SCM
MICROSYSTEMS

MCard DLL API Specification

Version No 1.9

Version History

Date	Version	Description of Changes	Author
Sep 06, 2002	1.0	Initial version	Simon Peter
Feb 17, 2003	1.1	All references to Infenion have been replaced with Infineon.	Simon Peter
Feb 20, 2003	1.2	The page references in the contents section were corrected	Simon Peter
Feb 22, 2003	1.3	Added section that lists the devices supported by this API	Calai Bhoopathi
May 13, 2003	1.4	Sec 9.1.1 – More description added to the MCardInitialize function. Sec 9.2.1 – Description of the McardShutdown API has been updated	Simon Peter
July 3, 2003	1.6	Added description for a new API MCardWaitForCardState Sec 9.1.1 – Description of the MCardInitialize function has been updated. Sec 9.2.1 – Description of the McardShutDown API has been updated	Simon Peter
January 27, 2004	1.7	Added section on SLE4404 card. Modified Verify PIN and error code sections to reflect changes needed for SLE4404 card.	Sudharsanan
April 2,2004	1.8	Updated error codes returned by APIs Sec 11.3 – Description of SLE4432/SLE4442 cards updated. Sec 9.3 – Description of McardConnect API has been updated.	Sudharsanan
June 4, 2004	1.9	Sec 9.4 – Description of McardDisconnect API has been updated.	Vidhya

Contents

(Highlight all the Fields; Go to "Insert" Menu and Choose "Index and Tables" to Update the Contents with respect to the Document)

1.0 Introduction

This document is for application developers who want to use the MCard API to interface memory cards in their application as well as for those who need to implement memory card DLL for their readers.

2.0 Reference Documents

- ISO/IEC 7816-1 1987(E) Standard
- ISO/IEC 7816-2 1988(E) Standard
- ISO/IEC 7816-3 1997(E) Standard
- ISO/IEC 7816-4 1995(E) Standard
- ISO/IEC 7816-5 1994(E) Standard
- ISO/IEC 7816-6 1996(E) Standard
- ISO/IEC 7816-10 1999(E) Standard

3.0 Terms and Abbreviations

Term	Expansion
Memory Cards	Synchronous cards mainly used for storage of Data – Phone Cards, Health cards, etc.
Protocol	The type of transmission followed by the Memory card – 2 Wire, 3 Wire, IIC, etc.
PIN / Security code	The Personal Identification number (Similar to password) given to the user of the Memory card for security reasons.
ISO PIN Contacts	The Contacts of the smart card with the interface device as described in ISO 7816 (C1 to C8)
ATR	Answer to Reset
PC/SC	The standard of interface for Smart card reader drivers
Infineon, ATMEL, Xicor, Schlumberger	Some Memory card suppliers in the industry
SLE4442/32/18/28	Some of the Memory Cards supplied by Infineon
EEPROM	Electrically erasable Programmable Read Only Memory

4.0 Readers Supported

The following devices from SCM Microsystems support the MCard API:

- SCRx31 Smart Card Reader (Firmware Rev 3.0 and above)
- SCR333 Internal Smart Card Reader
- SCR335 USB Smart Card Reader
- SCR338 Keyboard Smart Card Reader
- SCR241 PCMCIA Smart Card Reader
- SPRx32 Pinpad Reader
- SPR336 Biometric Reader
- SPR337 Biometric Reader
- Elektra 331 Smart card Reader

5.0 Introduction to MCard API

PC/SC has become the standard interface to smartcard readers and cards. Unfortunately, the current implementation on Microsoft Windows supports T=0 and T=1 processor cards only. Memory cards with their various protocols are unsupported.

However, in some cases, memory card offers cheaper solutions than processor cards, e.g. for storing data where not much processing is needed. With special read and write PINs, they can also provide protection for this data and with challenge response test, a user authentication is also possible. More advanced cards provide decrement-only counters which makes them perfectly suited as secure debit cards.

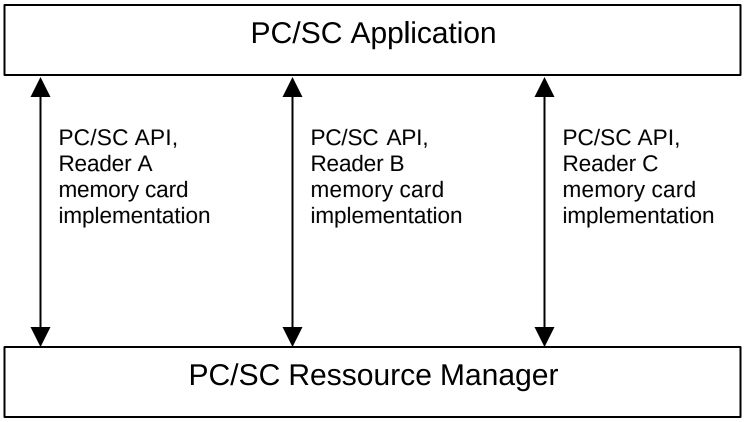
The demand for memory card support has lead to various vendor specific PC/SC driver extensions where such cards are presented e.g. as T=0 cards and access becomes possible via special APDUs. Another common solution is the support via the SCardControl() function with special parameters. Unfortunately, an application has to implement all these different and incompatible methods for each reader it wants to use.

The MCard API has been introduced to provide a reader independent memory card interface which is still compatible with the existing PC/SC API. Thus existing application require only little modification when they also want to support usage memory cards. Once they use the MCard API, they no longer need to care about any device specific interfaces.

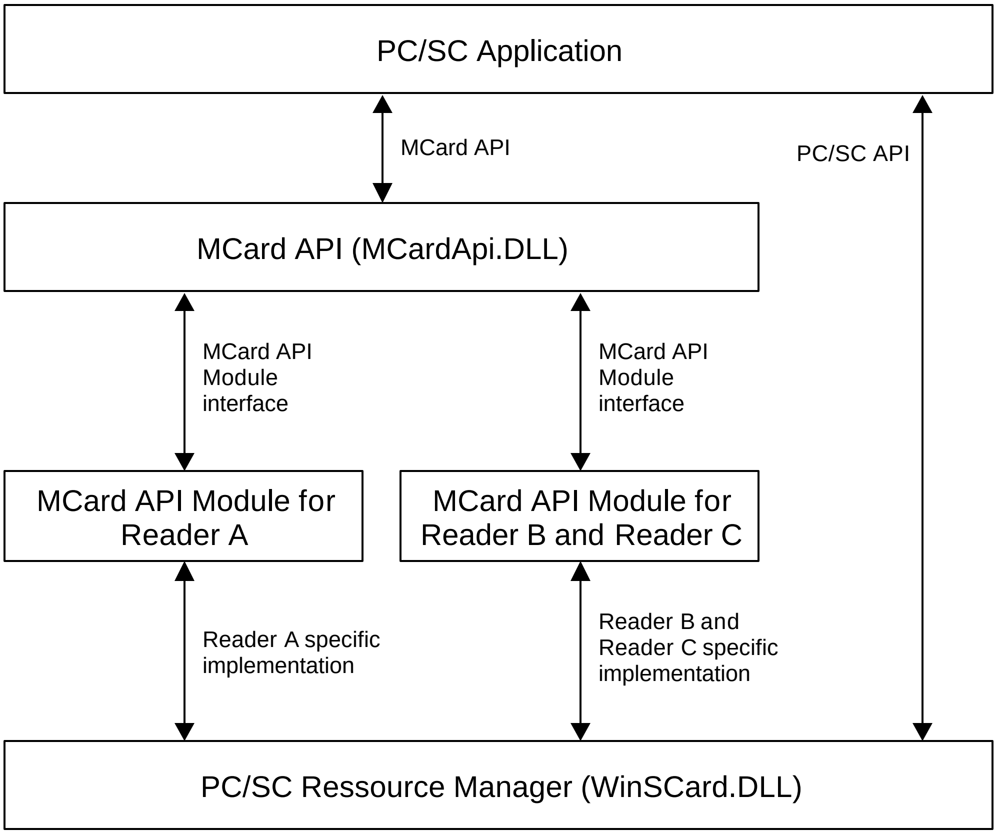
Furthermore, with the MCard API's abstract interface it is not necessary for application developers to fully care about any memory card specific protocol details. This will he handled internally in the API. Of course, a generally understanding of the memory card that is used and its features is still required on the part of the application developer.

Understanding this documentation requires background knowledge about the PC/SC API and smartcards in general. DLL developers must also have detailed knowledge and datasheets of the memory card they want to implement.

6.0 MCard API Design Overview



The MCard API is placed in between the PC/SC application and the Resource Manager. The application can still use all available PC/SC functionality to get the device information or monitor the card state. However, for all memory card related functionality it uses the MCard APIs.



7.0 Features of MCard API

- The most important aspect of the MCard API is its PC/SC compliance. It guarantees complete PC/SC compliance and it has been built around the standard PC/SC layer. This enables the application developer over MCard API to have full PC/SC compliance.

- The application can also use the memory card handle provided by the MCard API for his usual PC/SC calls. This enables the application developer to integrate the application with the MCard API.

