This document is provided on an AS-IS basis. Neither the authors nor members of the MUSCLE group are responsible for any mishaps, misuse, or loss caused by the use of this document and specification. This document can be freely distributed. Modifications must be approved by the authors before redistribution. Copyright (C) 2000 – 2002 David Corcoran
OVERVIEW OF MUSCLECARD FRAMEWORK

This document describes the client side API for the MUSCLE Framework. This API provides a near direct mapping of the function available on the MUSCLE Card Edge Applet. Some functions have been provided as helper functions to ease commonly known tasks which might require one or many of the other functions to complete.

The MUSCLE Framework provides cross-compatibility across multiple vendor smartcards and its client side API provides cross-compatibility across many platforms. MUSCLE has been designed to work on most platforms including: Unix (Linux, Solaris, *BSD, Tru64, HP-UX), Macintosh (OS X), and Windows (2000, 98, CE) based platforms. By being built on PC/SC and providing simple, clean functionality for multiple platforms it is possible to develop cross-platform applications which make use of cryptographic smartcards and tokens, independant of both the card/crypto token manufacturer, operating system, and platform.

This API is used to abstract many types of tokens through a token loading mechanism which dynamically loads tokens automatically by identifying them and loading their associated bundle or plug-in. This allows applications to make use of cryptographic tokens in a manner which removes them from statically binding their application to specific devices.

This document is meant to be used with the MUSCLE Cryptographic Card Edge Definition document which further describes some of the data types used in this API.

Applications will link to the MUSCLE library (-lpcsclite for gcc users, PCSC.framework for OS X users) and to the pcsc-lite library (non-OS X users). MuscleCard provides one header, musclecard.h, (PCSC/musclecard.h for OS X) which provides the following functions listed in the upcoming pages of this document.

For more information please contact: corcoran@linuxnet.com

ADDITIONAL REFERENCE

- MUSCLECARD Plug-In Developer's Manual
- MUSCLECARD Card Edge Applet Specification
- MUSCLE IFD Handler Developers Manual
## MUSCLECARD FUNCTIONS

<table>
<thead>
<tr>
<th>Function Name</th>
<th>Function Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>MSCListTokens</td>
<td>- List tokens available</td>
</tr>
<tr>
<td>MSCEstablishConnection</td>
<td>- Connects to a token</td>
</tr>
<tr>
<td>MSCReleaseConnection</td>
<td>- Releases a token</td>
</tr>
<tr>
<td>MSCWaitForTokenEvent</td>
<td>- Waits for token event</td>
</tr>
<tr>
<td>MSCCancelEventWait</td>
<td>- Cancels a pending MSCWaitForTokenEvent</td>
</tr>
<tr>
<td>MSCCallbackForTokenEvent</td>
<td>- Register a callback for token events</td>
</tr>
<tr>
<td>MSCCallbackCancelEvent</td>
<td>- Stops all event callback threads</td>
</tr>
<tr>
<td>MSCBeginTransaction</td>
<td>- Locks a transaction</td>
</tr>
<tr>
<td>MSCEndTransaction</td>
<td>- Unlocks a transaction</td>
</tr>
<tr>
<td>MSCGetStatus</td>
<td>- Gets card information</td>
</tr>
<tr>
<td>MSCGetCapabilities</td>
<td>- Gets the card’s capabilities</td>
</tr>
<tr>
<td>MSCExtendedFeature</td>
<td>- Vendor specific features</td>
</tr>
<tr>
<td>MSCGenerateKeys</td>
<td>- Generates keys</td>
</tr>
<tr>
<td>MSCImportKey</td>
<td>- Imports a key</td>
</tr>
<tr>
<td>MSCExportKey</td>
<td>- Exports a key</td>
</tr>
<tr>
<td>MSCComputeCrypt</td>
<td>- Performs crypto operation</td>
</tr>
<tr>
<td>MSCAuthenticate</td>
<td>- Authenticates host</td>
</tr>
<tr>
<td>MSCGetKeyAttributes</td>
<td>- Gets the attributes of a key</td>
</tr>
<tr>
<td>MSCListKeys</td>
<td>- Lists available keys</td>
</tr>
<tr>
<td>MSCCreatePIN</td>
<td>- Creates a PIN</td>
</tr>
<tr>
<td>MSCVerifyPIN</td>
<td>- Verifies a PIN</td>
</tr>
<tr>
<td>MSCChangePIN</td>
<td>- Changes a PIN</td>
</tr>
<tr>
<td>MSCUnblockPIN</td>
<td>- Unblocks a PIN</td>
</tr>
<tr>
<td>MSCListPINs</td>
<td>- Lists available PINs</td>
</tr>
<tr>
<td>MSCCreateObject</td>
<td>- Creates an object</td>
</tr>
<tr>
<td>MSCDeleteObject</td>
<td>- Deletes an object</td>
</tr>
<tr>
<td>MSCWriteObject</td>
<td>- Writes an object</td>
</tr>
<tr>
<td>MSCReadObject</td>
<td>- Reads an object</td>
</tr>
<tr>
<td>MSCReadAllocateObject</td>
<td>- Reads entire bulk object</td>
</tr>
<tr>
<td>MSCGetObjectAttributes</td>
<td>- Gets object information</td>
</tr>
<tr>
<td>MSCListObjects</td>
<td>- Lists available objects</td>
</tr>
<tr>
<td>MSCLogoutAll</td>
<td>- Logs out identities</td>
</tr>
<tr>
<td>MSCGetChallenge</td>
<td>- Gets random from card</td>
</tr>
</tbody>
</table>
### MUSCLECARD RETURN CODES

<table>
<thead>
<tr>
<th>Return Code</th>
<th>Return Code Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>MSC_SUCCESS</td>
<td>Successful</td>
</tr>
<tr>
<td>MSC_NO_MEMORY_LEFT</td>
<td>Not enough memory to perform operation</td>
</tr>
<tr>
<td>MSC_OPERATION_NOT_ALLOWED</td>
<td>Operation is not allowed</td>
</tr>
<tr>
<td>MSC_INCONSISTENT_STATUS</td>
<td>Operation inconsistent with current state</td>
</tr>
<tr>
<td>MSC_UNSUPPORTED_FEATURE</td>
<td>Feature not currently supported</td>
</tr>
<tr>
<td>MSC_OBJECT_NOT_FOUND</td>
<td>Object is not found</td>
</tr>
<tr>
<td>MSC_OBJECT_EXISTS</td>
<td>Object already exists</td>
</tr>
<tr>
<td>MSC_SEQUENCE_END</td>
<td>The sequence has ended</td>
</tr>
<tr>
<td>MSC_SIGNATURE_INVALID</td>
<td>Verify detected an invalid signature</td>
</tr>
<tr>
<td>MSC_IDENTITY_BLOCKED</td>
<td>Operation blocked</td>
</tr>
<tr>
<td>MSC_INCORRECT_ALG</td>
<td>Algorithm incorrect or not supported</td>
</tr>
<tr>
<td>MSC_UNAUTHORIZED</td>
<td>Not authorized to perform task</td>
</tr>
<tr>
<td>MSC_AUTH_FAILED</td>
<td>Authentication failed</td>
</tr>
<tr>
<td>MSC_INVALID_PARAMETER</td>
<td>Invalid parameter given</td>
</tr>
<tr>
<td>MSC_UNSPECIFIED_ERROR</td>
<td>Unspecified error</td>
</tr>
<tr>
<td>MSC_TRANSPORT_ERROR</td>
<td>Error in transport</td>
</tr>
<tr>
<td>MSC_INCORRECT_P1</td>
<td>Incorrect ISO P1 given</td>
</tr>
<tr>
<td>MSC_INCORRECT_P2</td>
<td>Incorrect ISO P2 given</td>
</tr>
<tr>
<td>MSC_INTERNAL_ERROR</td>
<td>An internal error has occurred</td>
</tr>
<tr>
<td>MSC_CANCELED</td>
<td>A blocking event was cancelled</td>
</tr>
<tr>
<td>MSC_INSUFFICIENT_BUFFER</td>
<td>The buffer provided is too short</td>
</tr>
<tr>
<td>MSC_UNRECOGNIZED_TOKEN</td>
<td>Chosen token not recognized</td>
</tr>
<tr>
<td>MSC_SERVICE_UNRESPONSIVE</td>
<td>Token services unavailable</td>
</tr>
<tr>
<td>MSC_TIMEOUT_OCCURRED</td>
<td>The action has timed out</td>
</tr>
<tr>
<td>MSC_TOKEN_REMOVED</td>
<td>The token was removed</td>
</tr>
<tr>
<td>MSC_TOKEN_RESET</td>
<td>The token was reset</td>
</tr>
<tr>
<td>MSC_TOKEN_INSERTED</td>
<td>The token was inserted</td>
</tr>
<tr>
<td>MSC_TOKEN_UNRESPONSIVE</td>
<td>The token is unresponsive</td>
</tr>
<tr>
<td>MSC_INVALID_HANDLE</td>
<td>The provided handle is invalid</td>
</tr>
<tr>
<td>MSC_SHARING_VIOLATION</td>
<td>The desired sharing is not available</td>
</tr>
</tbody>
</table>

An application can choose to get an English human readable string which describes the error condition which as occurred by using the following helper function:

```c
MSCString msc_error(MSCLong32 errorCode)
-returns a temporary character string of the explained error.
```
MUSCLECARD TYPES

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>MSC_RV</td>
<td>32 bit unsigned return for functions</td>
</tr>
<tr>
<td>MSCChar8;</td>
<td>08 bit signed</td>
</tr>
<tr>
<td>MSCPUChar8;</td>
<td>08 bit unsigned pointer</td>
</tr>
<tr>
<td>MSCPCUChar8;</td>
<td>08 bit constant unsigned pointer</td>
</tr>
<tr>
<td>MSCUChar8;</td>
<td>08 bit unsigned</td>
</tr>
<tr>
<td>MSCCString;</td>
<td>08 bit constant signed pointer</td>
</tr>
<tr>
<td>MSCString;</td>
<td>08 bit signed</td>
</tr>
<tr>
<td>MSCPUShort16;</td>
<td>16 bit unsigned pointer</td>
</tr>
<tr>
<td>MSCUShort16;</td>
<td>16 bit unsigned</td>
</tr>
<tr>
<td>MSCPShort16;</td>
<td>16 bit signed</td>
</tr>
<tr>
<td>MSCShort16;</td>
<td>16 bit signed</td>
</tr>
<tr>
<td>MSCPULong32;</td>
<td>32 bit unsigned pointer</td>
</tr>
<tr>
<td>MSCULong32;</td>
<td>32 bit unsigned</td>
</tr>
<tr>
<td>MSCPLong32;</td>
<td>32 bit signed</td>
</tr>
<tr>
<td>MSCPCVoid32;</td>
<td>32 bit constant void pointer</td>
</tr>
<tr>
<td>MSCPVoid32;</td>
<td>32 bit void pointer</td>
</tr>
</tbody>
</table>

MUSCLECARD STRUCTURES

The following structures are contained in the MuscleCard Framework. These structures can contain additional elements. The elements described here are ones which might be used by the application. Each structure is described by the following mechanism:

Structure Name
- Description

[ TYPE ELEMENT - Description (Read Only) TYPE ELEMENT - Description (Read/Write) ]

MSCTokenInfo *MSCLPTokenInfo
- This structure contains information about a particular token. It is used to retrieve information about a token and as a handle for connection and event functions.

[ MSCChar8[] tokenName - Friendlyname of the token MSCChar8[] slotName - Friendlyname of the slot MSCULong32 tokenState - Bimask state of the token ]
MSCTokenConnection, *MSCLPTokenConnection
- This structure is used as a handle to all functions after a connection is made to a token.

[ ]
MSCUChar8  pMac          - MAC cryptogram used for secure comm (RFU)
MSCULong32  macSize       - Size of the cryptogram
MSCTokenInfo  tokenInfo  - Token information for a particular connection

MSCStatusInfo, *MSCLPStatusInfo
- This structure is returned from MSCGetStatus which contains status information about the token. Capability information should be requested using MSCGetCapabilities.

[ ]
MSCUShort16  appVersion   - Application protocol version number
MSCUShort16  swVersion    - Software version number
MSCULong32  freeMemory   - Amount of free memory available
MSCULong32  totalMemory  - Total memory available
MSCUChar8  usedPINs       - Number of PINs used
MSCUChar8  usedKeys       - Number of Keys used
MSCUShort16  loggedID     - Bitmask of logged in identities

MSCKeyACL, *MSCLPKeyACL
- This structure contains a list of bitmasks used for an Access Control List (ACL) for a particular key. The bitmask will be a bitwise OR of the pre-defined AUT privileges.

[ ]
MSCUShort16  readPermission - Bitmask of AUT’s needed to read key
MSCUShort16  writePermission - Bitmask of AUT’s needed to write key
MSCUShort16  usePermission  - Bitmask of AUT’s needed to use key

MSCObjectACL, *MSCLPObjectACL
- This structure contains a list of bitmasks used for an Access Control List (ACL) for a particular object. The bitmask will be a bitwise OR of the pre-defined AUT privileges.

