

**EZ PC/SC Series  
Smart Card API  
Reference Manual  
v2.1**

# Table Of Contents

WARNING .....	4
ABOUT THIS MANUAL .....	4
1. Smart Card API Overview .....	5
Smart Card Database Query Functions .....	5
Smart Card Database Management Functions .....	5
Direct Card Access Functions .....	6
Resource Manager Context Functions .....	6
Resource Manager Support Function .....	6
Smart Card Tracking Functions .....	6
Smart Card and Reader Access Functions .....	7
2. Smart Card Structure .....	8
SCARD_IO_REQUEST .....	8
SCARD_READERSTATE .....	8
OPENCARDNAME_EX .....	10
OPENCARD_SEARCH_CRITERIA .....	13
3. API Reference .....	15
CasAddReaderToGroup() .....	15
CasBeginTransaction() .....	16
CasCancel() .....	17
CasConnect() .....	18
CasControl() .....	20
CasDisconnect() .....	21
CasEndTransaction() .....	22
CasEstablishContext() .....	23
CasForgetCardType() .....	24
CasForgetReader() .....	25
CasForgetReaderGroup() .....	26
CasFreeMemory() .....	27
CasGetAttrib() .....	28
CasGetCardTypeProviderName() .....	29
CasGetProviderId() .....	31
CasGetStatusChange() .....	32
CasIntroduceCardType() .....	33
CasIntroduceReader() .....	34
CasIntroduceReaderGroup() .....	35
CasListCards() .....	36
CasListInterfaces() .....	38
CasListReaderGroups() .....	39
CasListReaders() .....	40
CasLocateCards() .....	41
CasReconnect() .....	42
CasReleaseContext() .....	44
CasRemoveReaderFromGroup() .....	45
CasSetAttrib() .....	46

CasStatus().....	47
CasTransmit() .....	49
CasUIDlgSelectCard() .....	52
4. Smart Card Error Codes .....	53

## **WARNING**

Information in this document is subject to change without prior notice.

All trademarks mentioned are proprietary of their respective owners.

## **ABOUT THIS MANUAL**

This manual describes the smart card (CPU card) API (Application Programming Interface) functions of EZ PC/SC series smart card reader; Application software developers should refer to this manual to develop their own software for access smart card via EZ PC/SC series smart card reader.

## 1. Smart Card API Overview

### Smart Card Database Query Functions

These functions query the smart card database. They can provide a list of smart cards supplied by a specific user, the interfaces and primary service provider of a specific card, the reader groups defined for the system, and the readers within a set of reader groups. When using these functions, you can query the complete smart card database, or you can narrow the search by setting the resource manager context. The resource manager context is set by calling `CasEstablishContext()` before calling a query function.

**Note** Without a specified context, some information may still be inaccessible due to security restrictions.

To...	Call...
Retrieve the identifier (GUID) of the primary service provider for the given card.	<b>CasGetProviderId</b>
Retrieve a list of cards previously introduced to the system by a specific user.	<b>CasListCards</b>
Retrieve the identifiers (GUIDs) of the interfaces supplied by a given card.	<b>CasListInterfaces</b>
Retrieve a list of reader groups that have previously been introduced to the system.	<b>CasListReaderGroups</b>
Retrieve the list of readers within a set of named reader groups.	<b>CasListReaders</b>

### Smart Card Database Management Functions

These functions manage the smart card database, updating the database using a specified resource manager context.

**Note** Database security is maintained by placing access restrictions on the database, rather than by adding security mechanisms to the smart card subsystem.

To...	Call...
Add a reader to a reader group.	<b>CasAddReaderToGroup</b>
Remove a smart card from the system.	<b>CasForgetCardType</b>
Remove a reader from the system.	<b>CasForgetReader</b>
Remove a reader group from the system.	<b>CasForgetReaderGroup</b>
Introduce a new card to the system.	<b>CasIntroduceCardType</b>
Introduce a new reader to the system.	<b>CasIntroduceReader</b>
Introduce a new reader group to the system.	<b>CasIntroduceReaderGroup</b>
Remove a reader from a reader group.	<b>CasRemoveReaderFromGroup</b>

## Direct Card Access Functions

The smart card subsystem lets you communicate with cards that may not conform to the ISO 7816 specifications. To do this, these functions allow you to control the attributes of the communications between the application and the card by giving you direct, low-level manipulation of the reader.

To...	Call...
Provide direct control of the reader.	<b>CasCtrl</b>
Get reader attributes.	<b>CasGetAttrib</b>
Set reader attribute.	<b>CasSetAttrib</b>

## Resource Manager Context Functions

These functions establish and release the resource manager context that is used by the database query and database management functions.

To...	Call...
Establishes a context for accessing the smart card database.	<b>CasEstablishContext</b>
Closes an established context.	<b>CasReleaseContext</b>

## Resource Manager Support Function

This function frees memory allocated through the use of the SCARD\_AUTOALLOCATE length designator, simplifying the use of the other resource manager functions.

To...	Call...
Release memory returned through the use of SCARD_AUTOALLOCATE.	<b>CasFreeMemory</b>

## Smart Card Tracking Functions

These functions let you track cards within readers. These routines typically use the SCARD\_READERSTATE structure within an array.

To...	Call...
Search for a card whose ATR string matches a supplied card name.	<b>CasLocateCards</b>
Block execution until the current availability of cards changes.	<b>CasGetStatusChange</b>
Terminate outstanding actions.	<b>CasCancel</b>

## Smart Card and Reader Access Functions

These functions connect to and communicate with a specific smart card. I/O operations to the card use a buffer for sending or receiving data and a structure that contains protocol control information. The protocol control information structure always begins with an SCARD\_IO\_REQUEST structure.

<b>To...</b>	<b>Call...</b>
Connect to a card.	<b>CasConnect</b>
Reestablish a connection.	<b>CasReconnect</b>
Terminate a connection.	<b>CasDisconnect</b>
Start a transaction, blocking other applications from accessing a card.	<b>CasBeginTransaction</b>
End a transaction, allowing other applications to access a card.	<b>CasEndTransaction</b>
Provide the current status of the reader.	<b>CasStatus</b>
Requests service and receives data back from a card using T=0, T=1, and raw protocols.	<b>CasTransmit</b>

## 2. Smart Card Structure

### SCARD\_IO\_REQUEST

The SCARD\_IO\_REQUEST structure begins a protocol control information structure. Any protocol-specific information then immediately follows this structure. The entire length of the structure must be aligned with the underlying hardware architecture word size. For example, in Win32 the length of any PCI information must be a multiple of 4 bytes so that it aligns on a 32-bit boundary.

```
typedef struct {
    DWORD          dwProtocol
    DWORD          cbPciLength;
} SCARD_IO_REQUEST;
```

#### Members

##### *dwProtocol*

Identifies the protocol in use.

##### *cbPciLength*

Supplies the length, in bytes, of the SCARD\_IO\_REQUEST structure plus any following PCI-specific information.

### SCARD\_READERSTATE

The SCARD\_READERSTATE structure is used by functions for tracking smart cards within readers.

```
typedef struct {
    LPCTSTR      szReader
    LPVOID       pvUserData;
    DWORD        dwCurrentState;
    DWORD        dwEventState;
    DWORD        cbAtr;
    BYTE         rgbAtr[36];
} SCARD_READERSTATE, *PSCARD_READERSTATE, *LPSCARD_READERSTATE;
```

#### Members

##### *szReader*

Points to the name of the reader being monitored.

##### *pvUserData*

Not used by the smart card subsystem. Used by the application.

##### *dwCurrentState*

Supplies the current state of the reader, as seen by the application. This field can take on any of the following values, in combination, as a bit mask:

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 from state transition monitoring services. This is represented by all bits set to zero.

SCARD_STATE_IGNORE	The application is not interested in this reader, and it should not be considered during monitoring operations. If this bit value is set, all other bits are ignored.
SCARD_STATE_UNAVAILABLE	The application believes that this reader is not available for use. If this bit is set, then all the following bits are ignored.
SCARD_STATE_EMPTY	The application believes that there is not card in the reader. If this bit is set, all the following bits are ignored.
SCARD_STATE_PRESENT	The application believes that there is a card in the reader.
SCARD_STATE_ATRMATCH	The application believes that there is a card in the reader with an ATR matching one of the target cards. If this bit is set, SCARD_STATE_PRESENT is assumed. This bit has no meaning to CasGetStatusChange beyond SCARD_STATE_PRESENT.
SCARD_STATE_EXCLUSIVE	The application believes that the card in the reader is allocated for exclusive use by another application. If this bit is set, SCARD_STATE_PRESENT is assumed.
SCARD_STATE_INUSE	The application believes that the card in the reader is in use by one or more other applications, but may be connected to in shared mode. If this bit is set, SCARD_STATE_PRESENT is assumed.
SCARD_STATE_MUTE	The application believes that there is an unresponsive card in the reader.

