SMART MOUSE SERIES

SMART CARD READER/WRITER

REFERENCE MANUAL

for Version 3.0 Software

29 March, 2000

Cambridge, 1996

File: Usref3_0.doc

Contents

1. INTRODUCTION

The GIS Smart Mouse series smart card reader/writer product range is intended to offer a complete solution to the problem of interfacing with the wide variety of available smart cards in the marketplace today. The SM1 reader/writer is a low cost non intelligent smart card reader designed to handle processor based smart cards communicating using an asynchronous protocol. The SM2 reader/writer extends the capability to include the capability to handle a wide variety of non-processor based synchronous memory cards.

The SM3 reader/writer is an intelligent reader/writer that handles both synchronous and asynchronous smart cards. Because it is intelligent, the reader/writer may buffer data and therefore permit a wide variety of communications protocols and speeds to be supported between the reader/writer and the host. In addition, much of the software required to handle smart card dialogues may be held and executed in the reader/writer to relieve the programming load on the host. This may be important when the host is not a PC or similar general purpose computer, but may be a specialised terminal with limited programming capability such as an EPOS device.

The SM3 reader/writer may also be configured to hold user application code. To this end, all software in the SM3 is downloadable from a host. GIS are able to produce user specific application code for the SM3 upon user request.

The Smart Mouse series of smart card reader/writers are supplied with full software support for a PC Windows environment. Support software is supplied as Windows DLLs, and both 16 bit support (Windows 3.1) and 32 bit support (Windows NT, Windows 95) is supplied. The software is designed to provide a simple, compatible, common interface to the calling application and it is therefore compatible across the product range. Because there are no standards about the way that data is read and written to and from synchronous cards, the software presents a common calling methodology for applications dealing with synchronous cards and it is only necessary to first identify the specific card type to the software.

With v3.0 the management of DLLs has been very much simplified with the addition of support for explicit linking as well as the original implicit linking originally implemented in v2.0 DLLs.

2. SM1 ASYNCHRONOUS SMART CARD READER

The SM1 is the entry level smart card reader capable of interfacing to ISO 7816 compatible asynchronous smart cards. The clock rate applied to the inserted card is fixed at 3.57 MHz and the interface is controlled at a fixed baud rate of 9600 bps. An LED is provided to indicate that a card has been inserted fully and that power is applied to that card. The use of low current CMOS cards eliminates the use of an external power supply. Older generation NMOS higher current smart cards will require an external power source to be supplied. This may take the form of an external power supply or an adapter cable enabling power to be taken from a spare games, mouse or keyboard port.

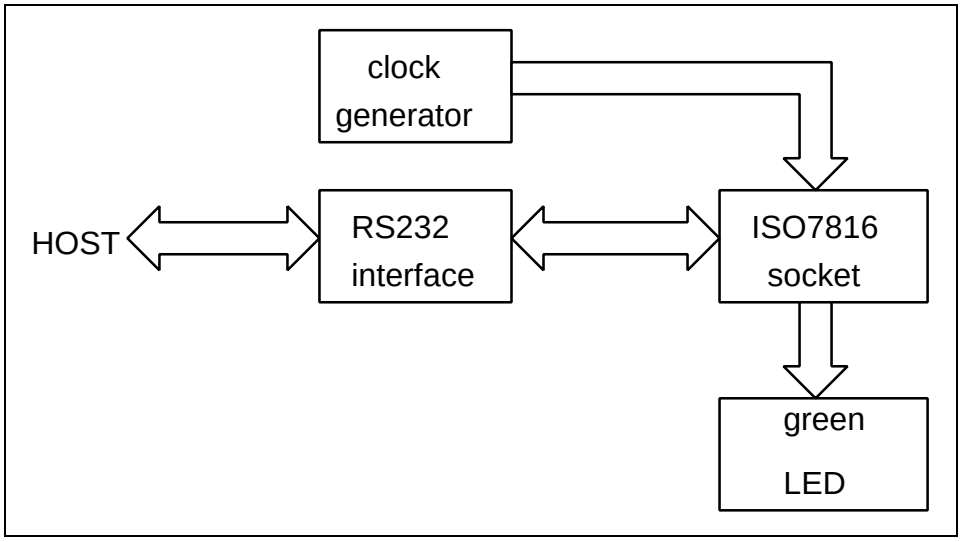


Figure 1. Block Diagram of SM1 hardware

Card Type:	Contact (centre position)
Card Specification:	ISO 7816 1/2/3
Hardware Interface:	RS232C Signal Levels 9600 baud serial (half duplex) Green LED to show card inserted Card reset using RTS Card detection using CTS
Card Insertion Unit:	Six contact wipe type Guaranteed 50,000 insertions Normally closed card detect switch