[ ]
MSCUShort16  readPermission - Bitmask of AUT’s needed to read object
MSCUShort16  writePermission - Bitmask of AUT’s needed to write object
MSCUShort16  deletePermission - Bitmask of AUT’s needed to delete object
MSCKeyPolicy, *MSLPKeyPolicy
- This structure is used to both describe a key usage policy for a key.

```
[  
  MSCUShort16  cipherMode - Bitmask of usage modes for policy  
  MSCUShort16  cipherDirection - Bitmask of direction modes for policy  
]
```

MSCKeyInfo, *MSLPKeyInfo
- This structure is used to describe the properties associated with a key.

```
[  
  MSCUChar8  keyNum - Key number used for identification of key  
  MSCUChar8  keyType - Type of key, algorithm/type  
  MSCUShort16  keySize - Size of the key in bits  
  MSCKeyPolicy  keyPolicy - Usage policy of the key  
  MSCKeyACL  keyACL - ACL used with this key  
]
```

MSCObjectInfo, *MSLPObjectInfo
- This structure is used to describe the properties associated with an object.

```
[  
  MSCChar8[]  objectID - Name used for object  
  MSCULong32  objectSize - Size of the object  
  MSCObjectACL  objectACL - ACL used with this object  
]
```

MSCGenKeyParams, *MSLPGenKeyParams
- This structure is used to set the parameters for on board key generation

```
[  
  MSCUChar8  algoType - Algorithm type  
  MSCUShort16  keySize - Key size in bits  
  MSCKeyACL  privateKeyACL - Private key ACL  
  MSCKeyACL  publicKeyACL - Public key ACL  
  MSCKeyPolicy  privateKeyPolicy - Private key usage policy  
  MSCKeyPolicy  publicKeyPolicy - Public key usage policy  
  MSCUChar8  keyGenOptions - Options bitmask for generation  
  MSCPUChar8  pOptParams - Reserved, set to NULL  
  MSCULong32  optParamsSize- Reserved, set to ZERO  
]
```
**MSCCryptInit, *MSCLPCryptInit**

- This structure is used to set the parameters for MSCComputeCrypt

```c
[  
  MSCUChar8  keyNum        - Key number  
  MSCUChar8  cipherMode    - Cipher mode  
  MSCUChar8  cipherDirection - Cipher direction  
  MSCPUCChar8 optParams     - Reserved, set to NULL  
  MSCUShort16 optParamsSize - Reserved, set to ZERO  
]
```

**MUSCLECARD AUTHENTICATION TYPES**

MuscleCard allows objects and keys to be protected through the use of it’s Access Control List (ACL). This list determines whether an application can perform a particular operation upon a key or object. Some operations might require no authentication to perform such as the reading of an object. Other operations such as the usage of a private key might require the authentication of a PIN to perform a signature function. The following is a list of pre-defined AUT’s which may be used in the ACL.

- **MSC_AUT_NONE** - The operation is never allowed
- **MSC_AUT_ALL** - The operation is always allowed
- **MSC_AUT_PIN_0** - The operation is allowed after PIN 0 verification
- **MSC_AUT_PIN_1** - The operation is allowed after PIN 1 verification
- **MSC_AUT_PIN_2** - The operation is allowed after PIN 2 verification
- **MSC_AUT_PIN_3** - The operation is allowed after PIN 3 verification
- **MSC_AUT_PIN_4** - The operation is allowed after PIN 4 verification
- **MSC_AUT_KEY_0** - The operation is allowed after KEY 0 authentication
- **MSC_AUT_KEY_1** - The operation is allowed after KEY 1 authentication
- **MSC_AUT_KEY_2** - The operation is allowed after KEY 2 authentication
- **MSC_AUT_KEY_3** - The operation is allowed after KEY 3 authentication
- **MSC_AUT_KEY_4** - The operation is allowed after KEY 4 authentication
- **MSC_AUT_KEY_5** - The operation is allowed after KEY 5 authentication

The following are reserved AUT’s used either for vendor specific capabilities or for future applet versions which support biometric pattern matching

- **MSC_AUT_USR_0** - The operation is allowed after USR 0 authentication
- **MSC_AUT_USR_1** - The operation is allowed after USR 1 authentication
MUSCLECARD TOKENS/SLOTS

Tokens can include any type of: smartcard, usb adaptor, pcmcia card, or generic cryptographic token in general. Slots are what contain the token. For example a smartcard reader would be a slot and the card itself would be the token. MuscleCard Framework provides a means for supporting tokens by a pluggable architecture. When an application uses MSCListTokens, the framework determines if the token is supported on the system. Tokens and slots both have names. A slot may not have a token in it, when this occurs the token name is: “Token Removed”. A slot which has a token which is unrecognized will have the token name: “Token Unknown”.

MUSCLECARD OBJECTS

MuscleCard objects are merely containers in which an application can store and retrieve data. These containers are fully generic in that they have not types nor format methods associated with them. This was done to allow a further specification to be written which addresses data formats, object id’s, etc. The goal was to have a clear separation of the interface and the data format.

Objects are fairly simple in design. Each object has a name or id which consists of 2 to 64 characters identifying it. Since the object id size is dependant on the token there is a tag in MSCGetCapabilities which returns the maximum object size for a token. An object also has a fixed size which is denoted at object creation. Each object also contains an ACL (Access Control List) which specifies what authentications are needed to read, write, and delete the object.

MUSCLECARD KEYS

Keys are identified by a number which can range from 0 to 15. Each key has specific properties such as key type, key size (in bits). Each key has both a key policy and an ACL associated with it. The key policy denotes how the key may be used, such as for signing only. The ACL specifies what authentications are needed to read, write, and use the key.

MUSCLECARD PINS

PINs are identified by a number which can range from 0 to 8. Each token will have a minimum and maximum size of a pin which can be retrieved using MSCGetCapabilities. PINs may also have a general pin policy which affects the entire token application. These policies might include pin strength, such as character sets, history checking, and case sensitivity.

MUSCLECARD MULTI-APPLICATION BEHAVIOR

Successful applications which use the MuscleCard Framework will allow them to be used in a multi-application environment where multiple applications would like to make use of the token and framework. To help provide this capability, MuscleCard framework has methods for both sharing the token, gaining exclusive access, and determining when the token has changed state such as after a reset.
Sharing

Applications may choose not to share the token which they connect to. This can be done in the MSCEstablishConnection function by using the MSC_SHARE_EXCLUSIVE tag for the share mode. Connection exclusivity can only be done if there is no other connection made to that token. If an application would like to share the token, but gain exclusive access to it when needed, it can use the MSC_SHARE_SHARED tag in MSCEstablishConnection. It may request to lock the token temporarily by calling MSCBeginTransaction and then release the lock by calling MSCEndTransaction.

Tracking Token State

MuscleCard Framework was designed to have a fairly simple state mechanism to ease the development of applications. A token may be either reset, authenticated, or moved. In traditional file system like tokens, the position of the file pointer must be considered a state. In MuscleCard it is up to the plugin to either maintain that state or to assume reset state with regards to the file pointer position at each transaction. MuscleCard applications do not worry about any other states besides: reset, authenticated, and removed.

The reset state occurs when application A shares a connection to a token with application B. Application A begins a transaction, verifies a pin, and then ends the transaction – resetting the token. MuscleCard will automatically re-establish all connections to the token automatically to a reset token.

The moved state occurs when the token is removed. Even if the token is removed and re-inserted it is still considered in the moved state. Once a token is in the moved state, any functions which use the token will return MSC_TOKEN_REMOVED. The application must then use MSCReleaseConnection to release the token, call MSCListTokens to refresh it’s record of tokens on the system, and then MSCEstablishConnection to re-establish connection.

Applications may check to see if a state change has occurred by using one of the following helper functions:

MSCIsTokenChanged(LPTokenConnection pConnection)
-returns 1 if the token is either moved or reset. returns 0 otherwise.

MSCIsTokenMoved(LPTokenConnection pConnection)
-returns 1 if the token is in moved state. returns 0 otherwise.

MSCIsTokenReset(LPTokenConnection pConnection)
-returns 1 if the token is in reset state. returns 0 otherwise.

MSCClearReset(LPTokenConnection pConnection)
 -clears the reset state. (application acknowledges the token was reset)
NAME

MSCListTokens - Lists tokens available on the system

SYNOPSIS

#include <musclecard.h>

MSCListTokens(
    MSCULong32   listScope,
    MSCLPTokenInfo    tokenArray,
    MSCPULong32  arrayLength
);

PARAMETERS

listScope    Scope of the desired listing
tokenArray Array of MSCTokenInfo structures returned
arrayLength Number of structures in tokenArray

DESCRIPTION

This function returns the tokens available on a system. arrayLength
is an INOUT variable. On IN it specifies the number of array structures
allocated by the application. On OUT it specifies the actual number of
structures returned. If either tokenArray or arrayLength are NULL the
function returns the number of structures in the array.

listScope specifies the scope of return and can be one of the following:

MSC_LIST_KNOWN List only tokens supported
MSC_LIST_ALL List all tokens whether supported or not
MSC_LIST_SLOTS List every slot even if no token is inserted

RETURN VALUES

Reference previously defined error codes.

EXAMPLES

MSCLPTokenInfo tokenList;
MSCTokenConnection pConnection;
MSC_RV rv; MSCULong32 listSize = 0;

rv = MSCListTokens( MSC_LIST_KNOWN, NULL, &listSize );
if (rv == MSC_SUCCESS) {
    tokenList = (MSCLPTokenInfo)malloc(sizeof(MSCTokenInfo)*listSize);
    rv = MSCListTokens( MSC_LIST_KNOWN, tokenList, &listSize );
    if ( rv == MSC_SUCCESS ) {
        printf("Token name    : %s\n", tokenList[0].tokenName);
        printf("Slot name     : %s\n", tokenList[0].slotName);
    }
}

SEE ALSO

MSCListTokens, MSCEstablishConnection, MSCReleaseConnection,
MSCBeginTransaction, MSCEndTransaction, MSCWaitForTokenEvent
NAME

MSCEstablishConnection – Establishes a connection to a token/slot

SYNOPSIS

#include <musclecard.h>

MSCEstablishConnection(
    MSCLPTokenInfo   tokenStruct,
    MSCULong32                sharingMode,
    MSCPUC Char8   applicationName,
    MSCULong32   nameSize,
    MSCLPTokenConnection  pConnection
);

PARAMETERS

tokenStruct Pointer to structure returned by MSCListTokens
sharingMode Determines if the token is shared
applicationName Application or applet ID
nameSize Length of the applicationName
pConnection Handle for this connection

DESCRIPTION

This function establishes a connection to a particular token which was
returned by MSCListTokens. applicationName can be an Applet ID (AID) or
it and nameSize can be set to NULL, indicating the default Application
should be used. pConnection is returned as a handle to all following
functions.

sharingMode is one of the following values:

MSC_SHARE_SHARED Allow this token to be shared
MSC_SHARE_EXCLUSIVE Do not allow sharing of this token
MSC_SHARE_DIRECT Connect directly to the reader (shared)

RETURN VALUES

Reference previously defined error codes.

EXAMPLES

MSCTokenInfo tokenList[16]; // 16 used as example
MSCTokenConnection pConnection;
MSC_RV rv; MSCULong32 listSize = 16;

rv = MSCListTokens( MSC_LIST_KNOWN, tokenList, &listSize );
if (rv == MSC_SUCCESS) {
    rv = MSCEstablishConnection( &tokenList[0], MSC_SHARE_SHARED, NULL,
                                 NULL, &pConnection );
    if (rv == MSC_SUCCESS)
    {
        ...
    }
}

SEE ALSO

MSCListTokens, MSCEstablishConnection, MSCReleaseConnection,
MSCBeginTransaction, MSCEndTransaction, MSCWaitForTokenEvent
NAME

MSCReleaseConnection - Releases a previous connection

SYNOPSIS
#include <musclecard.h>

MSCReleaseConnection(
    MSCLPTokenConnection pConnection,
    MSCULong32   endAction
);

PARAMETERS
pConnection Handle to a previously connected session
endAction Action to be performed on token

DESCRIPTION
This function releases a previous connection made by calling
MSCEstablishConnection.

endAction performs one of the following actions on the token:
MSC_LEAVE_TOKEN Do nothing to the token
MSC_RESET_TOKEN Reset the token
MSC_EJECT_TOKEN Physically eject the token

RETURN VALUES
Reference previously defined error codes.