**dwEventState**

Receives the 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 bit mask:

<b>Value</b>	<b>Meaning</b>
SCARD_STATE_IGNORE	This reader should be ignored.
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. If this bit is set, then SCARD_STATE_CHANGED and SCARD_STATE_IGNORE will also be set.
SCARD_STATE_UNAVAILABLE	The actual state of this reader is not available. If this bit is set, then all the following bits are clear.

SCARD_STATE_EMPTY	There is no card in the reader. If this bit is set, all the following bits will be clear.
SCARD_STATE_PRESENT	There is a card in the reader.
SCARD_STATE_ATRMATCH	There is a card in the reader with an ATR matching one of the target cards. If this bit is set, SCARD_STATE_PRESENT will also be set. This bit is only returned on the CasLocateCards function.
SCARD_STATE_EXCLUSIVE	The card in the reader is allocated for exclusive use by another application. If this bit is set, SCARD_STATE_PRESENT will also be set.
SCARD_STATE_INUSE	The card in the reader is in use by one or more other applications, but may be connected to in shared mode. If this bit is set, SCARD_STATE_PRESENT will also be set.
SCARD_STATE_MUTE	There is an unresponsive card in the reader.

cbAtr

The number of bytes in the returned ATR.

rgbAtr

The ATR of the inserted card, with extra alignment bytes.

## OPENCARDNAME\_EX

The OPENCARDNAME\_EX structure contains the information that the **CasUIDlgSelectCard** function uses to initialize a smart card Select Card dialog box.

```
typedef struct {
    DWORD                dwStructSize;
    SCARDCONTEXT         hSCardContext;
    HWND                hwndOwner;
    DWORD                dwFlags;
    LPCTSTR              lpstrTitle;
    LPCTSTR              lpstrSearchDesc;
    HICON                hIcon;
    POPENCARD_SEARCH_CRITERIA pOpenCardSearchCriteria;
    LPOCNCONNPROC        lpfnConnect;
    LPVOID                pvUserData;
    DWORD                dwShareMode;
    DWORD                dwPreferredProtocols;
    LPTSTR               lpstrRdr;
    DWORD                nMaxRdr;
    LPTSTR               lpstrCard;
    DWORD                nMaxCard;
    DWORD                dwActiveProtocol;
    SCARDHANDLE          hCardHandle;
} OPENCARDNAME_EX, *POPENCARDNAME_EX, *LPOPENCARDNAME_EX;
```

**Members***dwStructSize*

Specifies the length of the structure, in bytes. The value of this member must not be NULL.

*hSCardContext*

Context used for communication with the smart card resource manager. Call `CasEstablishContext` to set the resource manager context and `CasReleaseContext` to release it. The value of this member must not be NULL.

*hwndOwner*

Identifies the window that owns the dialog box. This member can be any valid window handle, or it can be NULL for the desktop default.

*dwFlags*

A set of bit flags you can use to initialize the dialog box. When the dialog box returns, it sets these flags to indicate the user's input. This member can be one of the following flags:

Flag	Meaning
------	---------

`SC_DLG_MINIMAL_UI` Displays the dialog only if the card being searched for by the calling application is not located and available for use in a reader. This allows the card to be found, connected (either through the internal dialog mechanism or the user callback functions), and returned to the calling application.

`SC_DLG_NO_UI` Force no display of the Select Card user interface (UI), regardless of search outcome.

`SC_DLG_FORCE_UI` Force display of the Select Card UI, regardless of the search outcome.

*lpstrTitle*

Points to a string to be placed in the title bar of the dialog box. If this member is NULL, the system uses the default title "Select Card:".

*lpstrSearchDesc*

Points to a string to be displayed to the user as a prompt to insert the smart card. If this member is NULL, the system uses the default text "Please insert a smart card".

*hIcon*

Handle to an icon (32 x 32 pixels). You can specify a vendor-specific icon to display in the dialog. If this value is NULL, a generic smart card reader – loaded icon is displayed.

*pOpenCardSearchCriteria*

Pointer to the `OPENCARD_SEARCH_CRITERIA` structure to be used, or NULL if one is not used.

*lpfnConnect*

Pointer to the caller's card connect routine. If the caller needs to perform additional processing to connect to the card, this function pointer is set to the user's connect function. If the connect function is successful, the card is left connected and initialized, and the card handle is returned.

The prototype for the connect routine is as follows:

`Connect(``hSCardContext, // the card context passed in the parameter block``szReader,      // the name of the reader``mszCards,      // multistring containing the possible card names in the reader`

pvUserData // pointer to user data passed in parameter block  
);

*pvUserData*

A void pointer to user data. This pointer is passed back to the caller on the Connect routine.

*dwShareMode*

If lpfnConnect is not NULL, the dwShareMode and dwPreferredProtocols members are ignored. If lpfnConnect is NULL and dwShareMode is non-zero, an internal call is made to CasConnect using dwShareMode and dwPreferredProtocols as the dwShareMode and dwPreferredProtocols parameters. If the connect succeeds, hCardHandle is set to the handle returned by CasConnect. If lpfnConnect is NULL and dwShareMode is zero, hCardHandle is set to NULL.

dwPreferredProtocols

Used for internal connection as described in dwShareMode.

*lpstrRdr*

If the card is located, the lpstrRdr buffer contains the name of the reader that contains the located card. The buffer should be at least 256 characters long.

*nMaxRdr*

Specifies the size, in bytes (ANSI version) or characters (UNICODE version), of the buffer pointed to by lpstrRdr. If the buffer is too small to contain the reader information, CasUIDlgSelectCard returns SCARD\_E\_NO\_MEMORY and the required size of the buffer pointed to by lpstrRdr.

*lpstrCard*

If the card is located, the lpstrCard buffer contains the name of the located card. The buffer should be at least 256 characters long.

*nMaxCard*

Specifies the size, in bytes (ANSI version) or characters (UNICODE version), of the buffer pointed to by lpstrCard. If the buffer is too small to contain the card information, CasUIDlgSelectCard returns SCARD\_E\_NO\_MEMORY and the required size of the buffer in nMaxCard.

*dwActiveProtocol*

Returns the actual protocol in use when the dialog makes a connection to a card.

*hCardHandle*

Handle of the connected card (either through an internal dialog connect or an lpfnConnect callback).

## OPENCARD\_SEARCH\_CRITERIA

The `OPENCARD_SEARCH_CRITERIA` structure is used by the `CasUIDlgSelectCard` function in order to recognize cards that meet the requirements set forth by the caller. You can, however, call `CasUIDlgSelectCard` without using this structure.

```
typedef struct {
    DWORD          dwStructSize;
    LPTSTR         lpstrGroupNames;
    DWORD          nMaxGroupNames;
    LPCGUID        rgguidInterfaces;
    DWORD          cguidInterfaces;
    LPTSTR         lpstrCardNames;
    DWORD          nMaxCardNames;
    LPOCNCHKPROC   lpfnCheck;
    LPOCNCONNPROC lpfnConnect;
    LPOCNDSCPROC   lpfnDisconnect;
    LPVOID         pvUserData;
    DWORD          dwShareMode;
    DWORD          dwPreferredProtocols;
} OPENCARD_SEARCH_CRITERIA, *POPENCARD_SEARCH_CRITERIA,
*LPOPENCARD_SEARCH_CRITERIA;
```

### Members:

#### *dwStructSize*

Specifies the length of the structure, in bytes. Must not be NULL.

#### *lpstrGroupNames*

Points to a buffer containing null-terminated group name strings. The last string in the buffer must be terminated by two NULL characters. Each string is the name of a group of cards that is to be included in the search. If `lpstrGroupNames` is NULL, the default group (`Scard$DefaultReaders`) is searched.

#### *nMaxGroupNames*

Maximum number of bytes (ANSI version) or characters (UNICODE version) in the `lpstrGroupNames` string.

#### *rgguidInterfaces*

Reserved for future use. An array of GUIDs identifying the interfaces required. Set this member to NULL for this release.