- The application developer is free from knowing many of the card specific details. A lot of those are internally handled by the Mcard API itself. However application developers are expected to have a basic understanding of the card that they are interfacing through the MCard API.

- The MCard API error codes are built around the standard PC/SC error codes. The errors reported back by the MCard APIs clearly indicate the type of error that has occurred. In some cases, it gives extra information regarding the error, so that the application developer can take the respective corrective action. Also wherever reasonable, the MCard API returns back the standard error codes as such.

- The MCard API have been modelled around PC/SC calls wherever possible. This will enable the application developers to understand the APIs and their functionality better.

8.0 MCard APIs Data Types

The following data type are used within the MCard AP I functions.

8.1 MCARDCONTEXT

This represents the context of the current MCard API session.

```
typedef   SCARDHANDLE         MCARDCONTEXT;
typedef   MCARDCONTEXT*       PMCARDCONTEXT;
```

8.2 MCARDHANDLE

This handle identifies the current connection to a card. It is also a valid PC/SC API **`SCARDHANDLE`**, so it can be used with some PC/SC functions, too.

```
typedef   SCARDHANDLE         MCARDHANDLE;
typedef   MCARDHANDLE*        PMCARDHANDLE;
```

8.3 MCARD-FEATURES

This structure gives the general features of the memory card.

```
typedef struct _MCARD_FEATURES {
                    DWORD                  dwFlags,
                    BYTE                   bytMemoryZones,
                    BYTE                   byPINs,
                    BYTE                   byCounters,
                    BYTE                   byCRs
                } MCARD_FEATURES, *PMCARD_FEATURES;
```

dwFlags

RFU

byMemoryZones

Number of memory zones. The first memory zone has the ID 0x00, the second the ID 0x01 and so on. This is to used in functions like **`MCardReadMemory.`**

byPINs

Number of PINs. The first PIN has the ID 0x00, the second the ID 0x01 and so on. This is used in functions like **`MCardVerifyPin`**.

byCounters

Number of counters. The first counter has the ID 0x00, the second the ID 0x01 and so on. This is used in functions like **`MCardSetCounter`**.

bCRs

Number of challenge -response tests. The first test in this array has the ID 0x1, the second the ID 0x2 and so on in functions like **`MCardChallengeResponse`** .

8.4 MCARD_MEMORY

This structure contains details about a memory zone of a memory card.

```
typedef struct _MCARD_MEMORY {
                        DWORD          dwFLags,
                        DWORD          dwSize
                       } MCARD_MEMORY, *PMCARD_MEMORY;
```

dwFlags

RFU

dwSize

This is the size in byte of a memory zone.

8.5 MCARD_PIN

This structure contains details about a PIN of a memory card.

```
typedef struct _MCARD_PIN {
                        DWORD   dwFlags,
                        BYTE    bySize,
                        BYTE    byRetries
                       } MCARD_PIN, *PMCARD_PIN;
```

dwFlags

RFU

bySize

This is the size of this PIN in byte.

byRetries

Number of trials left for this PIN.

8.6 MCARD_CR

This structure contains details about a challenge response authentication details supported by a memory card.

```
typedef struct _MCARD_CR {
                         DWORD          dwFlags,
                         DWORD          dwChallengeLen,
                         DWORD          dwResponseLen,
                         BYTE           byRetries
                        } MCARD_CR, *PMCARD_CR;
```

dwFlags

RFU

dwChallengeLen

Length of challenge in bytes.

dwResponseLen

Length of respones in bytes.

byRetries

Number of trials left for this Challenge Response sequence.

8.7 MCARD_COUNTER

This structure contains details about a specific counter on the memory card.

```
typedef struct _MCARD_COUNTER {
                         DWORD          dwFLags
                         BYTE           bySize;
                         DWORD          dwUnits
                        } MCARD_COUNTER, *PMCARD_COUNTER;
```

dwFlags

RFU

bySize

Length of counter in bytes.

dwUnits

Units left in the counter.

9.0 APIs exposed by DLL

9.1 MCardInitialize

9.1.1 Description

This API has to be called by the application before any other memory card API. This API can be called with or without a memory card inserted into the reader. This API is used to set the reader into the memory card mode of operation. Calling the MCardInitialize function for the second time without calling the MCardShutdown function will lead to failure.

```
LONG MCardInitialize (
       IN SCARDCONTEXT hScardContext,
       IN LPCTSTR szReaderName,
       OUT PMCARDCONTEXT phMCardContext,
       OUT PDWORD pdwDllVersion,
    );
```

9.1.2 Description of the Parameters

- The **hScardContext** is the context handle obtained by the SCardEstablishContext by the application with the Smart Card Resource Manager.

- The **szReaderName** is the reader name as given to the SCardConnect.

- The **phMCardContext** is a unique context that is returned by the DLL. This context is to identify the application and is not of any significance to the application developer, other than that it has to be used with the MCardShutdown API. However the application developer has to pass a valid pointer for this parameter to the MCardInitialize call.

- The **pdwDllVersion** will be filled with the current version of the DLL. This is also for reference to the application developer so that he can correlate it with the memory cards supported by the DLL.

9.1.3 Return

MCARD_S_SUCCESS	The DLL successfully initialized
SCARD_E_NO_SERVICE	The resource manager is not running

9.1.4 Sample code

```
/* ScardContext is the context obtained through SCardEstablishContext */
MCARDCONTEXT        hMCardContext;
DWORD               dwDLLVersion;
char                szReader[] = "CCID SCM Microsystems Reader";
LONG                lReturn;

lReturn = MCardInitialize (
ScardContext,
szReader,
&hMCardContext,
&dwDLLVersion
);
```

9.2 MCardShutdown

9.2.1 Description

This API invalidates the DLL context given during a MCard Initialize. It also restores the reader from memory card mode to the mode it was when MCardInitialize was called.

```
LONG MCardShutdown (
     IN MCARDCONTEXT hMCardContext
   );
```

9.2.2 Description of the Parameters

- **hMCardC ontext** is the DLL context supplied to the application during MCardInitialize

9.2.3 Return

MCARD_S_SUCCESS	The context successfully invalidated

9.2.4 Sample code

```
LONG lReturn;

lReturn = MCardShutDown (hMCardContext);

/* where hMCardContext is the context got in MCardInitialize */
```

9.3 MCardConnect

9.3.1 Description

This API is similar to the standard SCardConnect except that it connects to a memory card. The API actually performs a SCardConnect internally. The card handle MCARDHANDLE is returned to the application developer which is the standard SCARDHANDLE. The application developer can use this handle for further communication with the MCard APIs and also to perform card tracking and other such routines through the Smart card resource manager.

Disclaimer

The MCardConne ct in the "intelligent mode" performs **DESTRUCTIVE** tests with the card. This API during its process of finding the card type writes / reads back the data from the card. Though the DLL restores the card's original state once the identification process in ove r, neither the power to the card be interrupted, nor the card disturbed during the process. Since the process is prone to damage the card, application developers are advised to go for intelligent mode only when such a need arises.

```
LONG MCardConnect (
     IN MCARDCONTEXT hMCardContext,
     IN DWORD dwConnectMode,
     IN BYTE  byCardType,
     OUT PMCARDHANDLE phMCard
   );
```

9.3.2 Description of the Parameters

- **hMCardContext** is the DLL context supplied to the application during MCardInitialize
- The **dwConnectMode** chooses between the INTELLIGENT mode (read **disclaimer** above) and FORCED_MODE mode, in which the application has to identify the card type to the DLL.
- If the **dwConnectMode** is set as INTELLIGENT mode then **byCardType** is ignored. But if the dwConnectMode is set as FORCED_MODE, then the card type is passed in this.
- The DLL returns **phMCard** on successful connection to the memory card. This memory card handle will have to be supplied in all further calls to the DLL like MCardReadMemory, MCardWriteMemory etc.

9.3.3 Sample code

The intelligent card identification of the DLL is used.

```
DWORD                   dwConnectMode = INTELLIGENT_MODE
BYTE                    byCardType = 0x00;
MCARDHANDLE             hMCard;
LONG                    lReturn;

lReturn =
MCardConnect (
    hMCardContext,              /* Obtained from MCardInitialize */
    dwConnectMode,
    byCardType,
    &hMCard
);
```

The card type is forced from the application.

```
LONG lReturn;
MCARDHANDLE           hMCard;
DWORD dwConnectMode = FORCED_MODE;
BYTE byCardType = MCARDTYPE_SLE4432;              /* A list of all supported cards with
                                                        values is provided in Annex A  */
lReturn   =
MCardConnect (
      hMCardContext,                              /* Obtained from MCardInitialize */
      dwConnectMode,
      byCardType,
      &hMCard
);
```

9.3.4 Return

MCARD_S_SUCCESS	The memory card is successfully connected to.
MCARD_E_UNKNOWN_CARD	The memory card could not be identified.
MCARD_E_NOT_INITIALIZED	MCardInitialize was not called successfully before this.
MCARD_W_REMOVED_CARD	Card has been removed fr om the reader

Note:
McardConnect will return the error code MCARD_E_NOT_INITIALIZED if the reader is not in Memory card mode.