EXAMPLES
MSCTokenInfo tokenList[16]; // 16 used as example
MSCTokenConnection pConnection;
MSC_RV rv; MSCULong32 listSize = 16;

rv = MSCListTokens( MSC_LIST_KNOWN, tokenList, &listSize );
if (rv == MSC_SUCCESS) {
    rv = MSCEstablishConnection( &tokenList[0], MSC_SHARE_SHARED, NULL,
                                NULL, &pConnection );
    if (rv == MSC_SUCCESS)
    {
        rv = MSCReleaseConnection(&pConnection, MSC_LEAVE_TOKEN);
        if (rv == MSC_SUCCESS)
        {
            ...
        }
    }
}

SEE ALSO
MSCListTokens, MSCEstablishConnection, MSCReleaseConnection,
MSCBeginTransaction, MSCEndTransaction, MSCWaitForTokenEvent
NAME

MSCWaitForTokenEvent - Waits for a token event

SYNOPSIS
#include <musclecard.h>

MSCWaitForTokenEvent(
    MSCLPTokenInfo    tokenArray,
    MSCULong32        arraySize,
    MSCULong32        timeoutValue
);

PARAMETERS

    tokenArray    Array of token structures
    arraySize     Number of token structure in array
    timeoutValue  Timeout value in milliseconds

DESCRIPTION

This function waits (blocks) for an event to occur on a particular token
or tokens. The application may either specify which events it is
interested in or it may choose to block for any event. Typical events
would include the insertion or removal of a token into a particular slot.
A newly inserted token would update the friendlyname of the token if it
is identified. If you set tokenState to zero, this will return on any
new event which occurs to the tokenArray items. MSC_NO_TIMEOUT will block
forever.

tokenState in tokenArray is a bitmask of the following:

    MSC_STATE_UNAWARE  Return immediately with the state
    MSC_STATE_CHANGED  A change in state has occurred
    MSC_STATE_UNKNOWN  The state of this token/slot is unknown
    MSC_STATE_UNAVAILABLE A token/slot has become unavailable
    MSC_STATE_EMPTY    A token was removed from the slot
    MSC_STATE_PRESENT  A token was inserted into the slot
    MSC_STATE_EXCLUSIVE The token is in exclusive mode
    MSC_STATE_INUSE    The token already has a connection
    MSC_STATE_MUTE     The token is unresponsive

RETURN VALUES

Reference previously defined error codes.

EXAMPLES
MSCTokenInfo tokenList[16];  // 16 used as example
MSCTokenConnection pConnection;
MSC_RV rv; MSCULong32 listSize = 16;

    rv = MSCListTokens( MSC_LIST_KNOWN, tokenList, &listSize );
    tokenList[0].tokenState = MSC_STATE_EMPTY; // wait for insertion
    rv = MSCWaitForTokenEvent( &tokenList[0], listSize, MSC_NO_TIMEOUT );
    rv = MSCEstablishConnection( &tokenList[0], MSC_SHARE_SHARED, NULL,
                                 NULL, &pConnection );

SEE ALSO

MSCListTokens, MSCEstablishConnection, MSCReleaseConnection,
MSCBeginTransaction, MSCEndTransaction, MSCCancelEventWait
NAME

MSCCancelEventWait - Cancels a pending MSCWaitForTokenEvent

SYNOPSIS

#include <musclecard.h>

MSCCancelEventWait(
    void
);

PARAMETERS

DESCRIPTION

This function cancels all pending blocks for events in the function
MSCWaitForTokenEvent. Each function will return immediately with the
value MSC_CANCELLED.

RETURN VALUES

Reference previously defined error codes.

EXAMPLES

MSCTokenInfo tokenList[16];  // 16 used as example
MSCTokenConnection pConnection;
MSC_RV rv; MSCULong32 listSize = 16;

rv = MSCListTokens( MSC_LIST_KNOWN, tokenList, &listSize );
if (rv == MSC_SUCCESS) {
    ... Start new thread and start function MSCWaitForTokenEvent

    rv = MSCCancelEventWait();
    if (rv == MSC_SUCCESS)
    {
        // The blocking function will return
    }
}

SEE ALSO

MSCListTokens, MSCEstablishConnection, MSCReleaseConnection,
MSCBeginTransaction, MSCEndTransaction, MSCWaitForTokenEvent
NAME

MSCCallbackForTokenEvent – Register a callback for token events

SYNOPSIS

#include <musclecard.h>

MSCCallbackForTokenEvent(
    MSCLPTokenInfo   tokenArray,
    MSCULong32      arraySize,
    MSCCallback      callBack,
    MSCPVoid32       appData
);

PARAMETERS

tokenArray          Array of token structures
arraySize           Number of token structure in array
callBack            Callback function
appData             Application data passed to the callback

DESCRIPTION

This function spawns a thread which waits for events to occur to a
token or list of tokens specified by tokenArray. When an event occurs,
the function registered (callBack) will be called from the thread which
will pass the application data along with the tokenArray with updated
state structures so the application can determine which state has changed.

tokenState in tokenArray should be set to zero (0).

RETURN VALUES

Reference previously defined error codes.

EXAMPLES

MSCULong32 myCallback(MSCLPTokenInfo tokenList, MSCULong32 listSize,
                        MSCPVoid32 appData) {
    printf("I received an event\n");
}

MSCTokenInfo tokenList[16]; // 16 used as example
MSCTokenConnection pConnection;
MSC_RV rv; MSCULong32 listSize = 16;

rv = MSCListTokens( MSC_LIST_KNOWN, tokenList, &listSize );
tokenList[0].tokenState = 0; // wait for any event
rv = MSCCallbackForTokenEvent( &tokenList[0], 1, myCallback,
                               NULL );

SEE ALSO

MSCListTokens, MSCEstablishConnection, MSCReleaseConnection,
MSCBeginTransaction, MSCEndTransaction, MSCCancelEventWait,
MSCWaitForTokenEvent, MSCCallbackCancelEvent
NAME

MSCCallbackCancelEvent – Cancels a registered callback

SYNOPSIS
#include <musclecard.h>
MSCCallbackCancelEvent();

PARAMETERS
none

DESCRIPTION
This function cancels a registered callback which was previously
registered using MSCCallbackForTokenEvent.

RETURN VALUES
Reference previously defined error codes.

EXAMPLES
MSCULong32 myCallback(MSCLPTokenInfo tokenList, MSCULong32 listSize,
MSCPVoid32 appData) {
    printf("I received an event\n");
}

MSCTokenInfo tokenList[16]; // 16 used as example
MSCTokenConnection pConnection;
MSC_RV rv; MSCULong32 listSize = 16;
rv = MSCListTokens( MSC_LIST_KNOWN, tokenList, &listSize );
tokenList[0].tokenState = 0; // wait for any event
rv = MSCCallbackForTokenEvent( &tokenList[0], 1, myCallback,
    NULL );
rv = MSCCallbackCancelEvent();

SEE ALSO
MSCListTokens, MSCEstablishConnection, MSCReleaseConnection,
MSCBeginTransaction, MSCEndTransaction, MSCCancelEventWait,
MSCWaitForTokenEvent, MSCCallbackForTokenEvent
NAME

MSCBeginTransaction – Acquires a lock for a given transaction

SYNOPSIS

```c
#include <musclecard.h>

MSCBeginTransaction(
    MSCLPTokenConnection   pConnection
);
```

PARAMETERS

- `pConnection` Handle to a previously connected session

DESCRIPTION

This function requests a lock to secure an upcoming transaction. If another application holds the lock, this function will block until the other application releases the lock.

RETURN VALUES

Reference previously defined error codes.

EXAMPLES

```c
MSCTokenInfo tokenList[16]; // 16 used as example
MSCTokenConnection pConnection;
MSC_RV rv; MSCULong32 listSize = 16;

rv = MSCLListTokens( MSC_LIST_KNOWN, tokenList, &listSize );
if (rv == MSC_SUCCESS) {
    rv = MSCEstablishConnection( &tokenList[0], MSC_SHARE_SHARED, NULL,
                                NULL, &pConnection );
    if (rv == MSC_SUCCESS)
    {
        rv = MSCBeginTransaction(&pConnection);
        if (rv == MSC_SUCCESS)
        {
            ...
        }
    }
}
```

SEE ALSO

MSCLListTokens, MSCEstablishConnection, MSCReleaseConnection,
MSCBeginTransaction, MSCEndTransaction, MSCWaitForTokenEvent,
MSCCancelEventWait
NAME

MSCEndTransaction – Releases a lock for a given transaction

SYNOPSIS

#include <musclecard.h>

MSCEndTransaction(
    MSCLPTokenConnection pConnection,
    MSCULong32   endAction
);

PARAMETERS

pConnection    Handle to a previously connected session
endAction  Action to be performed on token

DESCRIPTION

This function releases a lock which was previously acquired using
MSCBeginTransaction.

dAction performs one of the following actions on the token:

MSC_LEAVE_TOKEN Do nothing to the token
MSC_RESET_TOKEN Reset the token
MSC_EJECT_TOKEN Physically eject the token

RETURN VALUES

Reference previously defined error codes.

EXAMPLES

MSCTokenInfo tokenList[16];  // 16 used as example
MSCTokenConnection pConnection;
MSC_RV rv; MSCULong32 listSize = 16;

rv = MSCListTokens( MSC_LIST_KNOWN, tokenList, &listSize );
if (rv == MSC_SUCCESS) {
    rv = MSCEstablishConnection( &tokenList[0], MSC_SHARE_SHARED, NULL, NULL, &pConnection );
    if (rv == MSC_SUCCESS) {
        rv = MSCBeginTransaction(&pConnection);
        if (rv == MSC_SUCCESS)
        {
            ...
            rv = MSCEndTransaction(&pConnection, MSC_RESET_TOKEN);
        }
    }
}

SEE ALSO

MSCListTokens, MSCEstablishConnection, MSCReleaseConnection,
MSCBeginTransaction, MSCEndTransaction, MSCWaitForTokenEvent,
MSCCancelEventWait
NAME

MSCGetStatus - Gets the Applet's Status Information

SYNOPSIS

#include <musclecard.h>

MSCGetStatus(
    MSCLPTokenConnection pConnection,
    MSCLPStatusInfo pStatusInfo
);

PARAMETERS

pConnection Handle to a previously connected session
pStatusInfo Returns the status information

DESCRIPTION

This function returns status information about the applet including the applet version, available memory, and logged authentication ID's.

RETURN VALUES

Reference previously defined error codes.

EXAMPLES

MSCTokenInfo tokenList[16]; // 16 used as example
MSCStatusInfo appStatus;
MSCTokenConnection pConnection;
MSC_RV rv; MSCULong32 listSize = 16;

rv = MSCListTokens( MSC_LIST_KNOWN, tokenList, &listSize );
if (rv == MSC_SUCCESS) {
    rv = MSCEstablishConnection( &tokenList[0], MSC_SHARE_SHARED, NULL,
                                  NULL, &pConnection );
    if (rv == MSC_SUCCESS)
    {
        rv = MSCGetStatus(&pConnection, &appStatus);
        if (rv == MSC_SUCCESS)
        {
            ...
        }
    }
}

SEE ALSO

MSCListTokens, MSCEstablishConnection, MSCReleaseConnection,
MSCBeginTransaction, MSCEndTransaction
NAME

MSCGetCapabilities - Gets the token’s supported capabilities

SYNOPSIS

#include <musclecard.h>

MSCGetCapabilities(
    MSCLPTokenConnection pConnection,
    MSCULong32 Tag, 
    MSCPUChar8 Value, 
    MSCPULong32 Length
);

PARAMETERS

pConnection  Handle to a previously connected session
Tag    Defined tag of information to retrieve
Value   Value of information returned
Length   Length of the information returned

DESCRIPTION

This function returns the capabilities of the connected token. These capabilities range from cryptographic functionality, behavior, etc. All listed tags and their potential values are listed at the end of this reference.

RETURN VALUES

Reference previously defined error codes.

EXAMPLES

MSCTokenInfo tokenList[16]; // 16 used as example
MSCConnection pConnection;
MSC_RV rv; MSCULong32 listSize = 16;
MSCULong32 algoLength, algoSupported;

rv = MSCListTokens( MSC_LIST_KNOWN, tokenList, &listSize );
if (rv == MSC_SUCCESS) {
    rv = MSCEstablishConnection( &tokenList[0], MSC_SHARE_SHARED, NULL, 
                                 NULL, &pConnection );
    if (rv == MSC_SUCCESS)
    {
        rv = MSCGetCapabilities(&pConnection, MSC_TAG_SUPPORT_CRYPTOALG,
                                (MSCPUChar8)&algoSupported, &algoLength );
        if (rv == MSC_SUCCESS)
        {
            if ( algoSupported & MSC_SUPPORT_AES )
                printf("Card supports AES\n");
        }
    }
}

SEE ALSO

MSCListTokens, MSCEstablishConnection, MSCReleaseConnection, 
MSCBeginTransaction, MSCEndTransaction, MSCGetStatus
NAME

MSCExtendedFeature – Exchanges vendor extended functionality

SYNOPSIS

#include <musclecard.h>

MSCExtendedFeature(
    MSCLPTokenConnection pConnection,
    MSCULong32 extFeature,
    MSCPUChar8 outData,
    MSCULong32 outLength,
    MSCPUChar8 inData,
    MSCPULong32 inLength
);

PARAMETERS

pConnection  Handle to a previously connected session
extFeature  Tag for extended feature
outData   Outgoing data
outLength   Outgoing data length
inData   Incoming data
inLength   Incoming data length

DESCRIPTION

This function allows vendor extended functionality outside the scope of this framework. For example, a vendor might have a card that supports self destruction. This function could send a vendor specific command to the card to perform this.