#### *cguidInterfaces*

Reserved for future use. The number of interfaces in the `rgguidInterfaces` array. Set this member to NULL for this release.

#### *lpstrCardNames*

Points to a buffer containing null-terminated card name strings. The last string in the buffer must be terminated by two NULL characters. Each string is the name of a card that is to be located.

#### *nMaxCardNames*

Maximum number of bytes (ANSI version) or characters (UNICODE version) in the `lpstrCardNames` string.

#### *lpfnCheck*

Pointer to the caller's card verify routine. If no special card verification is required,

this pointer is NULL. If the card is rejected by the verify routine, FALSE is returned and the card will be disconnected. If the card is accepted by the verify routine, TRUE is returned.

The prototype for the check routine is

```
Boolean Check(  
    hSCardContext, // the card context passed in the parameter block  
    hCard,         // card handle  
    pvUserData     // pointer to user data passed in the parameter block  
);
```

#### *lpfnConnect*

Pointer to the caller's card connect routine. If the caller needs to perform additional processing to connect to the card, this function pointer is set to the user's connect function. If the connect function is successful, the card is left connected and initialized, and the card handle is returned.

The prototype for the connect routine is

```
Connect(  
    hSCardContext, // the card context passed in the parameter block  
    szReader,      // the name of the reader  
    mszCards,      // multistring containing the possible card names in the reader  
    pvUserData     // pointer to user data passed in parameter block  
);
```

#### *lpfnDisconnect*

Pointer to the caller's card disconnect routine.

The prototype for the disconnect routine is

```
Disconnect(  
    hSCardContext, // the card context passed in the parameter block  
    hCard,         // card handle  
    pvUserData     // pointer to user data passed in the parameter block  
);
```

Note When you use *lpfnConnect*, *lpfnCheck*, and *lpfnDisconnect*, all three callback procedures should be present. Using these callbacks allows further verification that the calling application has found the appropriate card. This is the best way to ensure the appropriate card is selected. However, when using a non-NULL value for *lpfnCheck*, either both *lpfnConnect* and *lpfnDisconnect* must be non-NULL (and *pvUserData* should also be provided), or *dwShareMode* and *dwPreferredProtocols* must both be set.

#### *pvUserData*

A void pointer to user data. This pointer is passed back to the caller on the *Connect*, *Check*, and *Disconnect* routines.

#### *dwShareMode*

If *lpfnConnect* is not NULL, the *dwShareMode* and *dwPreferredProtocols* members are ignored. If *lpfnConnect* is NULL and *dwShareMode* is non-zero, an internal call is made to *CasConnect* using *dwShareMode* and *dwPreferredProtocols* as the parameter.

#### *dwPreferredProtocols*

Used for internal connection as described in *dwShareMode*.

### 3. API Reference

#### CasAddReaderToGroup()

The **CasAddReaderToGroup** function adds a reader to a reader group.

```
LONG CasAddReaderToGroup(  
    IN SCARDCONTEXT hContext,  
    IN LPCTSTR szReaderName,  
    IN LPCTSTR szGroupName  
);
```

#### Parameters

*hContext*

Supplies the handle that identifies the resource manager context. The resource manager context is set by a previous call to **CasEstablishContext**.

*szReaderName*

Supplies the friendly name of the reader that you are adding.

*szGroupName*

Supplies the friendly name of the group to which you are adding the reader.

#### Return Values

<b>If the function...</b>	<b>The return value is...</b>
Succeeds	SCARD_S_SUCCESS.
Fails	An error code (see <b>Error Codes</b> for a list of all error codes).

#### Remarks

**CasAddReaderToGroup** automatically creates the reader group specified if it does not already exist. **CasAddReaderToGroup** is a database management function.

### CasBeginTransaction()

The **CasBeginTransaction** function starts a transaction, waiting for the completion of all other transactions before it begins.

When the transaction starts, all other applications are blocked from accessing the smart card while the transaction is in progress.

```
LONG CasBeginTransaction(  
    IN SCARDHANDLE hCard  
);
```

### Parameters

*hCard*

Supplies the reference value obtained from a previous call to **CasConnect**.

### Return Values

If the function...	The return value is...
Succeeds	SCARD_S_SUCCESS.
Fails	An error code (see <b>Error Codes</b> for a list of all error codes).

### Remarks

**CasBeginTransaction** is a smart card and reader access function.

## CasCancel()

The **CasCancel** function terminates all outstanding actions within a specific resource manager context.

The only requests that you can cancel are those that require waiting for external action by the smart card or user. Any such outstanding action requests will terminate with a status indication that the action was canceled. This is especially useful to force outstanding **CasGetStatusChange** calls to terminate.

```
LONG CasCancel(  
    IN SCARDCONTEXT hContext  
);
```

### Parameters

*hContext*

Supplies the handle that identifies the resource manager context. The resource manager context is set by a previous call to **CasEstablishContext**.

### Return Values

If the function...	The return value is...
Succeeds	SCARD_S_SUCCESS.
Fails	An error code (see <b>Error Codes</b> for a list of all error codes).

### Remarks

**CasCancel** is a smart card tracking function.

**CasConnect()**

The **CasConnect** function establishes a connection (using a specific resource manager context) between the calling application and a smart card contained by a specific reader. If no card exists in the specified reader, an error is returned.

```
LONG CasConnect(
  IN SCARDCONTEXT hContext,
  IN LPCTSTR szReader,
  IN DWORD dwShareMode,
  IN DWORD dwPreferredProtocols,
  OUT LPSCARDHANDLE phCard,
  OUT LPDWORD pdwActiveProtocol
);
```

**Parameters***hContext*

Supplies the handle that identifies the resource manager context. The resource manager context is set by a previous call to **CasEstablishContext**.

*szReader*

Supplies the name of the reader containing the target card.

*dwShareMode*

Supplies a flag that indicates whether other applications may form connections to the card. Possible values are:

<b>Value</b>	<b>Meaning</b>
SCARD_SHARE_SHARED	This application is willing to share the card with other applications.
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.

*dwPreferredProtocols*

Supplies a bit mask of acceptable protocols for the connection. Possible values, which may be combined with the OR operation, are:

<b>Value</b>	<b>Meaning</b>
SCARD_PROTOCOL_T0	T=0 is an acceptable protocol.
SCARD_PROTOCOL_T1	T=1 is an acceptable protocol.
0	This parameter may be zero only if dwShareMode is set to SCARD_SHARE_DIRECT. In this case, no protocol negotiation will be performed by the drivers until an IOCTL_SMARTCARD_SET_PROTOCOL control directive is sent with <b>CasControl</b> .

*phCard*

Receives a handle that identifies the connection to the smart card in the designated reader.

*pdwActiveProtocol*

Receives a flag that indicates the established active protocol. Possible values are:

Value	Meaning
SCARD_PROTOCOL_T0	T=0 is the active protocol.
SCARD_PROTOCOL_T1	T=1 is the active protocol.
SCARD_PROTOCOL_UNDEFINED	SCARD_SHARE_DIRECT has been specified, so that no protocol negotiation

has

occurred. It is possible that there is no card in the reader.

### Return Values

If the function...	The return value is...
Succeeds	SCARD_S_SUCCESS.
Fails	An error code (see <b>Error Codes</b> for a list of all error codes).

### Remarks

**CasConnect** is a smart card and reader access function.

## CasControl()

The **CasControl** function gives you direct control of the reader. You can call it any time after a successful call to **CasConnect** and before a successful call to **CasDisconnect**. The effect on the state of the reader depends on the control code.

```
LONG CasControl(
    IN SCARDHANDLE hCard,
    IN DWORD dwControlCode,
    IN LPCVOID lpInBuffer,
    IN DWORD nInBufferSize,
    OUT LPVOID lpOutBuffer,
    IN DWORD nOutBufferSize,
    OUT LPDWORD lpBytesReturned
);
```

### Parameters

#### *hCard*

This is the reference value returned from **CasConnect**.

#### *dwControlCode*

Supplies the control code for the operation. This value identifies the specific operation to be performed.

#### *lpInBuffer*

Supplies a pointer to a buffer that contains the data required to perform the operation. This parameter can be NULL if the *dwControlCode* parameter specifies an operation that does not require input data.

#### *nInBufferSize*

Supplies the size, in bytes, of the buffer pointed to by *lpInBuffer*.

#### *lpOutBuffer*

Points to a buffer that receives the operation's output data. This parameter can be NULL if the *dwControlCode* parameter specifies an operation that does not produce output data.