9.4 McardDisconnect

9.4.1 Description

This API is similar to the SCardDisconnect. It disconnects a previously connected memory card as per the requested disposition.

```
LONG McardDisconnect (
     IN MCARDHANDLE hMCard,
     IN DWORD dwDisposition
   );
```

9.4.2 Description of the Parameters

- **hMCard**is the memory card handle obtained in the MCardConnect call.
- **dwDisposition** The type of disconnection similar to ScardDisconnect's disposition parameter. This parameter is no longer used and may be set to a default value of 0.

9.4.3 Return

MCARD_S_SUCCESS	The memory card is successfully connected to.

9.4.4 Sample code

```
/* hMCard is the handle obtained in the call to MCardInitialize */

LONG lReturn;

lReturn  =
MCardDisconnect(
     hMCard,
     MCARD_EJECT_CARD
);
```

9.5 MCardGetAttrib

9.5.1 Description

This API returns the various attributes of the memory card and also certain configurations of the DLL/reader. The data structures returned by the DLL for each attribute are specified in the Data Structures section.

```
LONG McardGetAttrib (
      IN MCARDHANDLE hMCard,
      IN DWORD dwAttrId,
      OUT LPBYTE pbAttr,
      IN OUT LPDWORD pcbAttrLen
    );
```

9.5.2 Description of the Parameters

- **hMCard** is the card handle of the card that was connected to
- **dwAttrId** indicates the attribute requested. It may be

dwAttrib	Value
MCARD_ATTR_TYPE	0x00
MCARD_ATTR_PROTOCOL	0x01
MCARD_ATTR_FEATURES	0x02
MCARD_ATTR_MEMOR Y	0x03
MCARD_ATTR_PIN	0x04
MCARD_ATTR_CR	0x05
MCARD_ATTR_COUNTERS	0x06
MCARD_ATTR_CLOCK	0x07
MCARD_ATTR_BIT_ORDER	0x08
MCARD_ATTR_CONFIGURATION	0x09

  - The **MCARD_ATTR_TYPE** will return the memory card type.

  - The **MCARD_ATTR_PROTOCOL** will return the protocol of the memory card.

  - The **MCARD_ATTR_FEATURES** will return the structure MCARD_FEATURES i.e
    - DWORD dwFlags;
    - BYTE byMemoryZones;
    - BYTE byPINs;
    - BYTE byCounters;
    - BYTE byCRs;

  - The **MCARD_ATTR_MEMORY** will return the MCARD_MEMORY structure i.e
    - DWORD dwFLags;
    - DWORD dwSize;

  - The **MCARD_ATTR_PIN** will return a the structure MCARD_PIN i.e
    - DWORD dwFlags;
    - BYTE bySize;
    - BYTE byRetries;

- The **MCARD_ATTR_CR** will return the structure MCARD_CR i.e
  - DWORD dwFlags;
  - DWORD dwChallengeLen;
  - DWORD dwResponseLen;
  - BYTE byRetries;

- The **MCARD_ATTR_COUNTERS** will return the structure MCARD_COUNTER i.e
  - DWORD dwFLags;
  - BYTE dwSize;
  - DWORD dwUnits;

- The **MCARD_ATTR_CLOCK** returns the Ton = Toff that is used within the reader to interface with the memory card.

- The **MCARD_ATTR_BIT_ORDER** returns the Bit order that is used within the reader to interface with the memory card.
  - LSB (Value = 0x00) – if the first bit sent/received has been forced by the application developer to be considered as the least significant bit.
  - MSB (Value = 0x01) – if the first bit sent/received has been forced by the application developer to be considered as the most significant bit.
  - DEFAULT (Value = 0xFF) – if the application developer has not forced any bit significance and the memory card reader intelligently identifies the bit significance based on the protocol used.

- The CONFIGURATION attribute has been reserved for future use.

- The contents of **pbAttr** will contain the respective structure(s) or value.
- The contents of **pbAttrLen** will contain the size of the structure returned in pbAttr (if a structure is returned) or the number of bytes (if a value is returned).

9.5.3 Return

MCARD_S_SUCCESS	Successfully got the attribute
MCARD_E_NOT_IMPLEMENTED	The MCardGetAttrib not implemented for this attribute

9.5.4 Sample code

```
/* hMCard is the handle obtained in the call to MCardInitialize */

MCARD_FEATURES MCardFeatures;
DWORD dwLen;
LONG lReturn;

lReturn  =
MCardGetAttrib (
        hMCard,
        MCARD_ATTR_FEATURES,
         (unsigned char *) &MCardFeatures,
        &dwLen
);

/* This will return the memory card features in the structure and in the dwLen it returns the
sizeof (MCARD_FEATURES). */
```

9.6 MCardSetAttrib

9.6.1 Description

This API will set the requested value to the attribute of interest. The clock rate has been set considering all the cards that have been supported by the DLL. If the application developer is comfortable with other values he can use this API to change it.

This API is also used to force the bit ordering to LSB / MSB or set it back to DEFAULT in which case the reader will intelligently decide the bit ordering.

```
LONG McardSetAttrib (
     IN MCARDHANDLE hMCard,
     IN DWORD dwAttrId,
     IN LPBYTE pbAttr,
     IN DWORD cbAttrLen
   );
```

9.6.2 Description of the Parameters

- The **hMCard** is the memory card handle got during a valid MCardConnect
- The **dwAttrib** can take one of these values

dwAttrib	**Value**
MCARD_ATTR_CLOCK	0x07
MCARD_ATTR_BIT_ORDER	0x08

  - The **MCARD_ATTR_CLOCK** attribute is used to set the Ton=Toff used for interfacing the memory card in the reader.

  - The **MCARD_ATTR_BIT_ORDER** attribute can be set to one of these
    - LSB (Value = 0x00) – if the first bit sent/received needs to be forced by the application developer to be considered as the least significant bit.
    - MSB (Value = 0x01) – if the first bit sent/received needs to be forced by the application developer to be considered as the most significant bit.
    - DEFAULT (Value = 0xFF) – if the application developer does not force any bit significance and accepts the memory card reader’s intelligent identification of the bit significance based on the protocol used.

- The **pbAttr** contains the value to be set to the attribute.

- The **cbAttrLen** contains the number of bytes contained in pbAttr

9.6.3 Return

MCARD_S_SUCCESS	Successfully got the attribute
MCARD_E_READ_ONLY_ATTRIBUTE	The MCardGetAttrib not implemented for this attribute

Sample code

```
/* hMCard is the handle obtained in the call to MCardInitialize */

BYTE byClock= 0x01;
LONG lReturn;

lReturn  =
MCardSetAttrib (
            hMCard,
            MCARD_ATTR_CLOCK,
             &byClock,
            1
);

/* This will set the clock to 1 microsecond and is advisable for IIC cards which can work in this clock */
```

9.7 McardReadMemory

9.7.1 Description

This API will read the requested bytes from the memory card. The buffer to store the bytes read is supplied by the application layer. The address roll over, reading beyond the available memory and other similar considerations are automatically taken care by the DLL and the appropriate error codes returned.

```
LONG MCardReadMemory (
     IN MCARDHANDLE hMCard,
     IN BYTE bMemZone,
     IN DWORD dwOffset,
     IN LPBYTE pbReadBuffer,
     IN OUT LPDWORD pbReadLen
   );
```

9.7.2 Description of the Parameters

- **hMCard**is the memory card handle returned on a successful call to MCardConnect
- **bMemZone** indicates the zone from which the data has to be read. For details on how the various memory cards are divided into zones refer Annex A.
- **dwOffset** indicates the offset from which the reading has to take place.
- **pbReadBuffer** is the buffer supplied by the application layer where the bytes read are to be stored.
- **pbReadLen**is the number of bytes to be read.

9.7.3 Return

MCARD_S_SUCCESS	Successfully read all data
MCARD_W_NOT_ALL_DATA_READ	Could not read all data from card
MCARD_W_PIN_VERIFY_NEEDED	Reading access requires a PIN verification.
MCARD_E_INVALID_MEMORY_RANGE	Offset + length greater than size of the zone
MCARD_E_INVALID_MEMORY_ZONE_ID	The specified memory zone ID is invalid.
MCARD_W_CARD_REMOVED	Card has been removed from the reader.

9.7.4 Sample code

```
/* hMCard is the handle obtained in the call to MCardInitialize */

LONG          lReturn;
BYTE          abyData [20];
DWORD         dwLen = 20;

lReturn =
MCardReadMemory (              /* Reading 20 bytes from offset 0x80 in memory zone 0 */f
        hMCard,
        0,
        0x80,
        abyData,
        &dwLen
);
```

9.8 McardWriteMemory

9.8.1 Description

This API is used to write data into the card’s memory. The DLL internally performs a read back for every write to verify whether the bytes have actually been written. Certain cards have permanant write protection mechanism and attempting a write on these bytes will fail.

```
LONG McardWriteMemory (
     IN MCARDHANDLE hMCard,
     IN BYTE bMemZone,
     IN DWORD dwOffset
     IN LPBYTE pbWriteBuffer,
     IN OUT LPDWORD pcbWriteLen
   );
```

9.8.2 Description of the Parameters

- **hMCard**is the memory card handle returned on a successful call to MCardConnect
- **bMemZone** indicates the zone from which the data has to be written. For details on how the various memory cards are divided into zones refer Annex A.
- **dwOffset** indicates the offset from which offset the writing has to begin. This offset can be anywhere, even between pages, as the DLL will internally .
- **pbWriteBuffer** is the buffer supplied by the application layer where the bytes to be written are stored.
- **pcbWriteLen**contains the number of bytes to be write.