RETURN VALUES

Reference previously defined error codes.

EXAMPLES

MSCTokenInfo tokenList[16]; // 16 used as example
MSCTokenConnection pConnection;
MSC_RV rv; MSCULong32 listSize = 16;

rv = MSCLListTokens( MSC_LIST_KNOWN, tokenList, &listSize );
if (rv == MSC_SUCCESS) {
    rv = MSCEstablishConnection( &tokenList[0], MSC_SHARE_SHARED, NULL,
        NULL, &pConnection );
    if (rv == MSC_SUCCESS) {
        rv = MSCExtendedFeature(&pConnection, VEND_KILL_CARD, NULL, 0,
            NULL, NULL);
        if (rv == MSC_SUCCESS)
            { ...
        }
    }
}

SEE ALSO

MSCLListTokens, MSCEstablishConnection, MSCReleaseConnection,
MSCBeginTransaction, MSCEndTransaction, MSCGetCapabilities, MSCGetStatus
NAME

MSCGenerateKeys - Generates keys on the token

SYNOPSIS
#include <musclecard.h>

MSCGenerateKeys(
    MSCLPTokenConnection   pConnection,
    MSCUChar8              prvKeyNum,
    MSCUChar8              pubKeyNum,
    MSCLPGenKeyParams      pParams
);

PARAMETERS
    pConnection        Handle to a previously connected session
    prvKeyNum          Private key number
    pubKeyNum          Public key number
    pParams            Additional generation parameters

DESCRIPTION
This function uses the token's on board key generation facilities
to generate a set of public and private keys for use with public
key cryptography.

pParams is a structure containing the following fields:
    algoType            Algorithm type
    keySize             Size of keys
    privateKeyACL       Private key ACL
    publicKeyACL       Public key ACL
    privateKeyPolicy  Private key usage policy
    publicKeyPolicy  Public key usage policy
    keyGenOptions       Key generation options
    pOptParams          Optional parameters
    optParamsSize       Optional parameters size

    pParams.algoType
    MSC_GEN_ALG_RSA         Generate an RSA keypair (modulus/exponent)
    MSC_GEN_ALG_RSA_CRT   Generate an RSA keypair (chinese remainder)
    MSC_GEN_ALG_DSA  Generate a DSA keypair

    pParams.keySize
      512, 768, 1024, 2048 ...

    pParams.privateKeyACL
    pParams.publicKeyACL

    readPermission        Bitwise 'OR' with defined ACL values
    writePermission       Bitwise 'OR' with defined ACL values
    usePermission         Bitwise 'OR' with defined ACL values
pParams.privateKeyPolicy
pParams.publicKeyPolicy

pParams.privateKeyPolicy.cipherDirection
pParams.publicKeyPolicy.cipherDirection
MSC_KEYPOLICY_DIR_SIGN        Can be used for signing
MSC_KEYPOLICY_DIR_VERIFY       Can be used for verification
MSC_KEYPOLICY_DIR_ENCRYPT     Can be used for encryption
MSC_KEYPOLICY_DIR_DECRYPT     Can be used for decryption

pParams.privateKeyPolicy.cipherMode
pParams.publicKeyPolicy.cipherMode
MSC_KEYPOLICY_MODE_RSA_NOPAD   RSA can be used with no pad
MSC_KEYPOLICY_MODE_RSA_PAD_PKCS1 RSA can be used with pkcs pad
MSC_KEYPOLICY_MODE_DSA_SHA       DSA can be used with SHA
MSC_KEYPOLICY_MODE_DES_CBC_NOPAD DES can be used CBC nopad
MSC_KEYPOLICY_MODE_DES_ECB_NOPAD DES can be used ECB nopad

pParams.keyGenOptions
MSC_OPT_DEFAULT            Use default options

pParams.pOptParams
Reserved for future use (RFU)

pParams.optParamsSize
Reserved for future use (RFU)

RETURN VALUES
Reference previously defined error codes.

EXAMPLES
MSCTokenInfo tokenList[16]; // 16 used as example
MSCTokenConnection pConnection;
MSCGenKeyParams keyParams;
MSC_RV rv; MSCULong32 listSize = 16;

rv = MSCListTokens( MSC_LIST_KNOWN, tokenList, &listSize );
if (rv == MSC_SUCCESS) {
  rv = MSCEstablishConnection( &tokenList[0], MSC_SHARE_SHARED, NULL,
                               NULL, &pConnection );
  if (rv == MSC_SUCCESS)
  {
    keyParams.algoType = MSC_GEN_ALG_RSA;
    keyParams.keySize  = 1024;

    keyParams.privateKeyACL.readPermission  = MSC_AUT_NONE;
    keyParams.privateKeyACL.writePermission = MSC_AUT_NONE;
    keyParams.privateKeyACL.usePermission   = MSC_AUT_PIN_0;

    keyParams.publicKeyACL.readPermission   = MSC_AUT_ANY;
    keyParams.publicKeyACL.writePermission  = MSC_AUT_PIN_0;
    keyParams.publicKeyACL.usePermission    = MSC_AUT_PIN_0;
/* Signing only key */
keyParams.privateKeyPolicy.cipherDirection = MSC_KEYPOLICY_DIR_SIGN;
keyParams.publicKeyPolicy.cipherDirection = 0;
keyParams.privateKeyPolicy.cipherMode = MSC_KEYPOLICY_MODE_RSA_NOPAD;
keyParams.publicKeyPolicy.cipherMode = 0;

keyParams.keyGenOptions = MSC_OPT_DEFAULT;
keyParams.optParamsSize = 0;

rv = MSCGenerateKeys(&pConnection, 0, 1, &keyParams);
if (rv == MSC_SUCCESS)
{

... 
}

SEE ALSO
MSCListTokens, MSCEstablishConnection, MSCReleaseConnection,
MSCBeginTransaction, MSCEndTransaction, MSCImportKey, MSCExportKey,
MSCListKeys, MSCExtAuthenticate, MSCComputeCrypt
NAME

MSCImportKey - Import an externally generated key

SYNOPSIS

#include <musclecard.h>

MSCImportKey(
    MSCLPTokenConnection   pConnection,
    MSCLPKeyACL            pKeyACL,
    MSCLPKeyPolicy    keyPolicy,
    MSCUChar8              keyNum,
    MSCLPKeyBlob           pKeyBlob,
    MSCLPKeyACL            keyACL,
    MSCLPKeyBlob           pKeyBlob,
    MSCLPKeyPolicy    keyPolicy,
    MSCLPKeyBlob           pKeyBlob,
    MSCLPKeyACL            keyACL,
    MSCLPKeyBlob           pKeyBlob,
    MSCLPKeyPolicy    keyPolicy,
    MSCLPKeyBlob           pKeyBlob,
    MSCLPKeyACL            keyACL,
    MSCLPKeyBlob           pKeyBlob,
    MSCLPKeyPolicy    keyPolicy,
    MSCLPKeyBlob           pKeyBlob,
    MSCLPKeyACL            keyACL,
    MSCLPKeyBlob           pKeyBlob,
    MSCLPKeyPolicy    keyPolicy,
    MSCLPKeyBlob           pKeyBlob,
    MSCLPKeyACL            keyACL,
    MSCLPKeyBlob           pKeyBlob,
    MSCLPKeyPolicy    keyPolicy,
    MSCLPKeyBlob           pKeyBlob,
    MSCLPKeyACL            keyACL,
    MSCLPKeyBlob           pKeyBlob,
    MSCLPKeyPolicy    keyPolicy,
    MSCLPKeyBlob           pKeyBlob,
    MSCLPKeyACL            keyACL,
    MSCLPKeyBlob           pKeyBlob,
    MSCLPKeyPolicy    keyPolicy,
    MSCLPKeyBlob           pKeyBlob,
    MSCLPKeyACL            keyACL,
    MSCLPKeyBlob           pKeyBlob,
    MSCLPKeyPolicy    keyPolicy,
    MSCLPKeyBlob           pKeyBlob,
    MSCLPKeyACL            keyACL,
    MSCLPKeyBlob           pKeyBlob,
    MSCLPKeyPolicy    keyPolicy,
    MSCLPKeyBlob           pKeyBlob,
    MSCLPKeyACL            keyACL,
    MSCLPKeyBlob           pKeyBlob,
    MSCLPKeyPolicy    keyPolicy,
    MSCLPKeyBlob           pKeyBlob,
    MSCLPKeyACL            keyACL,
    MSCLPKeyBlob           pKeyBlob,
    MSCLPKeyPolicy    keyPolicy,
    MSCLPKeyBlob           pKeyBlob,
    MSCLPKeyACL            keyACL,
    MSCLPKeyBlob           pKeyBlob,
    MSCLPKeyPolicy    keyPolicy,
    MSCLPKeyBlob           pKeyBlob,
    MSCLPKeyACL            keyACL,
    MSCLPKeyBlob           pKeyBlob,
    MSCLPKeyPolicy    keyPolicy,
    MSCLPKeyBlob           pKeyBlob,
    MSCLPKeyACL            keyACL,
    MSCLPKeyBlob           pKeyBlob,
    MSCLPKeyPolicy    keyPolicy,
    MSCLPKeyBlob           pKeyBlob,
    MSCLPKeyACL            keyACL,
    MSCLPKeyBlob           pKeyBlob,
    MSCLPKeyPolicy    keyPolicy,
    MSCLPKeyBlob           pKeyBlob,
    MSCLPKeyACL            keyACL,
    MSCLPKeyBlob           pKeyBlob,
    MSCLPKeyPolicy    keyPolicy,
    MSCLPKeyBlob           pKeyBlob,
    MSCLPKeyACL            keyACL,
    MSCLPKeyBlob           pKeyBlob,
    MSCLPKeyPolicy    keyPolicy,
    MSCLPKeyBlob           pKeyBlob,
    MSCLPKeyACL            keyACL,
    MSCLPKeyBlob           pKeyBlob,
    MSCLPKeyPolicy    keyPolicy,
    MSCLPKeyBlob           pKeyBlob,
    MSCLPKeyACL            keyACL,
    MSCLPKeyBlob           pKeyBlob,
    MSCLPKeyPolicy    keyPolicy,
    MSCLPKeyBlob           pKeyBlob,
    MSCLPKeyACL            keyACL,
    MSCLPKeyBlob           pKeyBlob,
    MSCLPKeyPolicy    keyPolicy,
    MSCLPKeyBlob           pKeyBlob,
    MSCLPKeyACL            keyACL,
    MSCLPKeyBlob           pKeyBlob,
    MSCLPKeyPolicy    keyPolicy,
    MSCLPKeyBlob           pKeyBlob,
    MSCLPKeyACL            keyACL,
    MSCLPKeyBlob           pKeyBlob,
    MSCLPKeyPolicy    keyPolicy,
    MSCLPKeyBlob           pKeyBlob,
    MSCLPKeyACL            keyACL,
    MSCLPKeyBlob           pKeyBlob,
    MSCLPKeyPolicy    keyPolicy,
    MSCLPKeyBlob           pKeyBlob,
    MSCLPKeyACL            keyACL,
    MSCLPKeyBlob           pKeyBlob,
    MSCLPKeyPolicy    keyPolicy,
    MSCLPKeyBlob           pKeyBlob,
    MSCLPKeyACL            keyACL,
    MSCLPKeyBlob           pKeyBlob,
    MSCLPKeyPolicy    keyPolicy,
    MSCLPKeyBlob           pKeyBlob,
    MSCLPKeyACL            keyACL,
    MSCLPKeyBlob           pKeyBlob,
    MSCLPKeyPolicy    keyPolicy,
    MSCLPKeyBlob           pKeyBlob,
    MSCLPKeyACL            keyACL,
    MSCLPKeyBlob           pKeyBlob,
    MSCLPKeyPolicy    keyPolicy,
    MSCLPKeyBlob           pKeyBlob,
    MSCLPKeyACL            keyACL,
    MSCLPKeyBlob           pKeyBlob,
    MSCLPKeyPolicy    keyPolicy,
    MSCLPKeyBlob           pKeyBlob,
    MSCLPKeyACL            keyACL,
    MSCLPKeyBlob           pKeyBlob,
    MSCLPKeyPolicy    keyPolicy,
    MSCLPKeyBlob           pKeyBlob,
    MSCLPKeyACL            keyACL,
    MSCLPKeyBlob           pKeyBlob,
    MSCLPKeyPolicy    keyPolicy,
    MSCLPKeyBlob           pKeyBlob,
    MSCLPKeyACL            keyACL,
    MSCLPKeyBlob           pKeyBlob,
    MSCLPKeyPolicy    keyPolicy,
    MSCLPKeyBlob           pKeyBlob,
    MSCLPKeyACL            keyACL,
    MSCLPKeyBlob           pKeyBlob,
    MSCLPKeyPolicy    keyPolicy,
    MSCLPKeyBlob           pKeyBlob,
    MSCLPKeyACL            keyACL,
    MSCLPKeyBlob           pKeyBlob,
    MSCLPKeyPolicy    keyPolicy,
    MSCLPKeyBlob           pKeyBlob,
    MSCLPKeyACL            keyACL,
    MSCLPKeyBlob           pKeyBlob,
    MSCLPKeyPolicy    keyPolicy,
    MSCLPKeyBlob           pKeyBlob,
    MSCLPKeyACL            keyACL,
    MSCLPKeyBlob           pKeyBlob,
    MSCLPKeyPolicy    keyPolicy,
    MSCLPKeyBlob           pKeyBlob,
    MSCLPKeyACL            keyACL,
    MSCLPKeyBlob           pKeyBlob,
    MSCLPKeyPolicy    keyPolicy,
    MSCLPKeyBlob           pKeyBlob,
    MSCLPKeyACL            keyACL,
    MSCLPKeyBlob           pKeyBlob,
    MSCLPKeyPolicy    keyPolicy,
    MSCLPKeyBlob           pKeyBlob,
    MSCLPKeyACL            keyACL,
    MSCLPKeyBlob           pKeyBlob,
    MSCLPKeyPolicy    keyPolicy,
    MSCLPKeyBlob           pKeyBlob,
    MSCLPKeyACL            keyACL,
    MSCLPKeyBlob           pKeyBlob,
    MSCLPKeyPolicy    keyPolicy,
    MSCLPKeyBlob           pKeyBlob,
    MSCLPKeyACL            keyACL,
    MSCLPKeyBlob           pKeyBlob,
    MSCLPKeyPolicy    keyPolicy,
    MSCLPKeyBlob           pKeyBlob,
    MSCLPKeyACL            keyACL,
    MSCLPKeyBlob           pKeyBlob,
    MSCLPKeyPolicy    keyPolicy,
    MSCLPKeyBlob           pKeyBlob,
    MSCLPKeyACL            keyACL,
    MSCLPKeyBlob           pKeyBlob,
    MSCLPKeyPolicy    keyPolicy,
    MSCLPKeyBlob           pKeyBlob,
    MSCLPKeyACL            keyACL,
    MSCLPKeyBlob           pKeyBlob,
    MSCLPKeyPolicy    keyPolicy,
    MSCLPKeyBlob           pKeyBlob,
    MSCLPKeyACL            keyACL,
    MSCLPKeyBlob           pKeyBlob,
    MSCLPKeyPolicy    keyPolicy,
    MSCLPKeyBlob           pKeyBlob,
    MSCLPKeyACL            keyACL,
    MSCLPKeyBlob           pKeyBlob,
    MSCLPKeyPolicy    keyPolicy,
    MSCLPKeyBlob           pKeyBlob,
    MSCLPKeyACL            keyACL,
    MSCLPKeyBlob           pKeyBlob,
    MSCLPKeyPolicy    keyPolicy,
    MSCLPKeyBlob           pKeyBlob,
    MSCLPKeyACL            keyACL,
    MSCLPKeyBlob           pKeyBlob,
    MSCLPKeyPolicy    keyPolicy,
    MSCLPKeyBlob           pKeyBlob,
    MSCLPKeyACL            keyACL,
    MSCLPKeyBlob           pKeyBlob,
    MSCLPKeyPolicy    keyPolicy,
    MSCLPKeyBlob           pKeyBlob,
    MSCLPKeyACL            keyACL,
    MSCLPKeyBlob           pKeyBlob,
    MSCLPKeyPolicy    keyPolicy,
    MSCLPKeyBlob           pKeyBlob,
    MSCLPKeyACL            keyACL,
    MSCLPKeyBlob           pKeyBlob,
    MSCLPKeyPolicy    keyPolicy,
    MSCLPKeyBlob           pKeyBlob,
    MSCLPKeyACL            keyACL,
    MSCLPKeyBlob           pKeyBlob,
    MSCLPKeyPolicy    keyPolicy,
    MSCLPKeyBlob           pKeyBlob,
    MSCLPKeyACL            keyACL,
    MSCLPKeyBlob           pKeyBlob,
    MSCLPKeyPolicy    keyPolicy,
    MSCLPKeyBlob           pKeyBlob,
    MSCLPKeyACL            keyACL,
    MSCLPKeyBlob           pKeyBlob,
    MSCLPKeyPolicy    keyPolicy,
    MSCLPKeyBlob           pKeyBlob,
    MSCLPKeyACL            keyACL,
    MSCLPKeyBlob           pKeyBlob,
    MSCLPKeyPolicy    keyPolicy,
    MSCLPKeyBlob           pKeyBlob,
    MSCLPKeyACL            keyACL,
    MSCLPKeyBlob           pKeyBlob,
    MSCLPKeyPolicy    keyPolicy,
    MSCLPKeyBlob           pKeyBlob,
    MSCLPKeyACL            keyACL,
    MSCLPKeyBlob           pKeyBlob,
    MSCLPKeyPolicy    keyPolicy,
    MSCLPKeyBlob           pKeyBlob,
    MSCLPKeyACL            keyACL,
    MSCLPKeyBlob           pKeyBlob,
    MSCLPKeyPolicy    keyPolicy,
    MSCLPKeyBlob           pKeyBlob,
    MSCLPKeyACL            keyACL,
    MSCLPKeyBlob           pKeyBlob,
    MSCLPKeyPolicy    keyPolicy,
    MSCLPKeyBlob           pKeyBlob,
    MSCLPKeyACL            keyACL,
    MSCLPKeyBlob           pKeyBlob,
    MSCLPKeyPolicy    keyPolicy,
    MSCLPKeyBlob           pKeyBlob,
    MSCLPKeyACL            keyACL,
{
    impACL.readPermission = MSC_AUT_ALL;
    impACL.writePermission = MSC_AUT_NONE;
    impACL.usePermission = MSC_AUT_PIN_0;