#### *nOutBufferSize*

Supplies the size, in bytes, of the buffer pointed to by *lpOutBuffer*.

#### *lpBytesReturned*

Points to a DWORD that receives the size, in bytes, of the data stored into the buffer pointed to by *lpOutBuffer*.

### Return Values

If the function...	The return value is...
Succeeds	SCARD_S_SUCCESS.
Fails	An error code (see <b>Error Codes</b> for a list of all error codes).

### Remarks

**CasControl** is a direct card access function

## CasDisconnect()

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

```
LONG CasDisconnect(
  IN SCARDHANDLE hCard,
  IN DWORD dwDisposition
);
```

### Parameters

*hCard*

Supplies the reference value obtained from a previous call to **CasConnect**.

*dwDisposition*

Indicates what to do with the card in the connected reader on close. Possible values are:

<b>Value</b>	<b>Meaning</b>
SCARD_LEAVE_CARD	Don't do anything special.
SCARD_RESET_CARD	Reset the card.
SCARD_UNPOWER_CARD	Power down the card.
SCARD_EJECT_CARD	Eject the card.

### Return Values

<b>If the function...</b>	<b>The return value is...</b>
Succeeds	SCARD_S_SUCCESS.
Fails	An error code (see <b>Error Codes</b> for a list of all error codes).

### Remarks

If an application (which previously called **CasConnect**) exits without calling **CasDisconnect**, the card is automatically reset.

**CasDisconnect** is a smart card and reader access function.

## CasEndTransaction()

The **CasEndTransaction** function completes a previously declared transaction, allowing other applications to resume interactions with the card.

```
LONG CasEndTransaction(  
    IN SCARDHANDLE hCard,  
    IN DWORD dwDisposition  
);
```

### Parameters

#### *hCard*

Supplies the reference value obtained from a previous call to **CasConnect**. This value would also have been used in an earlier call to **CasBeginTransaction**.

#### *dwDisposition*

Indicates what to do with the card in the connected reader on close. Possible values are:

<b>Value</b>	<b>Meaning</b>
SCARD_LEAVE_CARD	Don't do anything special.
SCARD_RESET_CARD	Reset the card.
SCARD_UNPOWER_CARD	Power down the card.
SCARD_EJECT_CARD	Eject the card.

### Return Values

<b>If the function...</b>	<b>The return value is...</b>
Succeeds	SCARD_S_SUCCESS.
Fails	An error code (see <b>Error Codes</b> for a list of all error codes).

### Remarks

**CasEndTransaction** is a smart card and reader access function.

## CasEstablishContext()

The **CasEstablishContext** function establishes the resource manager context (the scope) within which database operations are performed.

```
LONG CasEstablishContext(
IN DWORD dwScope,
IN LPCVOID pvReserved1,
IN LPCVOID pvReserved2,
OUT LPSCARDCONTEXT phContext
);
```

### Parameters

#### *dwScope*

Supplies the scope of the resource manager context. Possible values are:

<b>Value</b>	<b>Meaning</b>
SCARD_SCOPE_USER	Database operations are performed within the domain of the user.
SCARD_SCOPE_SYSTEM	Database operations are performed within the domain of the system. (The calling application must have appropriate access permissions for any database actions.)

#### *pvReserved1*

Reserved for future use, and must be NULL. Reserved to allow a suitably privileged management application to act on behalf of another user.

#### *pvReserved2*

Reserved for future use, and must be NULL. Reserved to allow a suitably privileged management application to act on behalf of another terminal.

#### *phContext*

Receives a handle to the established resource manager context. This handle can now be supplied to other functions attempting to do work within this context.

### Return Values

<b>If the function...</b>	<b>The return value is...</b>
Succeeds	SCARD_S_SUCCESS.
Fails	An error code (see <b>Error Codes</b> for a list of all error codes).

### Remarks

The context handle returned by **CasEstablishContext** can be used by database query and management functions.

To release an established resource manager context, use **CasReleaseContext**.

## CasForgetCardType()

The **CasForgetCardType** function removes an introduced smart card from the smart card subsystem.

```
LONG CasForgetCardType(  
    IN SCARDCONTEXT hContext,  
    IN LPCTSTR szCardName  
);
```

### Parameters

*hContext*

Supplies the handle that identifies the resource manager context. The resource manager context is set by a previous call to **CasEstablishContext**.

*szCardName*

Supplies the friendly name of the card to be removed from the smart card database.

### Return Values

If the function...	The return value is...
Succeeds	SCARD_S_SUCCESS.
Fails	An error code (see <b>Error Codes</b> for a list of all error codes).

### Remarks

**CasForgetCardType** is a database management function.

## CasForgetReader()

The **CasForgetReader** function removes a previously introduced reader from control by the smart card subsystem. It is removed from the smart card database, including from any reader group that it may have been added to.

```
LONG CasForgetReader(  
    IN SCARDCONTEXT hContext,  
    IN LPCTSTR szReaderName  
);
```

### Parameters

*hContext*

Supplies the handle that identifies the resource manager context. The resource manager context is set by a previous call to **CasEstablishContext**.

*szReaderName*

Supplies the friendly name of the reader to be removed from the smart card database.

### Return Values

If the function...	The return value is...
Succeeds	SCARD_S_SUCCESS.
Fails	An error code (see <b>Error Codes</b> for a list of all error codes).

### Remarks

If the specified reader is the last member of a reader group, the reader group is automatically removed as well.

**CasForgetReader** is a database management function.

## CasForgetReaderGroup()

The **CasForgetReaderGroup** function removes a previously introduced smart card reader group from the smart card subsystem. Although this function automatically clears all readers from the group, it does not affect the existence of the individual readers in the database.

```
LONG CasForgetReaderGroup(  
    IN SCARDCONTEXT hContext,  
    IN LPCTSTR szGroupName  
);
```

### Parameters

*hContext*

Supplies the handle that identifies the resource manager context. The resource manager context is set by a previous call to **CasEstablishContext**.

*szGroupName*

Supplies the friendly name of the reader group to be removed. System-defined reader groups cannot be removed from the database.

### Return Values

If the function...	The return value is...
Succeeds	SCARD_S_SUCCESS.
Fails	An error code (see <b>Error Codes</b> for a list of all error codes).

### Remarks

**CasForgetReaderGroup** is a database management function.

## CasFreeMemory()

The **CasFreeMemory** function releases memory that has been returned from the resource manager using the SCARD\_AUTOALLOCATE length designator.

```
LONG CasFreeMemory(  
    IN SCARDCONTEXT hContext,  
    IN LPVOID pvMem  
);
```

### Parameters

*hContext*

Supplies the handle that identifies the resource manager context returned from **CasEstablishContext**, or NULL if the creating function also specified NULL for its hContext.

*pvMem*

Supplies the memory block to be released.

### Return Values

If the function...	The return value is...
Succeeds	SCARD_S_SUCCESS.
Fails	An error code (see <b>Error Codes</b> for a list of all error codes).

## CasGetAttrib()

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

```
LONG CasGetAttrib(  
    IN SCARDHANDLE hCard,  
    IN DWORD dwAttrId,  
    OUT LPBYTE pbAttr,  
    IN OUT LPDWORD pcbAttrLen  
);
```

### Parameters

*hCard*

Supplies the reference value returned from **CasConnect**.

*dwAttrId*

Supplies the identifier for the attribute to get.

*pbAttr*

Points to a buffer that receives the attribute whose ID is supplied in *dwAttrId*. If this value is NULL, **CasGetAttrib** ignores the buffer length supplied in *pcbAttrLen*, writes the length of the buffer that would have been returned if this parameter had not been NULL to *pcbAttrLen*, and returns a success code.

*pcbAttrLen*

Supplies the length of the *pbAttr* buffer in bytes, and receives the actual length of the received attribute.

### Return Values

If the function...	The return value is...
Succeeds	SCARD_S_SUCCESS.
Fails	An error code (see <b>Error Codes</b> for a list of all error codes).

### Remarks

**CasGetAttrib** is a direct card access function.

## CasGetCardTypeProviderName()

The **CasGetCardTypeProviderName** function returns the name of the module (dynamic link library) containing the provider for a given card name and provider type.

```
LONG CasGetCardTypeProviderName(
  IN SCARDCONTEXT hContext,
  IN LPCTSTR szCardName,
  IN DWORD dwProviderId,
  OUT LPTSTR szProvider,
  IN OUT LPDWORD pcchProvider
);
```

### Parameters

#### *hContext*

Supplies the handle that identifies the resource manager context. The resource manager context can be set by a previous call to **CasEstablishContext**. This value can be NULL if the call to **CasGetCardTypeProviderName** is not directed to a specific context.