9.8.3 Return

MCARD_S_SUCCESS	Successfully writen all data
MCARD_W_PIN_VERIFY_NEEDED	Writing access requires a PIN verification.
MCARD_E_INVALID_MEMORY_RANGE	Offset + length greater than size of the zone
MCARD_E_INVALID_MEMORY_ZONE_ID	The specified memory zone ID is invalid.
MCARD_E_ERASURE_NEEDED	Write not possible erasure to be done first
MCARD_W_PROTECTED_AREA	Cannot write into Protected area
MCARD_W_CARD_REMOVED	Card has been removed from the reader.

9.8.4 Sample code

```
/* hMCard is the handle obtained in the call to MCardInitialize */

LONG            lReturn;
BYTE            abyData [10] = {0x01, 0x02, 0x03, 0x04, 0x05, 0x06, 0x07, 0x08, 0x09, 0x0A};
DWORD           dwLen = 10;

lReturn =
MCardWriteMemory (            /* Writes the 10 values in the buffer abyData to offset 0x80 in
                                      memory zone 0  */
        hMCard,
        0,
        0x80,
        abyData,
        &dwLen
);
```

9.9 McardSetMemoryWriteProtection

9.9.1 Description

This API is used to set the write protection that is available in a few memory cards. This feature may permanently write protect the bytes. The application developer has to make sure of the parameters before calling this API.

```
LONG McardSetMemoryWriteProtection (
      IN MCARDHANDLE hMCard,
      IN BYTE bMemZone,
      IN DWORD dwOffset,
      IN OUT LPDWORD pbcLen
    );
```

9.9.2 Description of the Parameters

- **hMCard**is the memory card handle returned on a successful call to MCardConnect
- **bMemZone** indicates the zone from which the data has to be protected. For details on how the various memory cards are divided into zones refer Annex A.
- **dwOffset** indicates the offset from which offset the protection has to begin.
- **pcbLen** contains the number of bytes to be write protected.

9.9.3 Return

MCARD_S_SUCCESS	Successfully protected all data
MCARD_W_PIN_VERIFY_NEEDED	Protection access requires a PIN verification.
MCARD_E_INVALID_MEMORY_RANGE	Offset + length greater than size of the zone
MCARD_E_INVALID_MEMORY_ZONE_ID	The specified memory zone ID is invalid.
MCARD_W_CARD_REMOVED	Card has been removed from the reader.

9.9.4 Sample code

```
/* hMCard is the handle obtained in the call to MCardInitialize */

LONG            lReturn;
DWORD           dwLen = 5;

lReturn =
MCardSetMemoryWriteProtection (
            hMCard,
            0x00,
            0x10,
            &dwLen
);

/* This sample shows how to write protect 5 bytes, starting from offset 0x10 in memory zone 0 */
```

Disclaimer

The write protection feature can permanantly protect the data from being altered again. So application developers must be sure enough before calling this API.

9.10 McardSetMemoryReadProtection

Description
This API is used to set the read protection available in certain cards.

```
LONG McardSetMemoryReadProtection (
      IN MCARDHANDLE hMCard,
      IN BYTE bMemZone,
      IN DWORD dwOffset,
      IN OUT LPDWORD pbcLen
    );
```

9.10.1 Description of the Parameters

- **hMCard**is the memory card handle returned on a successful call to MCardConnect
- **bMemZone** indicates the zone from which the data has to be protected. For details on how the various memory cards are divided into zones refer Annex A.
- **dwOffset** indicates the offset from which offset the protection has to begin.
- **pcbLen** contains the number of bytes to be write protected.

9.10.2 Return

MCARD_E_NOT_IMPLEMENTED	Since at present the DLL does not support such cards this API has been left unimplemented.

9.11 McardReadMemoryStatus

9.11.1 Description

This API will report the read/write protection status of the bytes.

```
LONG McardReadMemoryStatus (
     MCARDHANDLE hMCard,
     IN BYTE bMemZone,
     IN DWORD dwOffset,
     OUT PBYTE pbBuffer,
     IN OUT LPDWORD pcbLen
   );
```

9.11.2 Description of the Parameters

- **hMCard**is the memory card handle returned on a successful call to MCardConnect
- **bMemZone** indicates the for which the status is to be known. For details on how the various memory cards are divided into zones refer Annex A.
- **dwOffset** indicates the offset from which the status has to be reported
- **pbBuffer** is the buffer supplied by the application layer where the status i to be returned.
- **pcbLen** contains the number of bytes whose status is to be known.

9.11.3 Return

MCARD_S_SUCCESS	Successfully read the status
MCARD_W_CARD_REMOVED	Card has been removed from the reader.

9.11.4 Sample code

```
/* hMCard is the handle obtained in the call to MCardInitialize */
LONG            lReturn;
BYTE            abyData [10];
DWORD           dwLen = 10;

lReturn =
MCardReadMemoryStatus (              /* Gets the status of 10 bytes from offset 0x80 in zone 0 */
        hMCard,
        0,
        0x80,
        abyData,
        &dwLen
);
```

9.11.5 Status byte Interpretation

- Bit 7 - Bit 1 : RFU.
- Bit 0 (LSB) : Write protection bit set for this data byte.

9.12 McardVerifyPIN

9.12.1 Description

This API is used to verify the PIN from the user. This call has to be used to get write/read access to the card/zones.

```
LONG MCardVerifyPIN (
      IN MCARDHANDLE hMCard,
      IN BYTE bPinNumber,
      IN PBYTE pbBufferWithPIN,
      IN BYTE pbcLen
    );
```

9.12.2 Description of the Parameters

- **hMCard** is the memory card handle returned on a successful call to MCardConnect
- **bPinNumber** indicates the PIN which is to be verified
- **pbBufferWithPIN** is the buffer containing the bytes of the PIN
- **pcbLen** contains the number of bytes of the PIN

9.12.3 Return

MCARD_S_SUCCESS	Successfully verified the PIN.
MCARD_E_INVALID_PIN_ID	The specified PIN ID is invalid.
MCARD_W_PIN_VERIFY_FAILED	The PIN verification failed.
MCARD_W_NO_PIN_ATTEMPTS_LEFT	No PIN retries left in the card.
MCARD_W_CARD_REMOVED	Card has been removed from the reader.
MCARD_E_BITORDER_CHANGED	Bit Order has been changed from default.
MCARD_E_NOT_IMPLEMENTED	This Feature is not implemented in the DLL.

9.12.4 Sample code

```
/* hMCard is the handle obtained in the call to MCardInitialize */
LONG            lReturn;
BYTE            PINbuffer [3] = {0x73, 0x58, 0xDE};


lReturn =
MCardVerifyPIN (                /* Verifies the PIN for SLE4442 */
        hMCard,
        0,
        PINbuffer,
        3
);
```

Important Note:
The verify PIN option in the case of SLE4404 cards on Memory code area (PIN1) will erase the entire contents of user memory and also reset a bit in Memory counter area that results in the decrement of the no. of times the user area can be erased. Even an incorrect PIN1 entry will reset the bit in memory counter, decreasing the no. of times the user area can be erased.

Please ensure that before verifying PIN1 of SLE4404 card PIN0 is verified.

9.13 McardChangePIN

9.13.1 Description

The application will use this API to change the current PIN to a new PIN.

```
LONG MCardChangePIN (
     IN MCARDHANDLE hMCard,
     IN BYTE bPinNumber,
     IN PBYTE pbBufferWithOldPIN,
     IN BYTE cbOldLen
     IN PBYTE pbBufferWithNewPIN,
     IN BYTE cbNewLen
   );
```

9.13.2 Description of the Parameters

- **hMCard**is the memory card handle returned on a successful call to MCardConnect
- **bPinNumber** indicates the PIN which is to be changed
- **pbBufferWithOldPIN** is the buffer containing the bytes of the old PIN
- **cbOldLen** contains the number of bytes of the new PIN
- **pbBufferWithNewPIN** is the buffer containing the bytes of the old PIN
- **cbNewLen**contains the number of bytes of the new PIN

9.13.3 Return

MCARD_S_SUCCESS	Successfully changed the PIN.
MCARD_E_INVALID_PIN_ID	The specified PIN ID is invalid.
MCARD_W_PIN_VERIFY_FAILED	The PIN verification failed.
MCARD_W_NO_PIN_ATTEMPTS_LEFT	No PIN retries left in the card.
MCARD_W_CARD_REMOVED	Card has been removed from the reader.
MCARD_E_NOT_IMPLEMENTED	The feature is not implemented in the DLL.