3. SM2 SYNCHRONOUS & ASYNCHRONOUS SMART CARD READER

The SM2 smart card reader/writer is fully compatible with the SM1 when reading and writing ISO 7816 compatible asynchronous smart cards. The SM2 also contains the added capability to handle synchronous smart cards that require only a single 5v voltage level. A wide variety of synchronous cards may be supported by the hardware and most common types are included. The main limitation on support is the range of software drivers available since most synchronous cards will require a unique driver. New software drivers are being added on a regular basis.

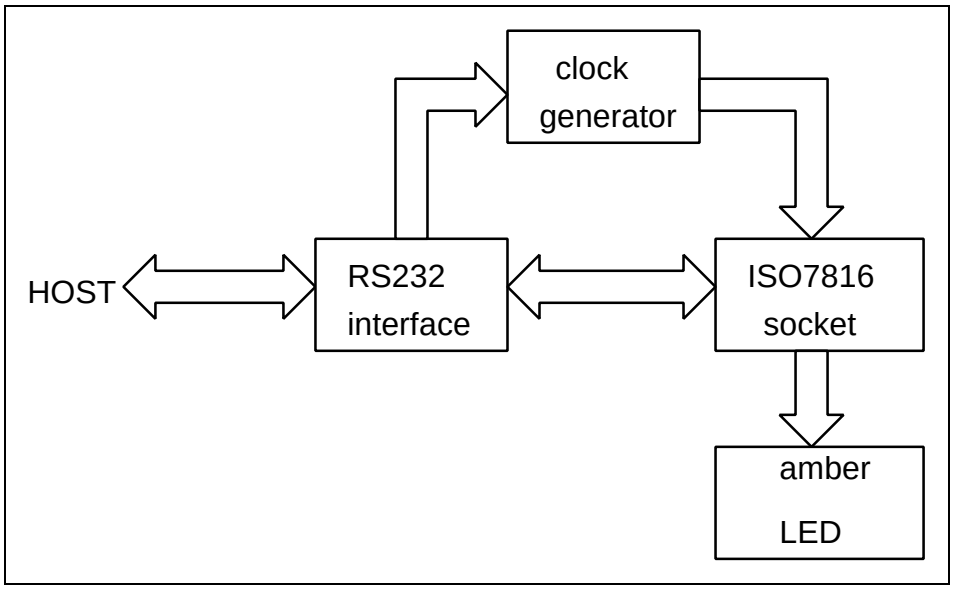


Figure 2. Block Diagram of SM2 Hardware

Card Type:	Synchronous (memory cards) and Asynchronous (processor cards) Contact (centre position)
Card Specification:	ISO 7816 1/2/3
Hardware Interface:	RS232C Signal Levels 9600 baud serial (half duplex), asynch. mode Processor controlled bit rate, synch. mode Amber LED to show card inserted Card reset using RTS, async. mode only Card detection using CTS
Card Insertion Unit:	Six contact wipe type Guaranteed 50,000 insertions Normally closed card detect switch

4. SM3 INTELLIGENT SMART CARD READER

The SM3 smart card reader/writer refines the role of smart card handling hardware from that of a dumb, single protocol device requiring extensive host support, to an intelligent smart card handling system insulating host devices from specific card environments and cards from reader dependency. This is achieved by the inclusion of a powerful on-board processor with local software to support a wide variety of card and host environments.