#### *szCardName*

Supplies the name of the card type with which this provider name is associated.

#### *dwProviderId*

Supplies the identifier for the provider associated with this card type. The value provided for *dwProviderId* can be one of the following.

<b>Value</b>	<b>Action</b>
SCARD_PROVIDER_PRIMARY	The function retrieves the name of the primary smart card service provider as a GUID string.
SCARD_PROVIDER_CSP	The function retrieves the name of the cryptographic service provider.

#### *szProvider*

String variable which will contain the retrieved provider name upon successful completion of this function.

#### *pcchProvider*

Pointer to DWORD value. On input, *pcchProvider* supplies the length of the *szProvider* buffer in characters. If this value is SCARD\_AUTOALLOCATE, then *szProvider* is converted to a pointer to a string pointer and receives the address of a block of memory containing the string. This block of memory must be deallocated by calling **CasFreeMemory**.

On output, this value represents the actual number of characters, including the null terminator, in the *szProvider* variable.

### Return Values

<b>If the function...</b>	<b>The return value is...</b>
Succeeds	SCARD_S_SUCCESS.
Fails	An error code (see <b>Error Codes</b> for a list of all error codes).

### Remarks

Upon successful completion of this function, the value in *szProvider* can be used as the third parameter in a call to **CryptAcquireContext**.

### Example Code

```
CASCONTEXT hContext = NULL;

LPTSTR szProvider = NULL;
LPTSTR szCardName = _T("WindowsCard");

DWORD chProvider = SCARD_AUTO_ALLOCATE;

LONG IReturn = SCARD_S_SUCCESS;

// Establish a smart card resource context.
IReturn = CasEstablishContext(SCARD_SCOPE_USER,
                             NULL,
                             NULL,
                             &hContext);

if (SCARD_S_SUCCESS == IReturn)
{
    // Retrieve the provider name.
    IReturn = CasGetCardTypeProviderName(hContext,
                                         szCardName,
                                         SCARD_PROVIDER_CSP,
                                         (LPSTR)&szProvider,
                                         &chProvider);
}

if (SCARD_S_SUCCESS == IReturn)
{
    BOOL fSts = TRUE;
    HCRYPTPROV hProv = NULL;

    // Acquire a Cryptographic operation context.
    fSts = CryptAcquireContext(&hProv,
                              NULL,
                              szProvider,
                              PROV_RSA_FULL,
                              0);

    // Perform Cryptographic operations with smart card ...

    // Free memory allocated by CasGetCardTypeProviderName
    IReturn = CasFreeMemory(hContext, szProvider);
}
}
```

## CasGetProviderId()

The **CasGetProviderId** function returns the identifier (GUID) of the primary service provider for a given card.

The caller supplies the name of a smart card (previously introduced to the system) and receives the registered identifier of the primary service provider GUID, if one exists.

```
LONG CasGetProviderId(  
IN SCARDCONTEXT hContext,  
IN LPCTSTR szCard,  
OUT LPGUID pguidProviderId  
);
```

## Parameters

*hContext*

Supplies the handle that identifies the resource manager context for the query. The resource manager context can be set by a previous call to **CasEstablishContext**, or set to NULL if the query is not directed towards a specific context.

*szCard*

Supplies the name of the card defined to the system.

*pguidProviderId*

Receives the identifier (GUID) of the primary service provider. This provider may be activated via COM, and will supply access to other services in the card.

## Return Values

If the function...	The return value is...
Succeeds	SCARD_S_SUCCESS.
Fails	An error code (see <b>Error Codes</b> for a list of all error codes).

## Remarks

**CasGetProviderId** is a database query function.

## CasGetStatusChange()

The **CasGetStatusChange** function blocks execution until the current availability of the cards in a specific set of readers changes.

The caller supplies a list of readers to be monitored by an SCARD\_READERSTATE array and the maximum amount of time (in milliseconds) that it is willing to wait for an action to occur on one of the listed readers. Note that **CasGetStatusChange** uses the user-supplied value in the dwCurrentState members of the rgReaderStates parameter as the definition of the current state of the readers. The function returns when there is a change in availability, having filled in the dwEventState members of the rgReaderStates parameter appropriately.

```
LONG CasGetStatusChange(  
    IN SCARDCONTEXT hContext,  
    IN DWORD dwTimeout,  
    IN OUT LPSCARD_READERSTATE rgReaderStates,  
    IN DWORD cReaders  
);
```

### Parameters

*hContext*

Supplies the handle that identifies the resource manager context. The resource manager context is set by a previous call to **CasEstablishContext**.

*dwTimeout*

Supplies the maximum amount of time (in milliseconds) to wait for an action. A value of zero implies a value of INFINITE, dwTimeout will never timeout.

*rgReaderStates*

Supplies an array of SCARD\_READERSTATE structures that specify the readers to watch, and receives the result.

*cReaders*

Supplies the number of elements in the rgReaderStates array.

### Return Values

If the function...	The return value is...
Succeeds	SCARD_S_SUCCESS.
Fails	An error code (see <b>Error Codes</b> for a list of all error codes).

### Remarks

**CasGetStatusChange** is a smart card tracking function.

## CasIntroduceCardType()

The **CasIntroduceCardType** function introduces a smart card to the smart card subsystem (for the active user) by adding it to the smart card database.

```

LONG CasIntroduceCardType(
    IN SCARDCONTEXT hContext,
    IN LPCTSTR szCardName,
    IN LPGUID pguidPrimaryProvider,
    IN LPGUID rgguidInterfaces,
    IN DWORD dwInterfaceCount,
    IN LPCBYTE pbAtr,
    IN LPCBYTE pbAtrMask,
    IN DWORD cbAtrLen
);

```

## Parameters

### *hContext*

Supplies the handle that identifies the resource manager context. The resource manager context is set by a previous call to **CasEstablishContext**.

### *szCardName*

Supplies the name by which the user can recognize the card.

### *pguidPrimaryProvider*

Points to the identifier (GUID) for the smart card's primary service provider.

### *rgguidInterfaces*

Supplies an array of identifiers (GUIDs) that identify the interfaces supported by the smart card.

### *dwInterfaceCount*

Supplies the number of identifiers in the *rgguidInterfaces* array.

### *pbAtr*

Supplies an ATR string that can be used for matching purposes when querying the smart card database (see **CasListCards**). The length of this string is determined by normal ATR parsing.

### *pbAtrMask*

Supplies an optional bitmask to use when comparing the ATRs of smart cards to the ATR supplied in *pbAtr*. If this value is non-NULL, it must point to a string of bytes the same length as the ATR string supplied in *pbAtr*. When a given ATR string 'A' is compared to the ATR supplied in *pbAtr*, it matches if and only if  $A \& M = pbAtr$ , where M is the supplied mask, and & represents bitwise logical AND.

### *cbAtrLen*

Supplies the length of the ATR and optional ATR Mask. If this value is zero, then the length of the ATR is determined by normal ATR parsing. This value cannot be zero if a *pbAtrMask* value is supplied.

## Return Values

If the function...	The return value is...
Succeeds	SCARD_S_SUCCESS.
Fails	An error code (see <b>Error Codes</b> for a list of all error codes).

## Remarks

**CasIntroduceCardType** is a database management function. To remove a smart card, use **CasForgetCardType**.

## CasIntroduceReader()

The **CasIntroduceReader** function introduces a new name for an existing smart card reader.

Note Smart card readers are automatically introduced to the system; a smart card reader vendor's setup program can also introduce a smart card reader to the system.

```
LONG CasIntroduceReader(  
    IN SCARDCONTEXT hContext,  
    IN LPCTSTR szReaderName,  
    IN LPCTSTR szDeviceName  
);
```

## Parameters

*hContext*

Supplies the handle that identifies the resource manager context. The resource manager context is set by a previous call to **CasEstablishContext**.

*szReaderName*

Supplies the friendly name to be assigned to the reader.

*szDeviceName*

Supplies the system name of the smart card reader.

## Return Values

If the function...	The return value is...
Succeeds	SCARD_S_SUCCESS.
Fails	An error code (see <b>Error Codes</b> for a list of all error codes).

## Remarks

All readers installed on the system are automatically introduced by their system name. Typically, **CasIntroduceReader** is called only to change the name of an existing reader.