9.13.4 Sample code

```
/* hMCard is the handle obtained in the call to MCardInitialize */
LONG            lReturn;
BYTE            oldPINbuffer [3] = {0x73, 0x58, 0xDE};
BYTE            newPINbuffer [3]={0x3D, 0xF3, 0x25};

lReturn =
MCardChangePIN (                /* Changes the PIN for SLE4442 */
        hMCard,
        0,
        oldPINbuffer,
        3,
        newPINbuffer,
        3
);
```

9.14 McardChallengeResponse

9.14.1 Description

This API implements the challenge response sequence supported by some secure memory cards. Provision is provided to select the challenge ID for a challenge-response sequence.

```
LONG McardChallengeResponse (
     IN MCARDHANDLE hMCard,
     IN BYTE bChallengeID,
     IN PBYTE pbChallengeBuffer,
     IN BYTE cbChallengeLen,
     OUT PBYTE pbResponseBuffer,
     OUT PBYTE cbResponseLen
   );
```

9.14.2 Description of the Parameters

- **hMCard** is the memory card handle returned on a successful call to MCardConnect
- **bChallengeID** indicates the challenge ID for the current authentication.
- **pbChallengeBuffer** is the buffer containing the challenge bytes
- **cbChallengeLen** contains the number of bytes in the challenge buffer
- **pbResponseBuffer** is the buffer where the response from the card will be stored
- **cbResponseLen** contains the number of bytes in the returned response buffer

9.14.3 Return

MCARD_S_SUCCESS	Successfully writen all data
MCARD_E_INVALID_CHAL_RESP_ID	The specified challenge ID is invalid.
MCARD_E_CHAL_RESP_FAILED	The challenge response sequence failed.
MCARD_W_NO_CR_ATTEMPTS_LEFT	No challenge response retry left.
MCARD_W_CARD_REMOVED	Card has been removed from the reader.

9.14.4 Sample code

```
/* hMCard is the handle obtained in the call to MCardInitialize */
LONG            lReturn;
BYTE            ChallengeBuffer [8] = {0x00,0x01,0x02,0x03,0x04,0x05,0x06,0x07}; /* Usually a
                                                                      random number */
BYTE            ResponseBuffer [3];

lReturn =
MCardChallengeResponse (                /* Authentication in SLE4436 */
        hMCard,
        0,                              /* Authentication with Key 1 */
        ChallengeBuffer,
        8,
        ResponseBuffer,
        3
);
```

9.15 McardDeductCounter

9.15.1 Description

This function is used to decrement a counter on the card. (**SLE4406/SLE4436/SLE5536 only**).

```
LONG MCardDeductCounter(
   IN MCARDHANDLE hMCard,
   IN BYTE bCounterID,
   IN DWORD dwUnits
   );
```

9.15.2 Description of the Parameters

- **hMCard**is the memory card handle returned on a successful call to MCardConnect
- **bCounterID** indicates the counter ID where the decrement will be performed
- **dwUnits** is the number of units to be decremented.

9.15.3 Return

MCARD_S_SUCCESS	Successfully writen all data
MCARD_W_CARD_REMOVED	Card has been removed from the reader.

9.15.4 Sample code

```
/* hMCard is the handle obtained in the call to MCardInitialize */
LONG            lReturn;

lReturn =
MCardDeductCounter(          /* To decrement 5 units from the counter in SLE4436 */
         hMCard,
         0,
         5
);
```

Disclaimer

The decrement option in SLE4406, SLE4436 and SLE5536 cause permenant decrement in the value of the counter. Values once decremented are irreversibly lost. So the application developer has to use this API only after through understanding of the purpose.

9.16 McardSetCounter

9.16.1 Description

This function is used to set the counter on the card to some defined bit pattern.

```
LONG MCardSetCounter(
  IN MCARDHANDLE hMCard,
  IN BYTE bCounterID,
  IN PBYTE pbCounter,
  IN BYTE cbCounterLen
  );
```

Explanation of the Parameters

Argument	Type	Meaning
hMCard	MCARDHANDLE	The card handle of the card that was connected to
bCounterID	BYTE	The counter to be decremented
pbCounter	PBYTE	Buffer containing the target bit pattern for the counter
cbCounterLen	BYTE	Number of bytes forming the counter

9.16.2 Description of the Parameters

- **hMCard** is the memory card handle returned on a successful call to MCardConnect
- **bCounterID** indicates the counter ID where the values will be set
- **pbCounter** is the buffer containing the bit pattern to be stored in the counter.
- **pbCounterLen** is the length in bytes of the counter.

9.16.3 Return

MCARD_E_NOT_IMPLEMENTED	At present the DLL does not support this API

9.17 MCardWaitForCardState

9.17.1 Description

This function is used to monitor the card state from the application.

```
LONG MCardWaitForCardState(
   IN     MCARDCONTEXT hMCardContext,
   IN     DWORD        dwExpectedCardState,
   OUT DWORD*          pdwCardState,
   IN    DWORD         dwTimeOut
   );
```

Explanation of the Parameters

Argument	Type	Meaning
hMCardContext	MCARDCONTEXT	The context given during MCardInitialize
dwExpectedCardState	DWORD	The card state that is expected
pdwCardState	PDWORD	The actual card state
dwTimeOut	DWORD	The maximum waiting time for the expected state.

9.17.2 Description of the Parameters

- **hMCardContext** is the memory card context returned on a successful call to MCardInitialize
- **dwExpectedCardState** indicates the expected card state.
  - SCARD_STATE_PRESENT
  - SCARD_STATE_EMPTY
- **pdwCardState** is the buffer containing the bit pattern to be stored in the counter.
  - SCARD_STATE_PRESENT
  - SCARD_STATE_EMPTY

    ```
    NOTE: These a bit flags, so the app must check if the bits are
    set.
    WRONG:  if (SCARD_STATE_PRESENT == *pdwState) { ... }
    RIGHT:  if ((*pdwState) & SCARD_STATE_PRESENT) { ... }
    ```

- **dwTimeOut** is the maximum waiting time for the expected state that the DLL should wait before returning the current state.
  - 0 – return immediately
  - INFINITE – return only when the expected state is reached
  - Any other value – timeout in milliseconds.

9.17.3 Return

MCARD_E_INVALID_PARAMETER	One of the parameters is invalid
MCARD_E_INTERNAL_ERROR	An internal error has occurred in the DLL
MCARD_W_ABORTED	The operation was aborted by a call to MCardShutdown
MCARD_W_TIMEOUT	Timeout occurred while waiting for expected state

10.0 Annex A

10.1 MCard API Error Codes

The MCard API can return the standards Windows error codes ERROR_xxx and the smart card error codes SCARD_xxx. Both are described in the Windows SDK files WINERROR.H and SCARDERR.H. In addition, there are MCARD_xxx error codes to identify the source of an error more precisely. The base value for MCARD_xxx error codes is 0x90100800, which is in fact the SCARD_xxx base value with the COSTOMER_CODE_FLAG set and an additional offset of 0x800 added.

There are two groups of MCARD_xxx error codes. The MCARD_E_xxx codes representing serious errors and the MCARD_W_xxx codes which are warnings. It's up to the application to decide whether those warning are really error.

Error Code	Meaning
MCARD_S_SUCCESS	Successful operation
MCARD_E_INTERNAL_ERROR	An internal error has occurred
MCARD_E_NOT_IMPLEMENTED	API / functionality not implemented
MCARD_E_NOT_INITIALIZED	MCardInitialize not successfully called
MCARD_E_INCOMPATIBLE_READER	The reader is incompatible with the DLL
MCARD_E_UNKNOWN_CARD	Card could not be identified
MCARD_E_BUFFER_TOO_SMALL	The buffer for return data is too small
MCARD_E_INVALID_PARAMETER	One or moreparameters are invalid
MCARD_E_READ_ONLY_ATTRIBUTE	This attribute can only be read.
MCARD_E_INVALID_HANDLE	The handle is invalid
MCARD_E_PROTOCOL_MISMATCH	Protocol error while connecting to card
MCARD_E_PROTOCOL_ERROR MCARD_E_CHAL_RESP_FAILED	Protocol error during card access Challenge response failed
MCARD_E_INVALID_MEMORY_RANGE	Invalid memory range
MCARD_E_INVALID_MEMORY_ZONE_ID	Specified memory zone ID is invalid for current card
MCARD_E_INVALID_PIN_ID	Specified PIN ID is invalid for current card
MCARD_E_INVALID_CHAL_RESP_ID	Specified challenge/response ID is invalid for current card
MCARD_E_ERASURE_NEEDED	Erasure to be done before this write
MCARD_E_BITORDER_CHANGED	Bit Order Changed from default
MCARD_W_NOT_ALL_DATA_READ	Could not read all data from card
MCARD_W_NOT_ALL_DATA_WRITTEN	Could not write all data to card
MCARD_W_PIN_VERIFY_NEEDED	PIN must be verified before access is possible
MCARD_W_PIN_VERIFY_FAILED	PIN verification failed
MCARD_W_NO_PIN_ATTEMPTS_LEFT	No PIN verification attempts left, card probably locked

MCARD_W_NO_UNITS_TO_DECREMENT	No units left in the card to decrement
MCARD_W_REMOVED_CARD	The card has been removed
MCARD_W_TIMEOUT	Timeout occurred
MCARD_W_ABORTED	Command Aborted
MCARD_W_PROTECTED_AREA	Can not write into protected area