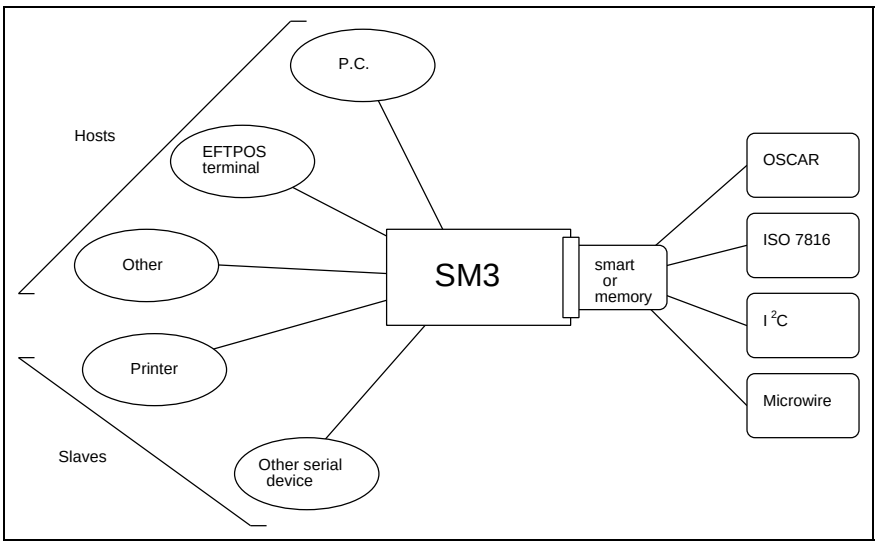


Figure 3. SM3 Functionality

The power of the included processor is sufficient to allow the SM3 to become the 'host' in less complex systems, its standard output being capable of driving simple output devices.

The intelligence provided by the inclusion of an on-board microprocessor allows the SM3 to perform many of the low-level card functions such as resets etc. freeing the host for application tasks. The ability to handle various asynchronous protocols (T=0, T=1 etc.) and synchronous protocols (I^2C, Microwire etc.) means that in this delegated role almost all card interfaces are accepted.

The programmability also gives the SM3 flexibility at the host interface allowing it to operate in an emulation mode, for example possibly appearing to an EFTPOS terminal host as a third party magnetic card reader. This facility would enable a system upgrade from magnetic card to smart card to proceed simply by replacing the card reading 'head' leaving the backbone hardware and software unchanged.

Another use of the native intelligence within the SM3 is to give it the duties of a host in simpler applications, an example might be that of a system requiring a hard copy of some information on a card. In this situation the SM3 would be used as the host for a printer connected via the serial line.

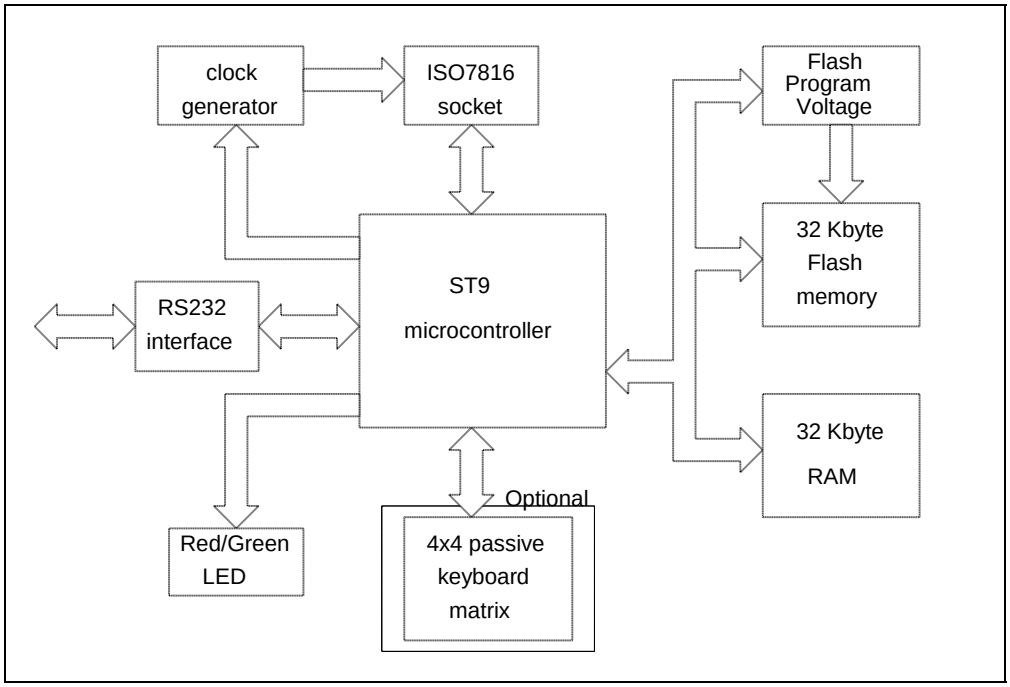
Unlike non-intelligent smart card reader/writers, the SM3 can take a variety of positions in the chain between card and application since the precise split in functionality between the SM3 and the host is definable by advanced applications developers.