**CasIntroduceReader** is a database management function. For a description of other database management functions, see Smart Card Database Management Functions.

To remove a reader, use **CasForgetReader**.

## CasIntroduceReaderGroup()

The **CasIntroduceReaderGroup** function introduces a reader group to the smart card subsystem. However, the reader group is not created until the group is specified when adding a reader to the smart card database.

```
LONG CasIntroduceReaderGroup(  
    IN SCARDCONTEXT hContext,  
    IN LPCTSTR szGroupName  
);
```

### Parameters

*hContext*

Supplies the handle that identifies the resource manager context. The resource manager context is set by a previous call to **CasEstablishContext**.

*szGroupName*

Supplies the friendly name to be assigned to the new reader group.

### Return Values

If the function...	The return value is...
Succeeds	SCARD_S_SUCCESS.
Fails	An error code (see <b>Error Codes</b> for a list of all error codes).

### Remarks

**CasIntroduceReaderGroup** is provided for PC/SC specification compatibility. Reader groups are not stored until a reader is added to the group.

**CasIntroduceReaderGroup** is a database management function.

To remove a reader group, use **CasForgetReaderGroup**.

## CasListCards()

The **CasListCards** function searches the smart card database and provides a list of named cards previously introduced to the system by the user.

The caller specifies an ATR string, a set of interface identifiers (GUIDs), or both. If both an ATR string and an identifier array are supplied, the cards returned will match the ATR string supplied and support the interfaces specified.

```
LONG CasListCards(
    IN SCARDCONTEXT hContext,
    IN LPCBYTE pbAtr,
    IN LPCGUID rgguidInterfaces,
    IN DWORD cguidInterfaceCount,
    OUT LPTSTR mszCards,
    IN OUT LPDWORD pcchCards
);
```

## Parameters

### *hContext*

Supplies the handle that identifies the resource manager context for the query. The resource manager context can be set by a previous call to **CasEstablishContext**, or set to NULL if the query is not directed towards a specific context.

### *pbAtr*

Supplies the address of an ATR string to compare to known cards, or NULL if no ATR matching is to be performed.

### *rgguidInterfaces*

Supplies an array of identifiers (GUIDs), or NULL if no interface matching is to be performed. When an array is supplied, a card name will be returned only if all the specified identifiers are supported by the card.

### *cguidInterfaceCount*

Supplies the number of entries in the rgguidInterfaces array. If rgguidInterfaces is NULL, then this value is ignored.

### *mszCards*

Receives a multi-string that lists the smart cards found. If this value is NULL, **CasListCards** ignores the buffer length supplied in pcchCards, returning the length of the buffer that would have been returned if this parameter had not been NULL to pcchCards and a success code.

### *pcchCards*

Supplies the length of the mszCards buffer in characters, and receives the actual length of the multi-string structure, including all trailing Null characters. If the buffer length is specified as SCARD\_AUTOALLOCATE, then mszCards is converted to a pointer to a string pointer, and receives the address of a block of memory containing the multi-string structure. This block of memory must be deallocated with **CasFreeMemory**.

## Return Values

If the function...	The return value is...
Succeeds	SCARD_S_SUCCESS.
Fails	An error code (see <b>Error Codes</b> for a list of all error codes).

**Remarks**

To return all smart cards introduced to the subsystem, set pbAtr and rguidInterfaces to NULL.

**CasListCards** is a database query function.

## CasListInterfaces()

The **CasListInterfaces** function provides a list of interfaces supplied by a given card.

The caller supplies the name of a smart card previously introduced to the subsystem, and receives the list of interfaces supported by the card.

```
LONG CasListInterfaces(  
    IN SCARDCONTEXT hContext,  
    IN LPCTSTR szCard,  
    OUT LPGUID pguidInterfaces,  
    IN OUT LPDWORD pcguidInterfaces  
);
```

## Parameters

### *hContext*

Supplies the handle that identifies the resource manager context for the query. The resource manager context can be set by a previous call to **CasEstablishContext**, or set to NULL if the query is not directed towards a specific context.

### *szCard*

Supplies the name of the smart card already introduced to the smart card subsystem.

### *pguidInterfaces*

Receives an array of interface identifiers (GUIDs) that indicate the interfaces supported by the smart card. If this value is NULL, **CasListInterfaces** ignores the array length supplied in pcguidInterfaces, returning the size of the array that would have been returned if this parameter had not been NULL to pcguidInterfaces and a success code.

### *pcguidInterfaces*

Supplies the size of the pguidInterfaces array, and receives the actual size of the returned array. If the array size is specified as SCARD\_AUTOALLOCATE, then pguidInterfaces is converted to a pointer to a GUID pointer, and receives the address of a block of memory containing the array. This block of memory must be deallocated with **CasFreeMemory**.

## Return Values

If the function...	The return value is...
Succeeds	SCARD_S_SUCCESS.
Fails	An error code (see <b>Error Codes</b> for a list of all error codes).

## Remarks

**CasListInterfaces** is a database query function.

## CasListReaderGroups()

The **CasListReaderGroups** function provides the list of reader groups that have previously been introduced to the system.

```
LONG CasListReaderGroups(
    IN SCARDCONTEXT hContext,
    OUT LPTSTR mszGroups,
    IN OUT LPDWORD pcchGroups
);
```

### Parameters

#### *hContext*

Supplies the handle that identifies the resource manager context for the query. The resource manager context can be set by a previous call to **CasEstablishContext**, or set to NULL if the query is not directed towards a specific context.

#### *mszGroups*

Receives a multi-string that lists the reader groups defined to the system and available to the current user on the current terminal. If this value is NULL, **CasListReaderGroups** ignores the buffer length supplied in *pcchGroups*, writes the length of the buffer that would have been returned if this parameter had not been NULL to *pcchGroups*, and returns a success code.

#### *pcchGroups*

Supplies the length of the *mszGroups* buffer in characters, and receives the actual length of the multi-string structure, including all trailing Null characters. If the buffer length is specified as SCARD\_AUTOALLOCATE, then *mszGroups* is converted to a pointer to a string pointer, and receives the address of a block of memory containing the multi-string structure. This block of memory must be deallocated with **CasFreeMemory**.

### Return Values

If the function...	The return value is...
Succeeds	SCARD_S_SUCCESS.
Fails	An error code (see <b>Error Codes</b> for a list of all error codes).

### Remarks

A group is returned only if it contains at least one reader. This includes the group Scard\$DefaultReaders. The group Scard\$AllReaders cannot be returned, since it only exists implicitly.

**CasListReaderGroups** is a database query function.

## CasListReaders()

The **CasListReaders** function provides the list of readers within a set of named reader groups, eliminating duplicates.

The caller supplies a list reader groups, and receives the list of readers within the named groups. Unrecognized group names are ignored.

```
LONG CasListReaders(
IN SCARDCONTEXT hContext,
IN LPCTSTR mszGroups,
OUT LPTSTR mszReaders,
IN OUT LPDWORD pcchReaders
);
```

## Parameters

### *hContext*

Supplies the handle that identifies the resource manager context for the query. The resource manager context can be set by a previous call to **CasEstablishContext**, or set to NULL if the query is not directed towards a specific context.

### *mszGroups*

Supplies the names of the reader groups defined to the system, as a multi-string. Use a NULL value to list all readers in the system (that is, the SCard\$AllReaders group).

### *mszReaders*

Receives a multi-string that list the card readers within the supplied reader groups. If this value is NULL, **CasListReaders** ignores the buffer length supplied in *pcchReaders*, writes the length of the buffer that would have been returned if this parameter had not been NULL to *pcchReaders*, and returns a success code.

### *pcchReaders*

Supplies the length of the *mszReaders* buffer in characters, and receives the actual length of the multi-string structure, including all trailing Null characters. If the buffer length is specified as SCARD\_AUTOALLOCATE, then *mszReaders* is converted to a pointer to a string pointer, and receives the address of a block of memory containing the multi-string structure. This block of memory must be deallocated with **CasFreeMemory**.

## Return Values

If the function...	The return value is...
Succeeds	SCARD_S_SUCCESS.
Fails	An error code (see <b>Error Codes</b> for a list of all error codes).

## Remarks

**CasListReaders** is a database query function.

## CasLocateCards()

The **CasLocateCards** function searches the readers listed in the `rgReaderStates` parameter for a card with an ATR string that matches one of the card names specified in `mszCards`, returning immediately with the result.

```
LONG CasLocateCards(
IN SCARDCONTEXT hContext,
IN LPCTSTR mszCards,
IN OUT LPSCARD_READERSTATE rgReaderStates,
IN DWORD cReaders
);
```

## Parameters

### *hContext*

Supplies the handle that identifies the resource manager context. The resource manager context is set by a previous call to **CasEstablishContext**.