10.2 Memory cards supported

Card Type	**Value**	**Cards covered**
MCARDTYPE_UNKNOWN	0x00	None
MCARDTYPE_SLE4406	0x01	SLE4406, SLE4406E, SLE4406S, SLE4406SE from **Infineon**
MCARDTYPE_SLE4418	0x02	SLE4418 from **Infineon**
MCARDTYPE_SLE4428	0x03	SLE4428 from **Infineon** Primflex Store8K from **Schlumberger**
MCARDTYPE_SLE4432	0x04	SLE4432 from **Infineon**
MCARDTYPE_SLE4436	0x05	SLE4436, SLE4436E from **Infineon**
MCARDTYPE_SLE4442	0x06	SLE4442 from **Infineon** Primeflex Store2K from **Sclumberger**
MCARDTYPE_SLE5536	0x07	SLE5536, SLE5536E from **Infineon**
MCARDTYPE_AT24C01ASC	0x08	AT24SC01ASC from **ATMEL**
MCARDTYPE_AT24C02SC	0x09	AT24C02SC from **ATMEL**
MCARDTYPE_AT24C04SC	0x0A	AT24C04SC from **ATMEL**
MCARDTYPE_AT24C08SC	0x0B	AT24C08SC from **ATMEL**
MCARDTYPE_AT24C16SC	0x0C	AT24C16SC from **ATMEL**
MCARDTYPE_AT24C32SC	0x0D	AT24C32SC from **ATMEL**
MCARDTYPE_AT24C64SC	0x0E	AT24C64SC from **ATMEL**
MCARDTYPE_AT24C128SC	0x0F	AT24C128SC from **ATMEL**
MCARDTYPE_AT24C256SC	0x10	AT24C256SC from **ATMEL**
MCARDTYPE_AT24C512SC	0x11	AT24C512SC from **ATMEL**
MCARDTYPE_AT88SC153	0x12	AT88SC153 from **ATMEL**
MCARDTYPE_AT88SC1608	0x13	AT88SC1608 from **ATMEL**
MCARDTYPE_SLE4404	0x14	SLE4404 from Infineon

10.3 Zone IDs

In the MCard API set, reference to Memory card zone is present in many APIs.

Most of the cards have just one zone with Zone ID '0' – SLE4406, SLE4418, SLE4428, SLE4432, SLE36, SLE4442, SLE5536, AT24x series.

The AT88SC153 card has 4 zones

- Zone ID 0 – Configuration zone
- Zone ID 1 – 3 – 3 User Zones

The AT88SC1608 card has 9 zones

- Zone ID 0 – Configuration zone
- Zone ID 1 – 9 – 8 User Zones

10.4 PIN IDs

All the **AT24x** series of cards the **SLE4418** and **SLE4432** do not have any PIN security.

The **SLE4406**, **SLE4436** and the **SLE5536** have transport PINs with PIN ID '0'.

The **AT88SC153** has two set of READ/WRITE PINs (a total of 4 PINs).

- PIN ID 0 – WRITE PIN of Set 0
- PIN ID 1 – WRITE PIN of Set 1
- PIN ID 2 – READ PIN of Set 0
- PIN ID 3 – READ PIN of Set 1

The **AT88SC1608** has eight set of READ/WRITE PINs (a total of 16 PINs).

- PIN ID 0 – WRITE PIN of Set 0
- PIN ID 1 – WRITE PIN of Set 1
- PIN ID 2 – WRITE PIN of Set 2
- PIN ID 3 – WRITE PIN of Set 3
- PIN ID 4 – WRITE PIN of Set 4
- PIN ID 5 – WRITE PIN of Set 5
- PIN ID 6 – WRITE PIN of Set 6
- PIN ID 7 – WRITE PIN of Set 7
- PIN ID 8 – READ PIN of Set 0
- PIN ID 9 – READ PIN of Set 1
- PIN ID 10 – READ PIN of Set 2
- PIN ID 11 – READ PIN of Set 3
- PIN ID 12 – READ PIN of Set 4
- PIN ID 13 – READ PIN of Set 5
- PIN ID 14 – READ PIN of Set 6
- PIN ID 15 – READ PIN of Set 7

The **SLE4404** card has two set of PINS

PIN ID 0 -READ/WRITE protection for protected area (scratchpad/user memory)
PIN ID 1 -that denotes the number of times the user memory can be erased.

11.0 Annex B

This annex describes in detail the characteristics of a few memory cards. The document also highlights the differences between the memory cards, on various attributes. The purpose of the document is to give an insight into the various features available in the different memory cards.

11.1 Memory card standards

The ISO 7816-10 describes two types of memory cards based on the contacts used. But most of the memory cards in the market have vendor defined standards. So memory cards vary in memory sizes, security features, zones, speed and complexity.

11.2 Memory card protocols

Since the memory cards do not follow a single standard the protocols used to interface them also differs from card to card. However they can be classified broadly as

- 2 – Wire Protocol
- 3 – Wire Protocol
- IIC Protocol
- Bit Protocol

11.2.1 Two-Wire protocol

In the 2-Wire protocol, apart from VCC, which is used to power the card, two other lines (CLOCK and I/O) are used for the interfacing. The RST line in the 2 – Wire cards are usually meant for aborting the command at any point of time. Otherwise the RST is maintained in the LOW state during normal processing.

There are definite steps called Start/Stop conditions which need to be followed when sending commands to a 2 – Wire protocol card.

11.2.2 Three-Wire protocol

The 3 – Wire protocol cards use a third line (RST) in addition to the two lines used by the 2 – Wire cards. The RST is maintained in the logical HIGH state during a command transfer to the card. During any other time it is maintained in the logical LOW state.

Similar to the 2 – Wire cards, the 3 – Wire cards also have a specific Start/Stop condition.

11.2.3 IIC protocol

This protocol is also similar to the 2– Wire protocol considering the lines involved in interfacing the card. However these cards require an “ACK” sequence. The card expects an ‘ACK’ from the reader after it has transferred a byte. Similarly if the card successfully receives a byte from the reader it will acknowledge it with the ‘ACK’. The ‘ACK’ is usually a bit ‘0’ transfer.

11.2.4 Bit level protocol

The protocol is known as bit level protocol as every pulse to the card will output the bit present in the internal address at that time. Also the address will increment to the next location, so that on the next clock pulse the bit at that address is output. The counter cards (used as phone cards) are implemented with this protocol.

11.3 Special features in various memory cards

11.3.1 SLE 4432

Attribute	Value
Protocol	2 – Wire
Manufacturer	**Infineon**
Total Memory size	256 bytes of byte organized memory
Zones	1
PINs	0
Other Variants	Not known

Permanent Write Protection

The card supports permanent write protection mechanism for its first 32 bytes. The write protection status can be got at any time by reading the 32 protection bits which correspond each to the first 32 bytes in the card's memory. Once set the protect bits cannot be reset.
The MCardSetMemoryWriteProtection API in the DLL supports this feature.

11.3.2 SLE 4442

Attribute	Value
Protocol	2 – Wire
Manufacturer	**Infineon**
Total Memory size	256 bytes of byte organized memory
Zones	1
PINs	1
Other Variants	Schlumberger PrimeFlex Store 2K / ISSI 2k

The card is the secure version of the SLE 4432.

Permanent Write Protection

The card supports permanent write protection mechanism for its first 32 bytes similar to the SLE4432. This feature is supported by the **MCardSetMemoryWriteProtection** API in the DLL

PIN security

The PIN is a 3 byte security code given to the card user. This PIN has to be verified to perform any type of write activity. There are 3 retries for the PIN verification after which the card will be permanently write protected and of no use.

Once the PIN is verified the number of retries is reset back to three. This feature is supported by the **MCardVerifyPIN** API in the DLL

The PIN once verified can be changed to any three byte number. This feature is supported by the **MCardChangePIN** API in the DLL. PIN verification is valid for the current card session.

11.3.3 SLE 4418

Attribute	Value
Protocol	3 – Wire
Manufacturer	**Infineon**
Total Memory size	1024 bytes of byte organized memory
Zones	1
PINs	0
Other Variants	Not known

Permanent Write Protection

The card supports permanent write protection mechanism for all its 1024 bytes. This is accomplished by having 1024 protect bits each of which can be individually set and which correspond directly to the 1024 bytes of the card. This feature is supportedby the **MCardSetMemoryWriteProtection** API in the DLL . The protection feature is irreversible similar to SLE4432/42.

11.3.4 SLE 4428

Attribute	Value
Protocol	3 – Wire
Manufacturer	**Infineon**
Total Memory size	1024 bytes of byte organized memory
Zones	1
PINs	1
Other Variants	Not known

Permanent Write Protection

The card supports permanent write protection mechanism for all its 1024 bytes similar to SLE4418. This feature is supported by the **MCardSetMemoryWriteProtection** API in the DLL.

PIN security

The PIN is a 2 byte security code given to the card user. This PIN has to be verified to perform any type of write activity. There are 8 retries for the PIN verification after which the card will be permanently write protected and of no use.

