with extra alignment bytes.

SCardAttribute

```
public class SCardAttribute{
        __in int nAttrId;
        __out int nAttrLen;
        __out byte[] abyAttr;
}
```

int nAttrId

Identifier for the attribute to get. The following table lists possible values for nAttrId. These values are read-only. Note that vendors may not support all attributes. The value for each definition can be got from the WinDefs class. E.g WinDefs.SCARD_ATTR_ATR_STRING

Value	Meaning
SCARD_ATTR_ATR_STRING	Answer to reset (ATR) string.
SCARD_ATTR_CHANNEL_ID	**DWORD** encoded as 0xDDDDCCCC, where DDDD = data channel type and CCCC = channel number: ● The following encodings are defined for DDDD: ● 0x20 USB; CCCC is device number.
SCARD_ATTR_CURRENT_BWT	Current block waiting time.
SCARD_ATTR_CURRENT_CLK	Current clock rate, in kHz.
SCARD_ATTR_CURRENT_CWT	Current character waiting time.
SCARD_ATTR_CURRENT_D	Bit rate conversion factor.
SCARD_ATTR_CURRENT_EBC_ENCODING	Current error block control encoding. 0 = longitudinal redundancy check (LRC) 1 = cyclical redundancy check (CRC)
SCARD_ATTR_CURRENT_F	Clock conversion factor.
SCARD_ATTR_CURRENT_IFSC	Current byte size for information field size card.
SCARD_ATTR_CURRENT_IFSD	Current byte size for information field size device.

SCARD_ATTR_CURRENT_N	Current guard time.
SCARD_ATTR_CURRENT_PROTOCOL_TYPE	**DWORD** encoded as 0x0rrrpppp where rrr is RFU and should be 0x000. pppp encodes the current protocol type. Whichever bit has been set indicates which ISO protocol is currently in use. (For example, if bit zero is set, T=0 protocol is in effect.)
SCARD_ATTR_CURRENT_W	Current work waiting time.
SCARD_ATTR_DEFAULT_CLK	Default clock rate, in kHz.
SCARD_ATTR_DEFAULT_DATA_RATE	Default data rate, in bps.
SCARD_ATTR_DEVICE_FRIENDLY_NAME	Reader's display name.
SCARD_ATTR_DEVICE_SYSTEM_NAME	Reader's system name.
SCARD_ATTR_DEVICE_UNIT	Instance of this vendor's reader attached to the computer. The first instance will be device unit 0, the next will be unit 1 (if it is the same brand of reader) and so on. Two different brands of readers will both have zero for this value.
SCARD_ATTR_ICC_INTERFACE_STATUS	Single byte. Zero if smart card electrical contact is not active; nonzero if contact is active.
SCARD_ATTR_ICC_PRESENCE	Single byte indicating smart card presence: 0 = not present 1 = card present but not swallowed (applies only if reader supports smart card swallowing) 2 = card present (and swallowed if reader supports smart card swallowing) 4 = card confiscated.
SCARD_ATTR_ICC_TYPE_PER_ATR	Single byte indicating smart card type: 0 = unknown type 1 = 7816 Asynchronous Other values RFU.
SCARD_ATTR_MAX_CLK	Maximum clock rate, in kHz.
SCARD_ATTR_MAX_DATA_RATE	Maximum data rate, in bps.
SCARD_ATTR_MAX_IFSD	Maximum bytes for information file size device.

SCARD_ATTR_POWER_MGMT_SUPPORT	Zero if device does not support power down while smart card is inserted. Nonzero otherwise.
SCARD_ATTR_PROTOCOL_TYPES	**DWORD** encoded as 0x0rrrpppp where rrr is RFU and should be 0x000. pppp encodes the supported protocol types. A '1' in a given bit position indicates support for the associated ISO protocol, so if bits zero and one are set, both T=0 and T=1 protocols are supported.
SCARD_ATTR_VENDOR_IFD_SERIAL_NO	Vendor-supplied interface device serial number.
SCARD_ATTR_VENDOR_IFD_TYPE	Vendor-supplied interface device type (model designation of reader).
SCARD_ATTR_VENDOR_IFD_VERSION	Vendor-supplied interface device version (**DWORD** in the form 0xMMmmbbbb whereMM = major version, mm = minor version, and bbbb = build number).
SCARD_ATTR_VENDOR_NAME	Vendor name.

int nAttrLen

Length (in bytes) of the attribute value in the *abyAttr* buffer

byte[] abyAttr

Buffer that supplies the attribute whose ID is supplied in *n*AttrId

13.0 List of Error Codes

ERROR CODE	VALUE
SCARD_S_SUCCESS	0x00
List of Error values	http://msdn.microsoft.com/en-us/library/ms936965.aspx