    /* This following function is for demo only */
    myKeyBlobSize = getMyRSAKeyBlob(myKeyBlob);
    rv = MSCImportKey(&pConnection, 1, &impACL, myKeyBlob,
                      myKeyBlobSize, 0, 0);
    if (rv == MSC_SUCCESS)
    {
        ...
    }
}

SEE ALSO
MSCListTokens, MSCEstablishConnection, MSCReleaseConnection,
MSCBeginTransaction, MSCEndTransaction, MSCImportKey, MSCExportKey,
MSCListKeys, MSCExtAuthenticate, MSCComputeCrypt
NAME

MSCExportKey - Export a card key

SYNOPSIS

#include <musclecard.h>

MSCExportKey(
    MSCLPTokenConnection   pConnection,
    MSCUChar8                keyNum,
    MSCPUChar8               pKeyBlob,
    MSCPULong32               keyBlobSize,
    MSCPVoid32               pAddParams,
    MSCUChar8                addParamsSize
);

PARAMETERS

pConnection     Handle to a previously connected session
keyNum          Key number to retrieve key
pKeyBlob        Key data formatted in KeyBlob format
keyBlobSize     Size of exported pKeyBlob
pAddParams      Additional parameters
addParamsSize   Size of Additional parameters

DESCRIPTION

This function takes an internal key and exports it to the host to be
used by a host application. The key will be formatted in the specified
KeyBlob format. Currently additional parameters and their size are not
used and should be set to zero.

Note: KeyBlob formatting can be found in the MUSCLE Cryptographic Card
Edge Definition Section 2.2.

RETURN VALUES

Reference previously defined error codes.

EXAMPLES

MSCTokenInfo tokenList[16];  // 16 used as example
MSCTokenConnection pConnection;
MSCUChar8 myKeyBlob[1000];
MSCPULong32 myKeyBlobSize;
MSC_RV rv; MSCPULong32 listSize = 16;

rv = MSCListTokens( MSC_LIST_KNOWN, tokenList, &listSize );
if (rv == MSC_SUCCESS) {
    rv = MSCEstablishConnection( &tokenList[0], MSC_SHARE_SHARED, NULL,
                                   NULL, &pConnection );
    if (rv == MSC_SUCCESS) {
        myKeyBlobSize = sizeof(myKeyBlob);
        rv = MSCExportKey(&pConnection, 1, myKeyBlob,
                           &myKeyBlobSize, 0, 0);
    }
}

SEE ALSO

MSCListTokens, MSCEstablishConnection, MSCReleaseConnection,
MSCBeginTransaction, MSCEndTransaction, MSCImportKey, MSCExportKey,
MSCListKeys, MSCExtAuthenticate, MSCComputeCrypt
NAME

MSCComputeCrypt - Execute a cryptographic function on the card

SYNOPSIS

#include <musclecard.h>

MSCComputeCrypt(
    MSCLPTokenConnection   pConnection,
    MSCLPCryptInit   cryptInit,
    MSCPUChar8    pInputData,
    MSCULong32    inputDataSize,
    MSCPUChar8    pOutputData,
    MSCPULong32    outputDataSize
);

PARAMETERS

pConnection  Handle to a previously connected session
cryptInit       Structure which contains key/crypto options
pInputData      Input data to the function
inputDataSize   Size of Input Data
pOutputData  Output data from the function
outputDataSize  Size of Output Data

DESCRIPTION

This function uses an internal key and performs a cryptographic
operation with it. Data is fed into the function through pInputData
and data comes out in pOutputData. This function is responsible for
digital signatures, encryptions, and decryptions with all types of
supported keys.

Note: Setting all MSCKeyPolicy fields to zero will result in no key
policy for that particular key. In many instances the service provider
does not support key policies - use MSCGetCapabilities to see if any of
the capabilities are supported.

cryptInit is a structure containing the following fields:

    keyNum                Key number to use
    cipherMode            Mode of the cipher
    cipherDirection       Direction of the cipher
    optParams           Optional parameters
    optParamsSize        Optional parameters size

    cryptInit.keyNum
    Any available key number

    cryptInit.cipherMode
    MSC_MODE_RSA_NOPAD    Use RSA and don't pad
    MSC_MODE_DSA_SHA    Use DSA with SHA
    MSC_MODE_DES_CBC_NOPAD   Use DES in CBC mode
    MSC_MODE_DES_ECB_NOPAD  Use DES in ECB mode

    cryptInit.cipherDirection
    MSC_DIR_SIGN    Perform a signing operation
    MSC_DIR_VERIFY   Verify a signature
    MSC_DIR_ENCRYPT Encrypt the data
MSC_DIR_DECRYPT  
Decrypt the data

cryptInit.optParams
Optional parameters  (RFU)

cryptInit.optParamsSize
Optional parameters size  (RFU)

RETURN VALUES
Reference previously defined error codes.

EXAMPLES
MSCTokenInfo tokenList[16];  // 16 used as example
MSCTokenConnection pConnection;
MSCCryptInit myCrypt;
MSCUChar8 inData[512], outData[512];
MSCULong32 outSize;
MSC_RV rv; MSCULong32 listSize = 16;

rv = MSCListTokens( MSC_LIST_KNOWN, tokenList, &listSize );
if (rv == MSC_SUCCESS) {
    rv = MSCEstablishConnection( &tokenList[0], MSC_SHARE_SHARED, NULL,
                               NULL, &pConnection );
    if (rv == MSC_SUCCESS)
        {
            myCrypt.keyNum          = 1;
            myCrypt.cipherMode      = MSC_MODE_RSA_NO_PAD;
            myCrypt.cipherDirection = MSC_DIR_SIGN;
            myCrypt.optParams       = 0;
            myCrypt.optParamsSize   = 0;

            rv = MSCComputeCrypt(&pConnection, &myCrypt, inData,
                                  sizeof(inData), outData, &outSize);
            if (rv == MSC_SUCCESS)
                {
                    ...
                }
        }
}

SEE ALSO
MSCListTokens, MSCEstablishConnection, MSCReleaseConnection,
MSCBeginTransaction, MSCEndTransaction, MSCImportKey, MSCExportKey,
MSCListKeys, MSCExtAuthenticate, MSCComputeCrypt
NAME

MSCExtAuthenticate - Authenticate the host to the card.