### *mszCards*

Supplies a multi-string that contains the names of the cards to search for.

### *rgReaderStates*

Supplies an array of SCARD\_READERSTATE structures that specify the readers to search, and receives the result.

### *cReaders*

Supplies the number of elements in the `rgReaderStates` array.

## Return Values

If the function...	The return value is...
Succeeds	SCARD_S_SUCCESS.
Fails	An error code (see <b>Error Codes</b> for a list of all error codes).

## Remarks

This service is especially useful when used in conjunction with **CasGetStatusChange**. If no matching cards are found by means of **CasLocateCards**, the calling application may use **CasGetStatusChange** to wait for card availability changes.

**CasLocateCards** is a smart card tracking function.

## CasReconnect()

The **CasReconnect** function reestablishes an existing connection between the calling application and a smart card. This function moves a card handle from direct access to general access, or acknowledges and clears an error condition that is preventing further access to the card.

```
LONG CasReconnect(
IN SCARDHANDLE hCard,
IN DWORD dwShareMode,
IN DWORD dwPreferredProtocols,
IN DWORD dwInitialization,
OUT LPDWORD pdwActiveProtocol
);
```

### Parameters

*hCard*

Supplies the reference value obtained from a previous call to **CasConnect**.

*dwShareMode*

Supplies a flag that indicates whether other applications may form connections to this card. Possible values are:

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

*dwPreferredProtocols*

Supplies a bit mask of acceptable protocols for this connection. Possible values, which may be combined with the OR operation, are:

Value	Meaning
SCARD_PROTOCOL_T0	T=0 is an acceptable protocol.
SCARD_PROTOCOL_T1	T=1 is an acceptable protocol.

*dwInitialization*

Indicates what kind of initialization should be performed on the card. Possible values are:

Value	Meaning
SCARD_LEAVE_CARD	Don't 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).

*pdwActiveProtocol*

Receives a flag that indicates the established active protocol. Possible values are:

Value	Meaning
SCARD_PROTOCOL_T0	T=0 is the active protocol.
SCARD_PROTOCOL_T1	T=1 is the active protocol.

### Return Values

If the function...	The return value is...
Succeeds	SCARD_S_SUCCESS.
Fails	An error code (see <b>Error Codes</b> for a list of all error codes).

Remarks

**CasReconnect** is a smart card and reader access function.

### CasReleaseContext()

The **CasReleaseContext** function closes an established resource manager context, freeing any resources allocated under that context, including SCARDHANDLE objects and memory allocated using the SCARD\_AUTOALLOCATE length designator.

```
LONG CasReleaseContext(  
    IN SCARDCONTEXT hContext  
);
```

### Parameters

*hContext*

Supplies the handle that identifies the resource manager context. The resource manager context is set by a previous call to **CasEstablishContext**.

### Return Values

<b>If the function...</b>	<b>The return value is...</b>
Succeeds	SCARD_S_SUCCESS.
Fails	An error code (see <b>Error Codes</b> for a list of all error codes).

## CasRemoveReaderFromGroup()

The **CasRemoveReaderFromGroup** function removes a reader from an existing reader group. This function has no affect on the reader or reader group.

```
LONG CasRemoveReaderFromGroup(  
    IN SCARDCONTEXT hContext,  
    IN LPCTSTR szReaderName,  
    IN LPCTSTR szGroupName  
);
```

### Parameters

*hContext*

Supplies the handle that identifies the resource manager context. The resource manager context is set by a previous call to **CasEstablishContext**.

*szReaderName*

Supplies the friendly name of the reader to be removed.

*szGroupName*

Supplies the friendly name of the group from which the reader should be removed.

### Return Values

If the function...	The return value is...
Succeeds	SCARD_S_SUCCESS.
Fails	An error code (see <b>Error Codes</b> for a list of all error codes).

### Remarks

When the last reader is removed from a group, the group is automatically forgotten.

**CasRemoveReaderFromGroup** is a database management function.

To add a reader to a reader group, use **CasAddReaderToGroup**.

## CasSetAttrib()

The **CasSetAttrib** function sets the given reader attribute for the given handle. It does not affect the state of the reader, reader driver, or smart card. Not all attributes are supported by all readers (nor can they be set at all times) as many of the attributes are under direct control of the transport protocol.

```
LONG CasSetAttrib(  
  IN SCARDHANDLE hCard,  
  IN dwAttrId,  
  IN LPCBYTE pbAttr,  
  IN DWORD cbAttrLen  
);
```

## Parameters

*hCard*

Supplies the reference value returned from **CasConnect**.

*dwAttrId*

Supplies the identifier for the attribute to set.

*pbAttr*

Points to a buffer that supplies the attribute whose ID is supplied in *dwAttrId*.

*cbAttrLen*

Supplies the length (in bytes) of the attribute value in the *pbAttr* buffer.

## Return Values

If the function...	The return value is...
Succeeds	SCARD_S_SUCCESS.
Fails	An error code (see <b>Error Codes</b> for a list of all error codes).

## Remarks

**CasSetAttrib** is a direct card access function.

## CasStatus()

The **CasStatus** function provides the current status of a smart card in a reader. You can call it any time after a successful call to **CasConnect** and before a successful call to **CasDisconnect**. It does not affect the state of the reader or reader driver.

```
LONG CasStatus(
  IN SCARDHANDLE hCard,
  OUT LPTSTR  szReaderName,
  IN OUT LPDWORD pcchReaderLen,
  OUT LPDWORD pdwState,
  OUT LPDWORD pdwProtocol,
  OUT LPBYTE pbAtr,
  OUT LPDWORD pcbAtrLen
);
```

## Parameters

*hCard*

This is the reference value returned from CasConnect.

*szReaderName*

Receives a list of friendly names (multi-string) by which the currently connected reader is known.

*pcchReaderLen*

On input, supplies the length of the szReaderName buffer

On output, receives the actual length (in characters) of the reader name list, including the trailing NULL character.

*pdwState*

Receives the 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.

*pdwProtocol*

Receives the current protocol, if any. The returned value is meaningful only if the returned value of pdwState is SCARD\_SPECIFICMODE. Possible returned values are the following:

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.

*pbAtr*

Points to a 32-byte buffer that receives the ATR string from the currently inserted card, if available.

*pcbAtrLen*

Points to a DWORD to receive the number of bytes in the ATR string (32 bytes maximum).

### Return Values

If the function...	The return value is...
Succeeds	SCARD_S_SUCCESS.
Fails	An error code (see <b>Error Codes</b> for a list of all error codes).

### Remarks

**CasStatus** is a smart card and reader access function.

## CasTransmit()

The **CasTransmit** function sends a service request to the smart card, and expects to receive data back from the card.

```
LONG CasTransmit(
    IN SCARDHANDLE hCard,
    IN LPCSCARD_IO_REQUEST pioSendPci,
    IN LPCBYTE pbSendBuffer,
    IN DWORD cbSendLength,
    IN OUT LPSCARD_IO_REQUEST pioRecvPci,
    OUT LPBYTE pbRecvBuffer,
    IN OUT LPDWORD pcbRecvLength
);
```

## Parameters

*hCard*

Supplies the reference value returned from **CasConnect**.

*pioSendPci*

Pointer to the protocol header structure for the instruction. This buffer is in the format of an SCARD\_IO\_REQUEST structure, followed by the specific protocol control information (PCI).

For the T=0, T=1, and Raw protocols, the PCI structure is constant. The smart card subsystem supplies a global T=0, T=1, or Raw PCI structure, which you can reference by using the symbols SCARD\_PCI\_T0, SCARD\_PCI\_T1, and SCARD\_PCI\_RAW respectively.

*pbSendBuffer*

Pointer to the actual data to be written to the card.

T=0 Note For T=0, the data parameters are placed into the pbSendBuffer according to the following structure:

```
struct {
    BYTE
    bCla, // The instruction class
    bIns, // The instruction code
    bP1,  // Parameter to the instruction
    bP2,  // Parameter to the instruction
    bP3;  // Size of I/O Transfer
} CmdBytes;
```

### Members:

*bCla*

The T=0 instruction class

*bIns*

An instruction code in the T=0 instruction class

*bP1, bP2*

Reference codes completing the instruction code

*bP3*

The number of data bytes which are to be transmitted during the command, per ISO 7816-4, Section 8.2.1.