- ST9 8/16 bit CMOS 24MHz Processor with
- 512bytes of on-board RAM
- 8 kbyte on-board ROM
- 32kbyte RAM
- 32kbyte Flash PROM with on-board voltage generator
- Programmable frequency multiplier from 2-20MHz

  This allows smart cards operating faster than 9600 baud to be presented with their maximum clock frequency while matching the resultant serial baud rate, e.g. standard 3.5795MHz clock will give a maximum 19.2k baud rate, increasing this to the 5MHz clock of appropriate cards gives 26.8k baud etc.

- Programmable RS232C serial port up to 19.2k baud full duplex
- Programmable Green/Amber/Red LED to show card inserted/comms
- Optional 4x4 keyboard

Figure 4. Block Diagram of SM3 hardware.



5. SOFTWARE PHILOSOPHY & OVERVIEW

The design of the software environment for the Smart Mouse smart card product set will allow reader/writer types and smart card types to be interchanged with the minimum of application modification. A compatible set of software is supplied for all reader types; the SM1 and SM2 non-intelligent reader/writers sharing one set of software, while the SM3 intelligent reader/writer is support by another set of software. Both software sets have a compatible function set.

The support software is supplied for a PC Windows environment in DLL form. DLLs are available for the SM1/SM2 products and SM3 product operating in both 16 bit (Windows 3.1) and 32 bit (Windows NT) environments. It is only necessary for the application to call the appropriate DLL to obtain the desired support. Clearly some functions will apply to one environment and not others, for example, the card eject function will only apply to an SM3 intelligent smart card reader fitted with an appropriate card connector. Where such support is not relevant, the function is omitted.

In the case of synchronous memory smart cards, only those requiring a single 5 volt power line are supported. In general this means that all newer synchronous cards are supported while some older ones will be excluded. There is no defined standard for the way that synchronous memory cards are addressed and consequently, specific support is required for each card or card family type. Many such cards are supported in the present software, with more being added all the time. New software releases will be made from time to time in order to distribute new levels of support.

At the application interface, it is desirable to arrange a standard mechanism for talking to synchronous cards. This has been achieved in the supplied software by defining a 'standard synchronous card' using a protocol somewhat similar to the ISO standard asynchronous protocol. It is a function of the support software to translate these function calls into the actual protocol of the card in use. In order to achieve this, the software needs to know the card type being addressed, and in the case of synchronous cards, it is not possible to determine this from the card in advance of knowing how to talk to the card. Therefore, it is necessary for the application to indicate the card type to the support software. A special function call is provided to enable this action.

In the case of asynchronous cards, there are various options within the existing ISO 7816 standard. The base standard describes a byte oriented protocol known as T=0. An improvement on this protocol wraps a header and trailer around this protocol to form a packet. This block protocol is known as T=1. At the present time, very few cards support the T=1 protocol and this support is absent from the present support software. The T=1 protocol will be included in the near future.

An extension to the ISO 7816 standard, known as ISO 7816-4 defines the specific card application level commands more specifically as an inter-industry command set. These commands permit the card to receive and transmit data in a single request-response message pair. Part 4 support, as it is known, is included within the support software. The part 4 support is independent of the transport protocol and will transparently select the protocol (T=0, T=1 etc.) supported by the card.

6. SOFTWARE FUNCTIONS

The compatible set of functions is listed below. Where a function does not apply to all of the SM1, SM2 and SM3, the function is marked to indicate where its use is relevant. In other reader/writer software support environments, the function will not be included.

In the functions described below, the types used in the function prototypes are given in Polish notation, as used by all Windows 3.1 and Windows NT API calls, and are defined in windows.h.