SYNOPSIS

#include <musclecard.h>

MSCExtAuthenticate(
    MSCLPTokenConnection   pConnection,
    MSCUChar8                keyNum,
    MSCUChar8                cipherMode,
    MSCUChar8                cipherDirection,
    MSCFPChar8               pData,
    MSCULong32                dataSize
);

PARAMETERS

pConnection  Handle to a previously connected session
keyNum      Key number for operation
cipherMode Cipher mode to use
cipherDirection Direction of the cipher
pData       Data presented to the card
dataSize    Size of pData

DESCRIPTION

This function authenticates the host to the card. When the host calls a
GetChallenge it can present this value back to the card ciphered with a
particular key. The card will use an internal key to decipher the
data presented to the card and determine whether the host is validated.

cipherMode

MSC_MODE_RSA_NO_PAD   Use RSA and don't pad
MSC_MODE_DSA_SHA    Use DSA with SHA
MSC_MODE_DES_CBC_NOPAD   Use DES in CBC mode
MSC_MODE_DES_ECB_NOPAD  Use DES in ECB mode

cipherDirection

MSC_DIR_SIGN          Perform a signing operation
MSC_DIR_VERIFY   Verify a signature
MSC_DIR_ENCRYPT   Encrypt the data
MSC_DIR_DECRYPT   Decrypt the data

RETURN VALUES

Reference previously defined error codes.

EXAMPLES

MSCTokenInfo tokenList[16]; // 16 used as example
MSCTokenConnection pConnection;
MSCryptInit myCrypt;
MSCUChar8 seedData[20], randomData[20];
MSCUChar8 cipherData[20];
MSCULong32 outSize;
MSC_RV rv; MSCULong32 listSize = 16;

rv = MSCLSTokenInfo( MSC_LIST_KNOWN, tokenList, &listSize );
if (rv == MSC_SUCCESS) {
rv = MSCEstablishConnection( &tokenList[0], MSC_SHARE_SHARED, NULL,
                             NULL, &pConnection );
if (rv == MSC_SUCCESS)
{
    MSCGetChallenge( pConnection, seedData, 0, randomData, 8 );
    /* The following function is for demo only */
    rv = des_cbc_encrypt(randomData, cipherData);
    rv = MSCExtAuthenticate(&pConnection, 1, MODE_DES_ECB_NOPAD,
                             MSC_DIR_ENCRYPT, cipherData, 8);
    if (rv == MSC_SUCCESS)
    {
        ...
    }
}

SEE ALSO
MSCListTokens, MSCEstablishConnection, MSCReleaseConnection,
MSCBeginTransaction, MSCEndTransaction, MSCImportKey, MSCExportKey,
MSCListKeys, MSCExtAuthenticate, MSCComputeCrypt, MSCGetChallenge
NAME

MSCGetKeyAttributes  -  Gets a key’s attributes

SYNOPSIS
#include <musclecard.h>

MSCGetObjectAttributes(
    MSCLPTokenConnection  pConnection,
    MSCUChar8             keyNumber,
    MSCLPKeyInfo          pKeyInfo
);

PARAMETERS
pConnection   Handle to a previously connected session
keyNumber   Number of the key to find
pKeyInfo   Structure holding key information

DESCRIPTION
This function returns information about a particular key including
its type, size, policy, and Access Control List (ACL).

RETURN VALUES
Reference previously defined error codes.

EXAMPLES
MSCTokenInfo tokenList[16];  // 16 used as example
MSCTokenConnection pConnection;
MSCKeyInfo keyInfo;
MSC_RV rv; MSCULong32 listSize = 16;

rv = MSCListTokens( MSC_LIST_KNOWN, tokenList, &listSize );
if (rv == MSC_SUCCESS)  {
    rv = MSCEstablishConnection( &tokenList[0], MSC_SHARE_SHARED, NULL,
                                NULL, &pConnection );
    if (rv == MSC_SUCCESS)  {
        rv = MSCGetKeyAttributes(&pConnection, 1, &keyInfo);
        if (rv == MSC_SUCCESS)
        {
            ...
        }
    }
}

SEE ALSO
MSCListTokens, MSCEstablishConnection, MSCReleaseConnection,
MSCBeginTransaction, MSCEndTransaction, MSCImportKey, MSCExportKey,
MSCListKeys, MSCExtAuthenticate, MSCComputeCrypt, MSCGetChallenge
NAME

MSCListKeys - Lists the currently available keys

SYNOPSIS

#include <musclecard.h>

MSCListKeys(
    MSCLPTokenConnection pConnection,
    MSCUChar8 seqOption,
    MSCLPKeyInfo pKeyInfo
);

PARAMETERS

pConnection         Handle to a previously connected session
seqOption         Sequence option
pKeyInfo      Returned key information

DESCRIPTION

This function returns structures of keys that are available on the card.
Each time this function is called it will return the next key structure
in the list until MSC_SEQUENCE_END is returned. At anytime seqOption can
be declared as MSC_SEQUENCE_RESET to return to the beginning of the list.

seqOption:
    MSC_SEQUENCE_RESET       Get the first item in the list
    MSC_SEQUENCE_NEXT        Get the next item in the list

pKeyInfo is a structure containing the following fields:
keyNum         Key number
keyType         Type of key
keySize         Size of key
keyPolicy   Key usage policy
keyACL          Access Control List (ACL) of key

RETURN VALUES

Reference previously defined error codes.

EXAMPLES

MSCTokenInfo tokenList[16]; // 16 used as example
MSCTokenConnection pConnection;
MSCKeypInfo keyData; MSC_RV rv; MSCULong32 listSize = 16;
rv = MSCListTokens( MSC_LIST_KNOWN, tokenList, &listSize );
rv = MSCEstablishConnection( &tokenList[0], MSC_SHARE_SHARED, NULL,
                             NULL, &pConnection );
rv = MSCListKeys(&pConnection, MSC_SEQUENCE_RESET, &keyData);
do {  
    rv = MSCListKeys(&pConnection, MSC_SEQUENCE_NEXT, &keyData);
} while ( rv == MSC_SUCCESS );

SEE ALSO

MSCImportKey, MSCExportKey, MSCListKeys, MSCExtAuthenticate,
MSCComputeCrypt
NAME

MSCCreatePIN - Create a PIN

SYNOPSIS
#include <musclecard.h>

MSCCreatePIN(
    MSCLPTokenConnection pConnection,
    MSCUChar8                pinNum,
    MSCUChar8                pinAttempts,
    MSCPUChar8               pPinCode,
    MSCULong32                pinCodeSize,
    MSCPUChar8               pUnblockCode,
    MSCUChar8                unblockCodeSize
);

PARAMETERS
    pConnection    Handle to a previously connected session
    pinNum        Number to identify PIN (1-7)
    pinAttempts   Number of bad tries until PIN blocks
    pPinCode      The PIN code
    pinCodeSize   Size of PIN code
    pUnblockCode  The Unblock code
    unblockCodeSize Size of Unblock code

DESCRIPTION
    This function creates a PIN on the card which can be used when
authenticating to object, keys, and other functions. The PIN
has an associated Unblock PIN in case the PIN is blocked from
bad entries.

RETURN VALUES
    Reference previously defined error codes.

EXAMPLES
    MSCTokenInfo tokenList[16]; // 16 used as example
    MSCTokenConnection pConnection;
    MSCUChar8 pinCode[] = "00000000";
    MSCUChar8 unbCode[] = "11111111";
    MSC_RV rv; MSCULong32 listSize = 16;
    rv = MSCListTokens( MSC_LIST_KNOWN, tokenList, &listSize );
    if (rv == MSC_SUCCESS) {
        rv = MSCEstablishConnection( &tokenList[0], MSC_SHARE_SHARED, NULL,
                                      NULL, &pConnection );
        if (rv == MSC_SUCCESS) {
            rv = MSCCreatePIN(&pConnection, 1, 5, pinCode, strlen(pinCode),
                               unbCode, strlen(unbCode));
        }
    }

SEE ALSO
    MSCCreatePIN, MSCChangePIN, MSCUnblockPIN, MSCListPINs
NAME

MSCVerifyPIN - Verify a PIN

SYNOPSIS

#include <musclecard.h>

MSCVerifyPIN(
    MSCLPTokenConnection pConnection,
    MSCUChar8   pinNum,
    MSCPUChar8   pPinCode,
    MSCULong32   pinCodeSize
);

PARAMETERS

pConnection  Handle to a previously connected session
pinNum   PIN identifier
pPinCode   PIN code to verify
pinCodeSize  Size of PIN code

DESCRIPTION

This function verifies a PIN in order to gain authentication
priveledges to perform a particular function.

RETURN VALUES

Reference previously defined error codes.

EXAMPLES

MSCTokenInfo tokenList[16];  // 16 used as example
MSCTokenConnection pConnection;
MSCUChar8 pinCode[] = "00000000";
MSC_RV rv; MSCULong32 listSize = 16;

rv = MSCListTokens( MSC_LIST_KNOWN, tokenList, &listSize );
if (rv == MSC_SUCCESS) {
    rv = MSCEstablishConnection( &tokenList[0], MSC_SHARE_SHARED, NULL,
                                    NULL, &pConnection );
    if (rv == MSC_SUCCESS) {
        rv = MSCVerifyPIN(&pConnection, 1, pinCode, 8);
        if (rv == MSC_SUCCESS) {
            ...
        }
    }
}

SEE ALSO

MSCCreatePIN, MSCChangePIN, MSCUnblockPIN, MSCListPINs
NAME

MSCChangePIN - Change an existing PIN

SYNOPSIS

#include <musclecard.h>

MSCChangePIN(
    MSCLPTokenConnection pConnection,
    MSCUChar8   pinNum,
    MSCPUChar8   pOldPinCode,
    MSCUChar8                oldPinCodeSize,
    MSCPUChar8               pNewPinCode,
    MSCUChar8   newPinCodeSize
    );

PARAMETERS

pConnection Handle to a previously connected session
pinNum PIN identifier
pOldPinCode The old PIN code
oldPinCodeSize Size of old PIN
pNewPinCode The new PIN code
newPinCodeSize Size of new PIN

DESCRIPTION

This function changes an existing PIN on the card which can be
used when authenticating to object, keys, and other functions.
The new PIN replaces the old PIN.

RETURN VALUES

Reference previously defined error codes.

EXAMPLES

MSCTokenInfo tokenList[16]; // 16 used as example
MSCTokenConnection pConnection;
MSCUChar8 pinCode[] = “00000000”;
MSCUChar8 newCode[] = “11111111”;
MSC_RV rv; MSCULong32 listSize = 16;

rv = MSCListTokens( MSC_LIST_KNOWN, tokenList, &listSize );
if (rv == MSC_SUCCESS) {
    rv = MSCEstablishConnection( &tokenList[0], MSC_SHARE_SHARED, NULL,
                                NULL, &pConnection );
    if (rv == MSC_SUCCESS) {
        rv = MSCChangePIN(&pConnection, 1, pinCode, 8, newCode, 8);
        if (rv == MSC_SUCCESS) {
            ...
        }
    }
}

SEE ALSO

MSCCreatePIN, MSCChangePIN, MSCUnblockPIN, MSCListPINs
NAME

MSCUnblockPIN - Unblocked a previously blocked PIN

SYNOPSIS

#include <musclecard.h>

MSCUnblockPIN(  
    MSCLPTokenConnection     pConnection,  
    MSCUChar8                pinNum,  
    MSCPUChar8               pUnblockCode,  
    MSCULong32               unblockCodeSize
    );

PARAMETERS

pConnection  Handle to a previously connected session
pinNum   PIN identifier
pUnblockCode  Unblock code to verify
unblockCodeSize  Size of Unblock code

DESCRIPTION

This function unblocks a previously blocked PIN identified by pinNum.  
Upon success of this function the PIN will no longer be blocked and  
will reset its number of attempts.

RETURN VALUES

Reference previously defined error codes.

EXAMPLES

MSCTokenInfo tokenList[16]; // 16 used as example
MSCTokenConnection pConnection;
MSCUChar8 unbCode[] = “00000000”;
MSC_RV rv; MSCULong32 listSize = 16;

rv = MSCListTokens( MSC_LIST_KNOWN, tokenList, &listSize );
if (rv == MSC_SUCCESS) {
    rv = MSCEstablishConnection( &tokenList[0], MSC_SHARE_SHARED, NULL,  
                                  NULL, &pConnection );
    if (rv == MSC_SUCCESS) {
        rv = MSCUnblockPIN(&pConnection, 1, unbCode, 8);
        if (rv == MSC_SUCCESS) {
            ...
        }
    }
}

SEE ALSO

MSCCreatePIN, MSCChangePIN, MSCUnblockPIN, MSCListPINs
NAME

MSCListPINs - List the currently available PINs

SYNOPSIS

#include <musclecard.h>

MSCListPINs(
    MSCLPTokenConnection pConnection,
    MSCPUShort16   pPinBitMask
);

PARAMETERS

pConnection  Handle to a previously connected session
pPinBitMask  Bitmask of currently available PINS

DESCRIPTION

This function returns a bitmask of the currently available PINs.
For example a bitmask of 0x0003 would denote the existence of
PINs 1 and 2.

RETURN VALUES

Reference previously defined error codes.