Once the PIN is verified the number of retries is reset back to eight. This feature is supported by the **MCardVerifyPIN** API in the DLL

The PIN once verified can be changed to any two byte number. This feature is supported by the **MCardChangePIN** API in the DLL . PIN verification is valid for the current card session.

11.3.5 AT24C01A / 02 / 04 / 08 / 16 / 32 / 64 / 128 / 256 / 512

Attribute	Value
Protocol	IIC
Manufacturer	ATMEL
Total Memory size	1 / 2 / 4 / 8 / 16 / 32 / 64 / 128 / 256 / 512 K bits depending on the card (Byte organised)
Zones	1
PINs	0
Other Variants	Some Xicor cards

All these cards are from ATMEL and they differ only in the memory capacity and thereby in the number of address bytes required in the command. These cards are plain cards and have no security whatsoever. There are no Protect bits or Security codes. They all follow IIC protocol ('ACK')

Random and Sequential Read

The SLE cards all support only random address READ, which require the address of the byte to be read in each READ command.

But all these ATMEL cards support sequential read. In these cards if 100 bytes are to be read which are in consecutive addresses, then a random read is done for the first byte and for the remaining bytes a single sequential read can be done (the address need not be mentioned). This is because the ATMEL cards internally remember the address and increment it for each read command automatically. Thus faster reading is possible. This feature is supported by the normal **MCardReadMemory** API itself. The sequential read is internally handled in the DLL.

Also, during a read operation if a read is attempted beyond the available memory the address roles over from the last byte of the card's entire memory to the first byte (0th address) and reading proceeds again from there.

Page Write

Similar to the sequential read, these cards also support a page write (address need to be mentioned only once). However the page write is limited by the page size of the card.

The page sizes range from 8 bytes to 128 bytes for these cards. Once the end of the a page is reached, the next write will wrap around to the first byte of the same page overwriting the data there. This is internally handled by the **MCardWriteMemory** API itself.

11.3.6 AT88SC153

Attribute	Value
Protocol	IIC
Manufacturer	ATMEL
Total Memory size	64 (config) + 3*64 (user) bytes
Zones	4
PINs	4 (2 READ and 2 WRITE as 2 sets)
Other Variants	Not Known

Random and Sequential Read

As an IIC card, this card also has the same Sequential read facility. The normal **MCardReadMemory** API can be used for this.

Page Write

Similarly there is the page write concept. The page size of these cards is 8. The normal **MCardWriteMemory** API can be used for this.

Multi Zones

The card has three user zones. The zones can have different access rights. They also can have different passwords (from the available set of two). Such cards have greater scope to be used as multi application cards. The APIs **MCardReadMemory** and **MCardWriteMemory** internally handle the set user zone as per the zone ID given by the application developer.

Two sets of PINs

The card supports two different sets of PINs. Each set has one READ PIN and one WRITE PIN. Thus in effect there are four PINs. The three user zones can refer to any of these two PIN sets. Since we have only 2 PIN sets but three user zones, one of the two PIN sets will have to be shared between the user zones. The **MCardVerifyPIN** and **MCardChangePIN** can be used to verify and modify the PIN sets, the PIN ID distinguishing the PINs in these calls.

Configuration Zone

The card has a special zone called the configuration zone. This zone contains data which configure certain attributes of the card.

- **Fuses**

  The various levels of personalization of the card is controlled by the Fuses present in the configuration zone. The fuses can be read any time. Writing of the fuses in allowed only on verification of the Write PIN of set 1. (also known as the SECURITY CODE). Writing the fuse is an irreversible process and has to be done with care.

- **Access Registers**

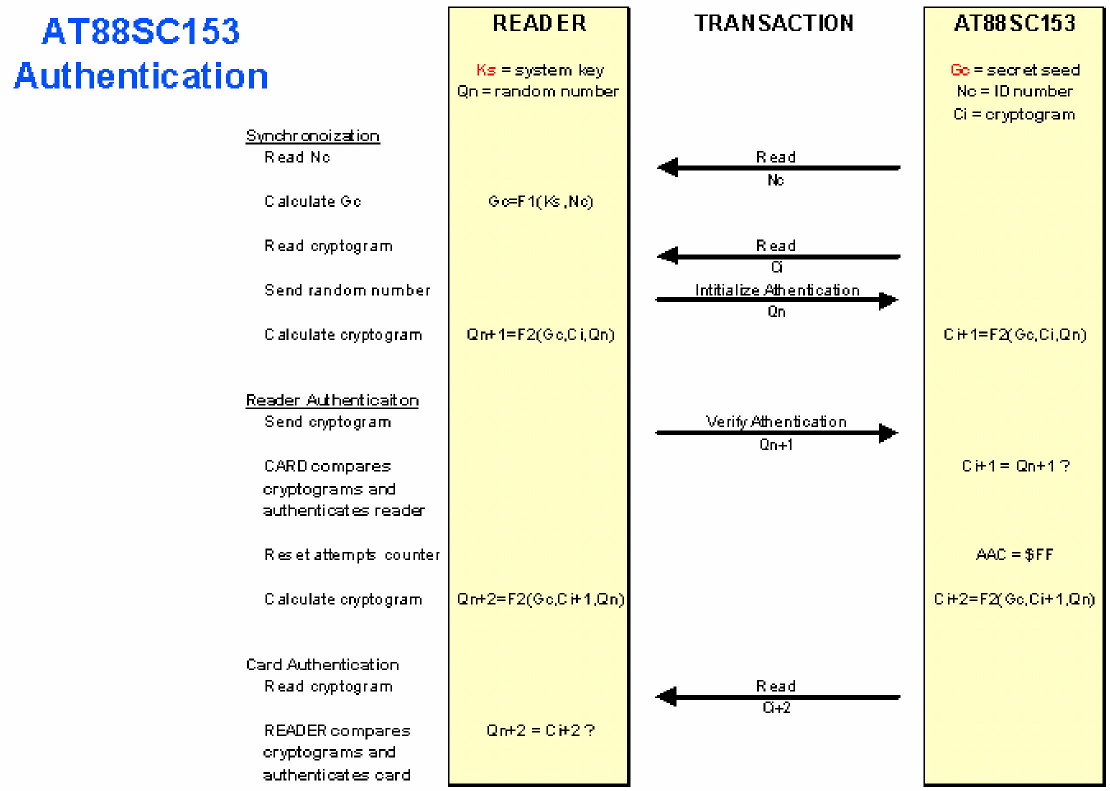
The access registers define the various access rights to the user zones and when such access is to be given. There is one byte of access register referring to a user zone. The eight bits in the access register decides

- The PIN set for the zone
- Should the READ PIN be verified before a read is attempted
- Should the WRITE PIN be verified before a write is attempted
- Can the zone be written to (modified)
- Is the zone erase proof (0 to 1 is allowed. 1 to 0 is not)
- Whether "Challenge – Response" Authentication is required before a READ/WRITE

The configuration can be read with **MCardReadMemory** API and written to using the **MCardWriteMemory** API with the appropriate zone ID. But writing to the configuration zone has to be done with care as there are sensitive data within the the zone.

Two way Challenge – Response Authentication

These cards support a two way Challenge/Response sequence. This will enable both the reader and the card to authenticate each other. The card has to respond to the random number generated by the reader and it will also authenticate the reader by indicating so in its authentication retry counter.
This can be carried out with the **MCardChallengeResponse** API.



11.3.7 AT88SC1608

Attribute	Value
Protocol	IIC
Manufacturer	ATMEL
Total Memory size	128 (config) + 8*256 (user) bytes
Zones	9
PINs	16 (8 READ and 8 WRITE as 8 sets)
Other Variants	Not Known

Random and Sequential Read

As an IIC card, this card also has the same Sequential read facility. The normal **MCardReadMemory** API can be used for this.

Page Write

Similarly there is the page write concept. The page size of these cards is 16. The normal **MCardWriteMemory** API can be used for this.

Multi Zones

The card has eight user zones. The zones can have different access rights. They also can have different passwords (from the available set of eight). Such cards have greater scope to be used as multi application cards. The APIs **MCardReadMemory** and **MCardWriteMemory** internally handle the set user zone as per the zone ID given by the application developer.

Eight sets of PINs

The card supports eight different sets of PINs. Each set has one READ PIN and one WRITE PIN. Thus in effect there are sixteen PINs. The eight user zones can refer to any of these eight PIN sets. The **MCardVerifyPIN** and **MCardChangePIN** can be used to verify and modify the PIN sets, the PIN ID distinguishing the PINs in these calls

Configuration Zone

The card has a special zone called the configuration zone. This zone contains data which configure certain attributes of the card.

- **Fuses**

  The various levels of personalization of the card is controlled by the Fuses present in the configuration zone. The fuses can be read any time. Writing of the fuses in allowed only on verification of the Write PIN of set 1. (also known as the SECURITY CODE). Writing the fuse is an irreversible process and has to be done with care.