The functions below are set down in three groups, reader/writer control functions, asynchronous card functions and synchronous card functions. In the case of reader/writer control functions, some of these functions are only applicable to the SM3 intelligent reader. In these cases, the applicability is noted.

6.1 Smart Mouse Reader/Writer Functions

6.1.1 GisSmBaudRate

`BOOL GisSmBaudRate(PORT_HANDLE port,WORD BaudRate)`

This function applies to the SM3 intelligent reader only.

This function sets the baud rate for subsequent communications to the reader/writer (but not the card).

The BaudRate argument can be any valid Windows supported baud rate from 300 baud to 19,200 baud.

Note: This function may need to reset the SM3 reader/writer if the baud rate being set is different from that previously set in the reader; this will cause any card that is currently loaded to power off. Therefore, this function should only be called when a reader/writer reset is appropriate, usually immediately after opening the port with GisSmOpen and probably with no card present.

Arguments :	PORT_HANDLE port - handle of port to address.
Returns :	BOOL - indicating success/failure of the command. 0 - Failure 1 - Successful

6.1.2 GisSmCardEject

`BOOL GisSmCardEject(PORT_HANDLE port)`

This function applies to the SM3 intelligent reader only.

Ejects the card from SM3 attached to the specified port, provided that the card connector fitted to the reader/writer supports the eject function. The standard SM3 does not support this function.

Arguments :	PORT_HANDLE port - handle of port to address.
Returns :	BOOL - indicating success/failure of the command. 0 - Failure - Call GisSmError 1 - Successful

6.1.3 GisSmClose

`BOOL GisSmClose(PORT_HANDLE port)`

Closes the currently open serial port

Arguments :	PORT_HANDLE port - handle of port to close.
Returns :	BOOL - indicating success/failure. 0 - Failure 1 - Successful

Notes: With an SM3, closing a Com port does not change the state of the Smart Mouse reader/writer, but simply releases the handle. To remove power from the card GisSmCardDisable should be called prior to closing.

With an SM1 or SM2, closing the port will remove power from the card.

6.1.4 GisSmError

`WORD GisSmError(void)`

This function applies to the SM3 intelligent reader only.

Requests the status of the most recently called GisSm command. Note that the port is implied by the port value of the last command.

Arguments :	VOID
Returns :	WORD - signalling the error type as defined in gis_sm.h.

Three groups of error conditions are defined, the first are simple errors, the second indicate internal errors which should be reported to the supplier and the third indicate more complex errors which may require user or application action. The error values are defined in the supplied header files.

CALL_OK	Command completed successfully
UNDEFINED	Undefined error
NO_CALL_YET	No call has been made yet.

Internal errors, report to supplier.

ARG_TOO_MANY_ARGS
ARG_BAD_RETURN_TYPE
ARG_TOO_MANY_FIELDS
ARG_BAD_FIELD_TYPE
ARG_BUFFERS_TOO_BIG_V
ARG_BUFFERS_TOO_BIG_F
ARG_BAD_ARG_TYPE
ARG_BAD_MODULE_ID
ARG_BAD_FUNCTION_ID
ARG_BAD_RPC_CALL

Fault with the SM3 unit. No part of the instruction was executed.

SEND_FAIL	Failure while sending request to SM3. This is due to some failure in the serial link or the SM3 unit -check power to the SM3 and the serial connection.
RECEIVE_FAIL	Failure during reception of call results from SM3 after having performed request. (Due to comms failure.)
PORT_NOT_OPEN	Bad port handle used, probably because GisSmOpen failed.
SM3_NOT_RESPONDING	Fault with SM3. No part of the instruction sent was executed.
RECEIVE_TIMEOUT	The SM3 took too long to perform the requested action.

6.1.5 GisSmLed

`BOOL GisSmLed(PORT_HANDLE port, BYTE colour)`

This function applies to the SM3 intelligent reader only.

Controls the light emitting diode (LED) on the specified SM3, the argument selects the colour.