EXAMPLES

MSCTokenInfo tokenList[16];  // 16 used as example
MSCTokenConnection pConnection;
MSCUShort16 pinMask;
MSC_RV rv; MSCULong32 listSize = 16;

rv = MSCListTokens( MSC_LIST_KNOWN, tokenList, &listSize );
if (rv == MSC_SUCCESS) {
    rv = MSCEstablishConnection( &tokenList[0], MSC_SHARE_SHARED, NULL,
                                  NULL, &pConnection );
    if (rv == MSC_SUCCESS)
        rv = MSCListPINs(&pConnection, &pinMask);
    if (rv == MSC_SUCCESS)
        ...
}

SEE ALSO

MSCCreatePIN, MSCChangePIN, MSCUnblockPIN, MSCListPINs
NAME

MSCCreateObject - Creates an object on the card

SYNOPSIS

#include <musclecard.h>

MSCCreateObject(
    MSCLPTokenConnection pConnection,
    MSCString objectID,
    MSCULong32 objectSize,
    MSCLPObjectACL pObjectACL
);

PARAMETERS

pConnection Handle to a previously connected session
objectID Name for the object
objectSize 16 bit size of the object
pObjectACL Access Control List (ACL) of the object

DESCRIPTION

This function creates an empty object on the smartcard of variable size
with a string identifier denoted by objectID. The object can then be
read and written to to store and retrieve data.

pObjectACL
    readPermission Bitwise 'OR' with defined ACL values
    writePermission Bitwise 'OR' with defined ACL values
    deletePermission Bitwise 'OR' with defined ACL values

RETURN VALUES

Reference previously defined error codes.

EXAMPLES

MSCTokenInfo tokenList[16]; // 16 used as example
MSCTokenConnection pConnection;
MSCObjectACL myACL;
MSC_RV rv; MSCULong32 listSize = 16;

rv = MSCListTokens( MSC_LIST_KNOWN, tokenList, &listSize );
if (rv == MSC_SUCCESS) {
    rv = MSCEstablishConnection( &tokenList[0], MSC_SHARE_SHARED, NULL,
                                 NULL, &pConnection );
    if (rv == MSC_SUCCESS) {
        myACL.readPermission   = MSC_AUT_ANY;
        myACL.writePermission  = MSC_AUT_ANY;
        myACL.deletePermission = MSC_AUT_NONE;
        rv = MSCCreateObject(&pConnection, "c1", 500, &myACL);
    }
}

SEE ALSO

MSCCreateObject, MSCDeleteObject, MSCWriteObject, MSCReadObject,
MSCListObjects, MSCGetObjectAttributes, MSCReadAllocateObject
NAME

MSCDeleteObject - Deletes an object on the card

SYNOPSIS
#include <musclecard.h>

MSCDeleteObject(
    MSCLPTokenConnection pConnection,
    MSCString   objectID,
    MSCUChar8   zeroFlag
);

PARAMETERS
pConnection  Handle to a previously connected session
objectID   Name for the object
zeroFlag   Flag to denote zeroing the object

DESCRIPTION
This function deletes an object identified by objectID that is located on the card. The zeroFlag is provided to overwrite the object with zeros upon deletion.

zeroFlag
MSC_ZF_DEFAULT  Leave object data as is
MSC_ZF_WRITE_ZERO  Write zeros to object

RETURN VALUES
Reference previously defined error codes.

EXAMPLES
MSCTokenInfo tokenList[16];  // 16 used as example
MSCTokenConnection pConnection;
MSC_RV rv; MSCULong32 listSize = 16;

rv = MSCListTokens( MSC_LIST_KNOWN, tokenList, &listSize );
if (rv == MSC_SUCCESS) {
    rv = MSCEstablishConnection( &tokenList[0], MSC_SHARE_SHARED, NULL,
                                 NULL, &pConnection );
    if (rv == MSC_SUCCESS)
    {
        rv = MSCDeleteObject(&pConnection, "c1", MSC_ZF_DEFAULT);
        if (rv == MSC_SUCCESS)
        {
            ...
        }
    }
}

SEE ALSO
MSCCreateObject, MSCDeleteObject, MSCWriteObject, MSCReadObject,
MSCListObjects, MSCGetObjectAttributes, MSCReadAllocateObject
NAME

MSCWriteObject - Writes data to an object on the card

SYNOPSIS

#include <musclecard.h>

MSCWriteObject(
    MSCLPTokenConnection pConnection,
    MSCString   objectID,
    MSCULong32   offset,
    MSCPUChar8   pInputData,
    MSCULong32   dataSize,
    LPRWEventCallback  rwCallback,
    MSCPVoid32   addParams
);

PARAMETERS

pConnection Handle to a previously connected session
objectID   Name for the object
offset   Offset to write data
pInputData  Data to write
dataSize   Amount of data to write
rwCallback  Callback function (optional)
addParams   Additional parameters for callback (optional)

DESCRIPTION

This function writes to an object specified by objectID. The calling application must have completed all necessary authentications before calling this function. This function allows the application to write any amount of data with an optional callback function. Both the callback function and additional parameters may be NULL if not needed.

RETURN VALUES

Reference previously defined error codes.

EXAMPLES

MSCTokenInfo tokenList[16]; // 16 used as example
MSCTokenConnection pConnection;
MSCUChar8 myData[] = {1,2,3,4,5,6,7,8};
MSC_RV rv; MSCULong32 listSize = 16;

rv = MSCListTokens( MSC_LIST_KNOWN, tokenList, &listSize );
if (rv == MSC_SUCCESS) {
    rv = MSCEstablishConnection( &tokenList[0], MSC_SHARE_SHARED, NULL,
        NULL, &pConnection );
    if (rv == MSC_SUCCESS) {
        rv = MSCWriteObject(&pConnection, "c1", 0, myData, 8);
        if (rv == MSC_SUCCESS) {
            ...
        }
    }
}

SEE ALSO

MSCCreateObject, MSCDeleteObject, MSCWriteObject, MSCReadObject,
MSCListObjects, MSCGetObjectAttributes, MSCReadAllocateObject
NAME

MSCReadObject - Reads data from an object on the card

SYNOPSIS

#include <musclecard.h>

MSCReadObject(
    MSCLPTokenConnection pConnection,
    MSCString objectID,
    MSCULong32 offset,
    MSCPUChar8 pOutputData,
    MSCULong32 dataSize
    LPRWEventCallback rwCallback,
    MSCPVoid32 addParams
);

PARAMETERS

pConnection    Handle to a previously connected session
objectID       Name for the object
offset         Offset to read data
pInputData     Data read
dataSize        Size of data to be read
rwCallback     Callback function (optional)
addParams      Additional parameters for callback (optional)

DESCRIPTION

This function reads an object specified by objectID. The calling application must have completed all necessary authentications before calling this function. This function allows the application to read any amount of data with an optional callback function. Both the callback function and additional parameters may be NULL if not needed.

RETURN VALUES

Reference previously defined error codes.

EXAMPLES

MSCTokenInfo tokenList[16]; // 16 used as example
MSCTokenConnection pConnection;
MSCUChar8 myData[8];
MSC_RV rv; MSCULong32 listSize = 16;
rv = MSCListTokens( MSC_LIST_KNOWN, tokenList, &listSize );
if (rv == MSC_SUCCESS) {
    rv = MSCEstablishConnection( &tokenList[0], MSC_SHARE_SHARED, NULL,
                                  NULL, &pConnection );
    if (rv == MSC_SUCCESS) {
        rv = MSCReadObject(&pConnection, "c1", 0, myData, 8);
        if (rv == MSC_SUCCESS) {
            ...
        }
    }
}

SEE ALSO

MSCCreateObject, MSCDeleteObject, MSCWriteObject, MSCReadObject,
MSCListObjects, MSCGetObjectAttributes, MSCReadAllocateObject
NAME

MSCReadAllocateObject - Reads and allocates array to fill read

SYNOPSIS

#include <musclecard.h>

MSCReadAllocateObject(
    MSCLPTokenConnection   pConnection,
    MSCString              objectID,
    MSCPUChar8             *pOutputData,
    MSCPULong32            dataSize
    LPRWEventCallback       rwCallback,
    MSCPVoid32             addParams
);  

PARAMETERS

pConnection        Handle to a previously connected session
objectID           Name for the object
pOutputData        Data to read
dataSize           Amount of data read
rwCallback         Callback function (optional)
addParams          Additional parameters for callback (optional)

DESCRIPTION

This function reads from an object specified by objectID. The calling application must have completed all necessary authentications before calling this function. This function automatically calculates the size of the object, allocates pOutputData, writes the object to pOutputData, and returns the size in dataSize. The calling application must free this allocated data when finished with it. Both the callback function and additional parameters may be NULL if not needed.

RETURN VALUES

Reference previously defined error codes.

EXAMPLES

MSCTokenInfo tokenList[16];  // 16 used as example
MSCTokenConnection pConnection;
MSCPUChar8 outBuffer;
MSCULong32 objSize;
MSC_RV rv; MSCULong32 listSize = 16;
rv = MSCListTokens( MSC_LIST_KNOWN, tokenList, &listSize );
if (rv == MSC_SUCCESS) {
    rv = MSCEstablishConnection( &tokenList[0], MSC_SHARE_SHARED, NULL,
                          NULL, &pConnection );
    if (rv == MSC_SUCCESS) {
        rv = MSCReadAllocateObject(&pConnection, "c1", &outBuffer,
                              &objSize);
        if (rv == MSC_SUCCESS) {
            ...
            free(outBuffer); /* Important */
        }
    }
}
SEE ALSO
MSCCreateObject, MSCDeleteObject, MSCWriteObject, MSCReadObject,
MSCListObjects, MSCGetObjectAttributes, MSCReadAllocateObject
NAME

MSCGetObjectAttributes - Gets an objects attributes

SYNOPSIS
#include <musclecard.h>

MSCGetObjectAttributes(
    MSCLPTokenConnection    pConnection,
    MSCString                objectID,
    MSCLPObjectInfo  pObjectInfo
);

PARAMETERS
pConnection    Handle to a previously connected session
objectID       Name of the object to find
pObjectInfo    Structure holding object information

DESCRIPTION
This function returns information about a particular object including
its size and Access Control List (ACL).

RETURN VALUES
Reference previously defined error codes.

EXAMPLES
MSCTokenInfo tokenList[16]; // 16 used as example
MSCTokenConnection pConnection;
MSCObjectInfo objInfo;
MSC_RV rv; MSCULong32 listSize = 16;

rv = MSCListTokens( MSC_LIST_KNOWN, tokenList, &listSize );
if (rv == MSC_SUCCESS) {
    rv = MSCEstablishConnection( &tokenList[0], MSC_SHARE_SHARED, NULL,
                                  NULL, &pConnection );
    if (rv == MSC_SUCCESS) {
        rv = MSCGetObjectAttributes(&pConnection, “c1”, &objInfo);
        if (rv == MSC_SUCCESS) {
            ...
        }
    }
}

SEE ALSO
MSCCreateObject, MSCDeleteObject, MSCWriteObject, MSCReadObject,
MSCListObjects, MSCGetObjectAttributes, MSCReadAllocateObject
NAME

MSCListObjects - Lists the currently available objects

SYNOPSIS
#include <musclecard.h>

MSCListObjects(
    MSCLPTokenConnection  pConnection,
    MSCUChar8                seqOption,
    MSCLPObjectInfo  pObjectInfo
);

PARAMETERS
pConnection Handle to a previously connected session
seqOption Sequence option
pObjectInfo Returned object information

DESCRIPTION
This function returns structures of objects that are available on the card.
Each time this function is called it will return the next object structure
in the list until MSC_SEQUENCE_END is returned. At anytime seqOption can
be declared as SEQUENCE_RESET to return to the beginning of the list.

seqOption:
MSC_SEQUENCE_RESET Get the first item in the list
MSC_SEQUENCE_NEXT Get the next item in the list

pObjectInfo is a structure containing the following fields:
objectID ID of the object
objectSize Size of the object
objectACL Objects Access Control List (ACL)

RETURN VALUES
Reference previously defined error codes.

EXAMPLES
MSCTokenInfo tokenList[16]; // 16 used as example
MSCTokenConnection pConnection;
MSCObjectInfo objData;
MSC_RV rv; MSCULong32 listSize = 16;

rv = MSCListTokens( MSC_LIST_KNOWN, tokenList, &listSize );
rv = MSCEstablishConnection( &tokenList[0], MSC_SHARE_SHARED, NULL,
                           NULL, &pConnection );
rv = MSCListObjects(&pConnection, MSC_SEQUENCE_RESET, &objData);
do {
    rv = MSCListObjects(&pConnection, MSC_SEQUENCE_NEXT, &objData);
} while ( rv == MSC_SUCCESS );

SEE ALSO
MSCCreateObject, MSCReadAllocateObject, MSCGetObjectAttributes,
MSCDeleteObject, MSCWriteObject, MSCReadObject, MSCListObjects
NAME

MSCLogoutAll - Logs out all logged identities

SYNOPSIS

```
#include <musclecard.h>

MSCLogoutAll(
    MSCLPTokenConnection pConnection,
);
```

PARAMETERS

- `pConnection`: Handle to a previously connected session

DESCRIPTION

This function logs out all logged identities. Any PINs, or external authentications previously made will no longer hold merit after this function call.