The data sent to the card should immediately follow the send buffer. In the special case where no data is sent to the card and no data is expected in return, bP3 is not sent.

*cbSendLength*

Supplies the length (in bytes) of the pbSendBuffer parameter.

T=0 Note For T=0, in the special case where no data is sent to the card and no data expected in return, this length must reflect that the bP3 member is not being sent: the length should be `sizeof(CmdBytes) – sizeof(BYTE)`.

*pioRecvPci*

Pointer to the protocol header structure for the instruction, followed by a buffer in which to receive any returned protocol control information (PCI) specific to the protocol in use. This parameter may be NULL if no returned PCI is desired.

*pbRecvBuffer*

Pointer to any data returned from the card.

T=0 Note For T=0, the data is immediately followed by the SW1 and SW2 status bytes. If no data is returned from the card, then this buffer will only contain the SW1 and SW2 status bytes.

*pcbRecvLength*

Supplies the length of the *pbRecvBuffer* parameter (in bytes) and receives the actual number of bytes received from the smart card.

T=0 Note For T=0, the receive buffer must be at least two bytes long, in order to receive the SW1 and SW2 status bytes.

## Return Values

If the function...	The return value is...
Succeeds	SCARD_S_SUCCESS.
Fails	An error code (see <b>Error Codes</b> for a list of all error codes).

## Remarks

**CasTransmit** is a smart card and reader access function.

### T=0 Protocol Remarks

For the T=0 protocol, the data received back are the SW1 and SW2 status codes, possibly preceded by response data. The following paragraphs provide information on the send and receive buffers used to transfer data and issue a command.

#### Sending Data to the Card

To send  $n > 0$  bytes of data to the card, where  $n > 0$ , the send and receive buffers must be formatted as follows.

The first four bytes of the *pbSendBuffer* buffer contain the CLA, INS, P1, and P2 values for the T=0 operation. The fifth byte shall be set to  $n$ : the size (in bytes) of the data to be transferred to the card. The next  $n$  bytes shall contain the data to be sent to the card.

The *cbSendLength* parameter shall be set to the size of the T=0 header information (CLA, INS, P1 and P2) plus a byte containing the length of the data to be transferred ( $n$ ), plus the size of data to be sent. In this example, this is  $n+5$ .

The *pbRecvBuffer* will receive the SW1 and SW2 status codes from the operation.

The *pcbRecvLength* should be at least 2, and will be set to 2 upon return.

#### Obtaining Data from the Card

To receive  $n > 0$  bytes of data from the card, the send and receive buffers must be formatted as follows.

The first four bytes of the *pbSendBuffer* buffer contain the CLA, INS, P1, and P2 values for the T=0 operation. The fifth byte shall be set to  $n$ : the size (in bytes) of the data to be

transferred from the card. If 256 bytes are to be transferred from the card, then this byte shall be set to zero.

The `cbSendLength` parameter shall be set to 5, the size of the T=0 header information.

The `pbRecvBuffer` will receive the data returned from the card, immediately followed by the SW1 and SW2 status codes from the operation.

The `pcbRecvLength` should be at least  $n+2$ , and will be set to  $n+2$  upon return.

#### Issuing a Command Without Exchanging Data

To issue a command to the card that does not involve the exchange of data (either sent or received), the send and receive buffers must be formatted as follows.

The `pbSendBuffer` buffer shall contain the CLA, INS, P1, and P2 values for the T=0 operation. The P3 value is not sent. (This is to differentiate the header from the case where 256 bytes are expected to be returned.)

The `cbSendLength` parameter shall be set to 4, the size of the T=0 header information (CLA, INS, P1, and P2).

The `pbRecvBuffer` will receive the SW1 and SW2 status codes from the operation.

The `pcbRecvLength` should be at least 2, and will be set to 2 upon return.

**CasUIDlgSelectCard()**

The **CasUIDlgSelectCard** function displays the smart card "select card" dialog.

```
LONG (
    IN LPOPCARDNAME_EX pDlgStruc
);
```

**Parameters**

*pDlgStruc*

Points to the OPENCARDNAME\_EX structure for the "select card" dialog.

**Remarks**

The **CasUIDlgSelectCard** function provides a method for connecting to a specific smart card. When called, this function performs a search for appropriate smart cards matching the OPENCARD\_SEARCH\_CRITERIA member of \*pDlgStruc. Depending of the dwFlags member of \*pDlgStruc, this function takes the following actions.

<b>Value for dwFlag</b>	<b>Action</b>
SC_DLG_FORCE_UI	Connects to the card selected by the user from the smart card 'Select Card' dialog.
SC_DLG_MINIMAL_UI	Selects the smart card if only one meets the criteria, or returns information about the user's selection if more than one smart card meets the criteria.
SC_DLG_NO_UI	Selects the first available card.

This function replaces GetOpenCardName. GetOpenCardName is maintained for backward compatibility with version 1.0 of the MS Smart Card Base Components.

**Return Values**

<b>If the function...</b>	<b>The return value is...</b>
Succeeds	SCARD_S_SUCCESS.
Fails	An error code (see <b>Error Codes</b> for a list of all error codes).

## 4. Smart Card Error Codes

This section describes the primary error codes returned by smart card functions.

<b>Error</b>	<b>Definition</b>
SCARD_E_CANCELLED	The action was canceled by an <b>CasCancel</b> request.
SCARD_E_CANT_DISPOSE	The system could not dispose of the media in the requested manner.
SCARD_E_CARD_UNSUPPORTED	The smart card does not meet minimal requirements for support.
SCARD_E_DUPLICATE_READER	The reader driver didn't produce a unique reader name.
SCARD_E_INSUFFICIENT_BUFFER	The data buffer for returned data is too small for the returned data.
SCARD_E_INVALID_ATR	An ATR string obtained from the registry is not a valid ATR string.
SCARD_E_INVALID_HANDLE	The supplied handle was invalid.
SCARD_E_INVALID_PARAMETER	One or more of the supplied parameters could not be properly interpreted.
SCARD_E_INVALID_TARGET	Registry startup information is missing or invalid.
SCARD_E_INVALID_VALUE	One or more of the supplied parameter values could not be properly interpreted.
SCARD_E_NOT_READY	The reader or card is not ready to accept commands.
SCARD_E_NOT_TRANSACTED	An attempt was made to end a non-existent transaction.
SCARD_E_NO_MEMORY	Not enough memory available to complete this command.
SCARD_E_NO_SERVICE	The smart card resource manager is not running.
SCARD_E_NO_SMARTCARD	The operation requires a smart card, but no smart card is currently in the device.
SCARD_E_PCI_TOO_SMALL	The PCI receive buffer was too small.
SCARD_E_PROTO_MISMATCH	The requested protocols are incompatible with the protocol currently in use with the card.
SCARD_E_READER_UNAVAILABLE	The specified reader is not currently available for use.
SCARD_E_READER_UNSUPPORTED	The reader driver does not meet minimal requirements for support.
SCARD_E_SERVICE_STOPPED	The smart card resource manager has shut down.
SCARD_E_SHARING_VIOLATION	The smart card cannot be accessed because of other outstanding connections.
SCARD_E_SYSTEM_CANCELLED	The action was canceled by the system, presumably to log off or shut down.
SCARD_E_TIMEOUT	The user-specified timeout value has

	expired.
SCARD_E_UNKNOWN_CARD	The specified smart card name is not recognized.
SCARD_E_UNKNOWN_READER	The specified reader name is not recognized.
SCARD_F_COMM_ERROR	An internal communications error has been detected.
SCARD_F_INTERNAL_ERROR	An internal consistency check failed.
SCARD_F_UNKNOWN_ERROR	An internal error has been detected, but the source is unknown.
SCARD_F_WAITED_TOO_LONG	An internal consistency timer has expired.
SCARD_S_SUCCESS	No error was encountered.
SCARD_W_REMOVED_CARD	The smart card has been removed, so that further communication is not possible.
SCARD_W_RESET_CARD	The smart card has been reset, so any shared state information is invalid.
SCARD_W_UNPOWERED_CARD	Power has been removed from the smart card, so that further communication is not possible.
SCARD_W_UNRESPONSIVE_CARD	The smart card is not responding to a reset.
SCARD_W_UNSUPPORTED_CARD	The reader cannot communicate with the card, due to ATR string configuration conflicts.