- **Access Registers**

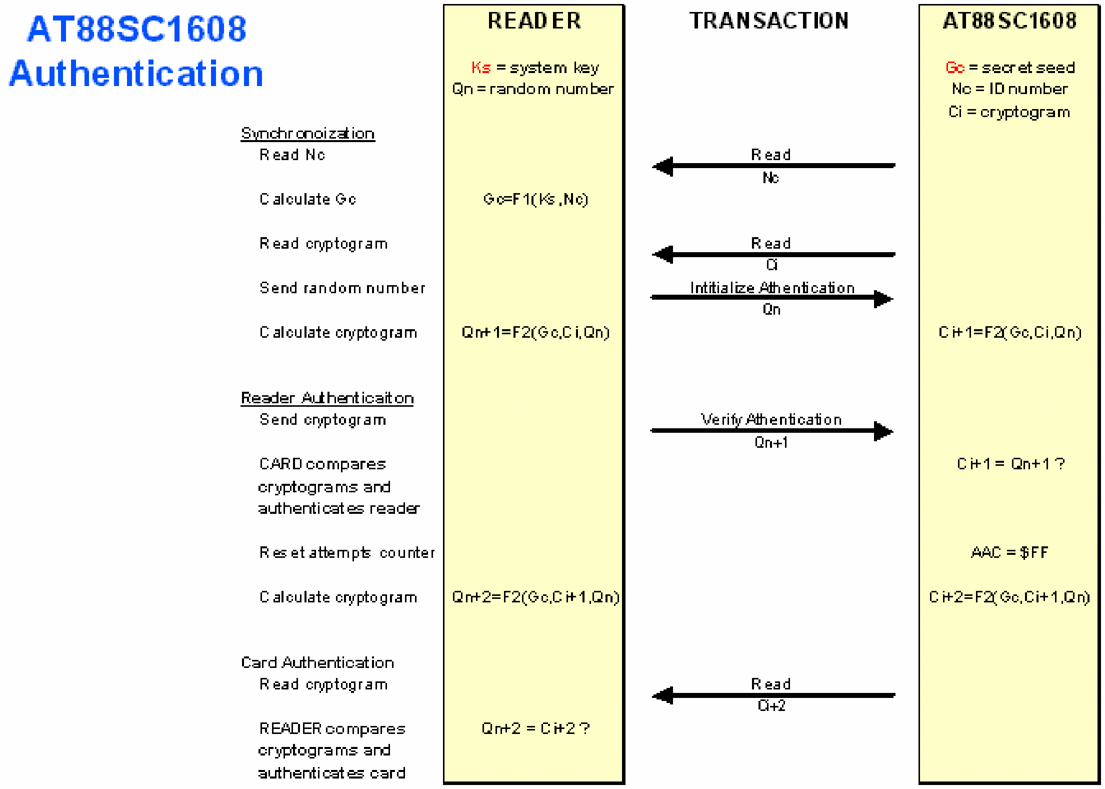
The access registers define the various access rights to the user zones and when such access is to be given. There is one byte of access register referring to a user zone. The eight bits in the access register decides

- The PIN set for the zone
- Should the READ PIN be verified before a read is attempted
- Should the WRITE PIN be verified before a write is attempted
- Should the authentication be done before the READ/WRITE

The configuration canbe read with **MCardReadMemory** API and written to using the **MCardWriteMemory** API with the appropriate zone ID. But writing to the configuration zone has to be done with care as there are sensitive data within the the zone.

Two way Challenge – Response Authentication

These cards support a two way Challenge/Response sequence. This will enable both the reader and the card to authenticate each other. The card has to respond to the random number generated by the reader and it will also authenticate the reader by indicating so in its authentication retry counter.
This can be carried out with the **MCardChallengeResponse** API.



11.3.8 SLE4406

Attribute	Value
Protocol	Bit protocol
Manufacturer	Infineon
Total Memory size	16 bytes including 5(8-bit) stage counter
Zones	1
PINs	1 (Transport code)
Other Variants	Not Known

Phone cards

These cards are usually used as phone cards. They have a 5 stage counter and they are an abacus type of counter.The counters cannot be recharged.

The cards are of use and throw type. There is a hardware fuse mechanism that is permanently blown during decrement counter. The counter can be decremented with the **MCardDecrementCounter** API.

Secure Challenge – Response Authentication

These cards support Challenge/Response sequence and hence are considered very secure. So freaking such cards is very difficult. The host reader will challenge the card each time with a random number and based on the algorithm implemented the card will return the response which can be authenticated by the reader. This can be carried out with the **MCardChallengeResponse** API.

Transport Code Protection

This card is protected during the transport from the chip manufacturer to the card manufacturer by a special code called transport code. The transport code can be verified through the **MCardVerifyPIN** API.

11.3.9 SLE4436

Attribute	Value
Protocol	Bit protocol
Manufacturer	Infineon
Total Memory size	46 bytes including 5(8-bit) stage counter
Zones	1
PINs	1 (Transport code)
Other Variants	Not Known

Phone cards

These cards are usually used as phone cards. They have a 5 stage counter and they are an abacus type of counter.The counters cannot be recharged.

The cards are of use and throw type. There is a hardware fuse mechanism that is permanently blown during decrement counter. The counter can be decremented with the **MCardDecrementCounter** API.

Secure Challenge – Response Authentication

These cards support Challenge/Response sequence and hence are considered very secure. So freaking such cards is very difficult. The host reader will challenge the card each time with a random number and based on the algorithm implemented the card will return the response which can be authenticated by the reader. This can be carried out with the **MCardChallengeResponse** API.

Transport Code Protection

This card is protected during the transport from the chip manufacturer to the card manufacturer by a special code called transport code. The transport code can be verified through the **MCardVerifyPIN** API.

Counter Backup

These cards support the counter backup mechanism. This is useful when there is a failure in the middle of a decrement operation, in which the decrement has been done but a reload of the lesser significant stage has not yet been done. The backup bits indicate this and can be made use of to issue the reload later on. The user can read the backup bits through the **MCardReadMemory** API and do the necessary corrective action.

11.3.10 SLE5536

Attribute	Value
Protocol	Bit protocol
Manufacturer	Infineon
Total Memory size	46 bytes including 5(8-bit) stage counter
Zones	1
PINs	1 (Transport code)
Other Variants	Not Known

Phone cards

These cards are usually used as phone cards. They have a 5 stage counter and they are an abacus type of counter.The counters cannot be recharged.

The cards are of use and throw type. There is a hardware fuse mechanism that is permanently blown during decrement counter. The counter can be decremented with the **MCardDecrementCounter** API.

Secure Challenge – Response Authentication

These cards support Challenge/Response sequence and hence are considered very secure. So freaking such cards is very difficult. The host reader will challenge the card each time with a random number and based on the algorithm implemented the card will return the response which can be authenticated by the reader. This can be carried out with the **MCardChallengeResponse** API.

Transport Code Protection

This card is protected during the transport from the chip manufacturer to the card manufacturer by a special code called transport code. The transport code can be verified through the **MCardVerifyPIN** API.

Counter Backup

These cards support the counter backup mechanism. This is useful when there is a failure in the middle of a decrement operation, in which the decrement has been done but a reload of the lesser significant stage has not yet been done. The backup bits indicate this and can be made use of to issue the reload later on. The user can read the backup bits through the **MCardReadMemory** API and do the necessary corrective action.

Extended Authentication

Apart from the normal authentication, these cards support Extended authentication known as the Cipher Block Chaining. In extended authentication the result of the previous authentication is remembered by the card (stored internally) and used for the subsequent authentication.

This will enable the cards to check that decrement has indeed happenened on the card. The extended authentication mode is reverted back to normal upon power reset or addess reset. This can be carried out with the **MCardChallengeResponse** API with a different ChallengeID.

11.3.11 SLE4404

Attribute	Value
Protocol	Bit protocol
Manufacturer	Infineon
Total Memory size	52 bytes (including counters)
Zones	1
PINs	2 (user code & Memory Code)
Other Variants	Not Known

Fuses:

The cards are of use and throw type. There is a hardware fuse mechanism that is permanently blown with the **MCardWriteMemory** API with the arguments offset 0x3e and length 1 with the data buffer to be written containing the first byte 0x00.
The fuse can be read w ith **MCardReadMemory** API with offset 0x3e and length 1.the byte returned will contain the value of fuse in LSB.

Memory Counter :

The SLE4404 card contains a Counter that **cannot** be decremented using a McardDecrementCounter API.It denotes the number of times the user memory can be erased.

User Code Protection:

This card comes with the user code that has to be verified for writing the contents of scratchpad and the user memory area .The bits112/113 configures the user memory as read/write protected accordingly (For further details refer the card specification). Modifying those bits (byte 0x0e) will change the functionality of the user memory area. The user code can be verified using the **MCardVerifyPIN** API and PIN0.

Memory Code Protection:

The memory code verification (PIN 1) has to be done for erasing the contents of the user memory, using the **MCardVerifyPIN** API with pinNumber1. On successful entry of memory code,the user memory contents are erased (bits set to 1) and the memory counter is decremented by 1(a bit is reset). Even an incorrect entry of memory code will decrement the memory counter and hence the number of times the user memory can be erased. Even after the PIN1 trials are exhausted (meaning that the user memory cannot be erased), the applicati on can still reset the bits (can still write to the remaining area). The memory code **cannot/should not** be changed using the MCardChangePIN API

Contents