RETURN VALUES

Reference previously defined error codes.

EXAMPLES

```
MSCTokenInfo tokenList[16]; // 16 used as example
MSCTokenConnection pConnection;
MSC_RV rv; MSCULong32 listSize = 16;

rv = MSCListTokens( MSC_LIST_KNOWN, tokenList, &listSize );
if (rv == MSC_SUCCESS) {
    rv = MSCEstablishConnection( &tokenList[0], MSC_SHARE_SHARED, NULL, NULL, &pConnection );
    if (rv == MSC_SUCCESS)
    {
        rv = MSCLogoutAll(&pConnection);
        if (rv == MSC_SUCCESS)
        {
            ...
        }
    }
}
```

SEE ALSO

MSCListTokens, MSCEstablishConnection, MSCReleaseConnection, MSCBeginTransaction, MSCEndTransaction, MSCVerifyPIN, MSCExtAuthenticate, MSCLogoutAll
NAME

MSCGetChallenge - Retrieve a random number from the card

SYNOPSIS

#include <musclecard.h>

MSCGetChallenge(
    MSCLPTokenConnection pConnection,
    MSCPUChar8 pSeed,
    MSCUShort16 seedSize,
    MSCPUChar8 pRandomData,
    MSCUShort16 randomDataSize
);

PARAMETERS

pConnection Handle to a previously connected session
pSeed Seed to inject into random algorithm
seedSize Size of the seed
pRandomData Random data from the card
randomDataSize Amount of random data requested

DESCRIPTION

This function requests a random number from the card which can be used for many purposes including the verify an authentication using the MSCExtAuthenticate function. A seed may be presented into pSeed. A seedSize of zero denotes no seed presented.

RETURN VALUES

Reference previously defined error codes.

EXAMPLES

MSCTokenInfo tokenList[16]; // 16 used as example
MSCTokenConnection pConnection;
MSCPUChar8 randomData[8];
MSC_RV rv; MSCULong32 listSize = 16;

rv = MSCListTokens( MSC_LIST_KNOWN, tokenList, &listSize );
if (rv == MSC_SUCCESS) {
    rv = MSCEstablishConnection( &tokenList[0], MSC_SHARE_SHARED, NULL, NULL, &pConnection );
    if (rv == MSC_SUCCESS)
    {
        rv = MSCGetChallenge(&pConnection, NULL, 0, randomData, 8);
        if (rv == MSC_SUCCESS)
        {
            ...
        }
    }
}

SEE ALSO

MSCListTokens, MSCEstablishConnection, MSCReleaseConnection, MSCBeginTransaction, MSCEndTransaction, MSCExtAuthenticate
CAPABILITY DEFINITIONS

The following contains Tags and the available responses which can be retrieved from the MSCGetCapabilities function. The returned data size will be contained in the brackets []. For example MSC_TAG_SUPPORT_FUNCTIONS [4] means the tag name is MSC_TAG_SUPPORT_FUNCTIONS and it returns 4 bytes. Each Tag is **Bold** and its members will be values which can be tested by a bitmask test to determine whether that feature is supported. If a particular function of crypto algorithm is not supported, any further tags related to that unsupported feature do not have to be defined. If a Tag is not defined, MSC_INVALID_PARAMETER will be returned and the application can assume the feature is not supported. All data requiring more than one byte are stored in the host’s byte order so that typecasts may be used. For Tags which do not return bitmasks, it will be listed as to what is returned.

**MSC_TAG_SUPPORT_FUNCTIONS [4]**

This tag returns a bitmask of the functions supported by this token provider. The functions are listed below:

MSC_SUPPORT_GENKEYS -Supports MSCGenerateKeys
MSC_SUPPORT_IMPORTKEY -Supports MSCImportKey
MSC_SUPPORT_EXPORTKEY -Supports MSCExportKey
MSC_SUPPORT_COMPUTECRYPTO -Supports MSCComputeCrypt
MSC_SUPPORT_EXTAUTH -Supports MSCExternalAuth
MSC_SUPPORT_LISTKEYS -Supports MSCListKeys
MSC_SUPPORT_CREATEPIN -Supports MSCCreatePIN
MSC_SUPPORT_VERIFYPIN -Supports MSCVerifyPIN
MSC_SUPPORT_CHANGEPIN -Supports MSCChangePIN
MSC_SUPPORT_UNBLOCKPIN -Supports MSCUnblockPIN
MSC_SUPPORT_LISTPINS -Supports MSCListPINS
MSC_SUPPORT_CREATEOBJECT -Supports MSCCreateObject
MSC_SUPPORT_DELETEOBJECT -Supports MSCDeleteObject
MSC_SUPPORT_WRITEOBJECT -Supports MSCWriteObject
MSC_SUPPORT_READOBJECT -Supports MSCReadObject
MSC_SUPPORT_LISTOBJECTS -Supports MSCListObjects
MSC_SUPPORT_LOGOUTALL -Supports MSCLogoutAll
MSC_SUPPORT_GETCHALLENGE -Supports MSCGetChallenge

**MSC_TAG_SUPPORT_CRYPTOALG [4]**

This tag returns a bitmask of the supported crypto and digest algorithms which are listed below:

MSC_SUPPORT_RSA -Supports the RSA algorithm
MSC_SUPPORT_DSA -Supports the DSA algorithm
MSC_SUPPORT_ELGAMAL -Supports the ElGamal algorithm
MSC_SUPPORT_DES -Supports the DES algorithm
MSC_SUPPORT_3DES -Supports the Triple DES algorithm
MSC_SUPPORT_IDEA -Supports the IDEA algorithm
MSC_SUPPORT_AES -Supports the AES algorithm
MSC_SUPPORT_BLOWFISH -Supports the Blowfish algorithm
MSC_SUPPORT_TWOFISH -Supports the Twofish algorithm
MSC_SUPPORT_SHA1 -Supports the SHA1 algorithm
MSC_SUPPORT_MD5 - Supports the MD5 algorithm

MSC_TAG_CAPABLE_KEY_AUTH [2]
This tag returns the Access Control List (ACL) required to import or generate keys. In this case an ACL consists of one SHORT.

MSC_TAG_CAPABLE_RSA [4]
This tag returns a bitmask of the supported features available to the RSA algorithm as defined below:

MSC_CAPABLE_RSA_512 - Supports 512 bit RSA
MSC_CAPABLE_RSA_768 - Supports 768 bit RSA
MSC_CAPABLE_RSA_1024 - Supports 1024 bit RSA
MSC_CAPABLE_RSA_2048 - Supports 2048 bit RSA
MSC_CAPABLE_RSA_4096 - Supports 4096 bit RSA
MSC_CAPABLE_RSA_KEYGEN - Supports RSA key generation
MSC_CAPABLE_RSA_NOPAD - Supports RSA with no pad
MSC_CAPABLE_RSA_PKCS1 - Supports RSA with PKCS1 padding

MSC_TAG_CAPABLE_DSA [4]
This tag returns a bitmask of the supported features available to the DSA algorithm as defined below:

MSC_CAPABLE_DSA_512 - Supports 512 bit DSA
MSC_CAPABLE_DSA_768 - Supports 768 bit DSA
MSC_CAPABLE_DSA_1024 - Supports 1024 bit DSA
MSC_CAPABLE_DSA_2048 - Supports 2048 bit DSA
MSC_CAPABLE_DSA_4096 - Supports 4096 bit DSA
MSC_CAPABLE_DSA_KEYGEN - Supports DSA key generation

MSC_TAG_CAPABLE_DES [4]
This tag returns a bitmask of the supported features available to the DES algorithm as defined below:

MSC_CAPABLE_DES_KEYGEN - Supports DES key generation
MSC_CAPABLE_DES_CBC - Supports DES in CBC mode
MSC_CAPABLE_DES_EBC - Supports DES in EBC mode
MSC_CAPABLE_DES_ECB - Supports DES in ECB mode

MSC_TAG_CAPABLE_3DES [4]
This tag returns a bitmask of the supported features available to the Triple DES algorithm as defined below:

MSC_CAPABLE_3DES_KEYGEN - Supports Triple DES key generation
MSC_CAPABLE_3DES_3KEY - Supports 3 key Triple DES
MSC_CAPABLE_3DES_CBC - Supports Triple DES in CBC mode
MSC_CAPABLE_3DES_EBC - Supports Triple DES in EBC mode
MSC_CAPABLE_3DES_ECB - Supports Triple DES in ECB mode
MSC_TAG_CAPABLE_IDEA [4]

This tag returns a bitmask of the supported features available to the IDEA algorithm as defined below:

MSC_CAPABLE_IDEA_KEYGEN - Supports Triple DES key generation
MSC_CAPABLE_IDEA_CBC - Supports Triple DES in CBC mode
MSC_CAPABLE_IDEA_ECB - Supports Triple DES in ECB mode

MSC_TAG_CAPABLE_AES [4]

This tag returns a bitmask of the supported features available to the AES algorithm as defined below:

MSC_CAPABLE_AES_KEYGEN - Supports AES key generation
MSC_CAPABLE_AES_CBC - Supports AES in CBC mode
MSC_CAPABLE_AES_ECB - Supports AES in ECB mode

MSC_TAG_CAPABLE_OBJ_ATTR [4]

This tag returns a bitmask of the following object related attributes:

MSC_CAPABLE_OBJ_ZERO - Is capable of zeroing data on object deletion

MSC_TAG_CAPABLE_OBJ_IDSIZE [1]

This tag returns the size of an object ID. For example, it may return the number 4. This means it uses 4 byte object ID’s.

MSC_TAG_CAPABLE_OBJ_AUTH [2]

This tag returns the Access Control List (ACL) required to create objects. In this case an ACL consists of one SHORT.

MSC_TAG_CAPABLE_OBJ_MAXNUM [4]

This tag returns the maximum number of objects which may exist on the token.

MSC_TAG_CAPABLE_PIN_ATTR [4]

This tag returns a bitmask of the following pin related attributes.

MSC_CAPABLE_PIN_RESET - Unblock PIN reset’s the PIN to the default PIN.
MSC_CAPABLE_PIN_LEAVE - Unblock PIN leaves the PIN as it’s original value.

MSC_TAG_CAPABLE_PIN_MAXNUM [1]

This tag returns the maximum number of pins which may be on the token.
MSC_TAG_CAPABLE_PIN_MINSIZE [1]

This tag returns the minimum number of characters which may be used in a pin. For example, a return of 4 means you may have a minimum pin size of 4 characters.

MSC_TAG_CAPABLE_PIN_MAXSIZE [1]

This tag returns the maximum number of characters which may be used in a pin. For example, a return of 8 means you may have a maximum pin size of 8 characters.

MSC_TAG_CAPABLE_PIN_CHARSET [4]

This Tag returns a bitmask of the supported character set based on the pin policy set in the token:

- MSC_CAPABLE_PIN_A_Z        -Supports uppercase A-Z
- MSC_CAPABLE_PIN_a_z        -Supports lowercase a-z
- MSC_CAPABLE_PIN_0_9        -Supports numbers 0-9
- MSC_CAPABLE_PIN_SPACE      -Supports spaces
- MSC_CAPABLE_PIN_CALC       -Supports + - / * % .= (calculator chars)
- MSC_CAPABLE_PIN_NONALPHA   -Supports 101 key English keyboard chars

MSC_TAG_CAPABLE_PIN_POLICY [4]

This Tag returns a bitmask of the pin policy checking and requirement attributes used by the token when creating pins.

- MSC_CAPABLE_PIN_A_Z        -Must have uppercase A-Z
- MSC_CAPABLE_PIN_a_z        -Requires lowercase a-z
- MSC_CAPABLE_PIN_0_9        -Requires numbers 0-9
- MSC_CAPABLE_PIN_NONALPHA   -Requires non-alphanumeric
- MSC_CAPABLE_PIN_HISTORY    -Checks pin history

MSC_TAG_CAPABLE_PIN_AUTH [2]

This tag returns the Access Control List (ACL) required to create pins. In this case an ACL consists of one SHORT.

MSC_TAG_CAPABLE_ID_STATE [1]

This Tag returns a bitmask of one value. A token can keep ID state when it can keep track of whether a pin or other id has been logged. A token with this capability will be able to return its logged ID’s with the GetStatus function.

- MSC_CAPABLE_ID_STATE    -Token maintains id state

MSC_TAG_CAPABLE_RANDOM [4]

This Tag returns a bitmask of capabilities of the on-board random number generation.

- MSC_CAPABLE_RANDOM_SEED    -Uses input of seed
MSC_TAG_CAPABLE_RANDOM_MAX [1]

This tag returns the maximum number of bytes which may be returned from the random number generator.

MSC_TAG_CAPABLE_RANDOM_MIN [1]

This tag returns the minimum number of bytes which may be returned from the random number generator.