Arguments :	PORT_HANDLE port - handle of port to address
	BYTE colour - colour of the LED - see below
Returns :	BOOL - indicating success/failure of the command 0 - Failure - Call GisSmError 1 - Successful

LED_OFF	Turns LEDs off
LED_RED	Turns LED red
LED_GREEN	Turns LED green
LED_AMBER	Turns LED amber

These constants are defined in the supplied header files.

6.1.6 GisSmOpen

`PORT_HANDLE GisSmOpen(WORD com_port)`

Opens the specified serial port.

Arguments :	WORD com_port - port to open - e.g. com1 = 1, com2 = 2
Returns :	PORT_HANDLE - the handle of the opened com port.

The constant OPEN_FAILED is returned if the port could not be opened.

Note: In a Windows NT 32 bit environment, the returned handle is only valid within the current process, but may be used by any thread within the process.

6.1.7 GisSmReaderReset

`BOOL GisSmReaderReset(PORT_HANDLE port, LPBYTE buffer, LPWORD count)`

This function applies to the SM3 intelligent reader only.

Resets the SM3 card reader, returning the reader's answer-to-reset in the supplied buffer.

Note: this function takes approximately 3 seconds to complete.

Arguments :	PORT_HANDLE port - handle of port to address.
	LPBYTE buffer - buffer to receive message
	LPWORD count - initially the size of the buffer offered by the application program, after the call it represents the length of the message placed in the buffer.
Returns :	BOOL - indicating success/failure of command 0 - Failure 1 - Successful

6.1.8 GisSmReaderStatus

`BYTE GisSmReaderStatus(PORT_HANDLE port)`

Returns the status of the Smart Mouse device on the specified port as a bit field.

Arguments :	PORT_HANDLE port - handle of port to address.
Returns :	BYTE - Bit mask containing the following flags.

RETURN VALUE	**COMMENT**
STATUS_CARD_FULLY_IN	Card fully inserted into reader
STATUS_CARD_ALMOST_IN (*SM3 only*)	Only returned by card reader/writers with connectors supporting this function.
STATUS_LED_RED (*SM3 only*)	Red LED on
STATUS_LED_GREEN (*SM3 only*)	Green LED on
STATUS_FAILED (*SM3 only*)	This indicates SM3 error, call GisSmError.

These constants are defined in the supplied header files.

6.1.9 GisSmVersion

`BOOL GisSmVersion(LPBYTE version)`

Returns the DLL version information.

Arguments :	LPBYTE version - pointer to buffer to receive version information. Supplied buffer must be at least 30 bytes long.
Returns :	BOOL - indicating success/failure of the command 0 - Failure - Call GisSmError 1 - Successful

6.2 Asynchronous Smart Card Functions

6.2.1 GisSm7816Reset

```
signed short GisSm7816Reset(PORT_HANDLE port, LPBYTE message,
     LPWORD count)
```

Performs a standard ISO 7816-3 smart card reset operation, returning the Answer-to-Reset message.

Note : This must be called before any asynchronous smart card operations are attempted.

Arguments :	PORT_HANDLE port - handle of port to address.
	LPBYTE message - buffer to receive answer-to-reset message.
	LPWORD count - initially the size of the buffer offered by the application program, after the call it represents the length of the message placed in the buffer or the number of bytes which would have been placed in the buffer had it been sufficiently large.
Returns :	signed short - see below

Return Values:

Value	**Description**
OK	The card was successfully reset.
SM3_PROBLEM (SM3 only)	There is a problem with the communications to reader/writer - Call GisSmError.
CARDOUT	Card removed before operation complete
TIMEOUT	Card failed to respond before expiry of timer
UNKNOWN_TS	Unknown TS in the ATR, possibly a bad card.
BAD_TCK	The checksum of the ATR is incorrect (only when TCK is sent).

6.2.2 GisSmCardDisable

`BOOL GisSmCardDisable(PORT_HANDLE port)`

Disables the card by removing power. The card must be reset using GisSmCardReset before any further asynchronous card operations are attempted.

Arguments :	PORT_HANDLE port - handle of port to address.
Returns :	BOOL - indicates the success/failure of the command. 0 - Failure - Call GisSmError 1 - Successful

6.2.3 GisSmPart4Cmd

`BOOL GisSmPart4Cmd (PORT_HANDLE port, LPADPU Adpu, LPBYTE BuffOut, LPBYTE BuffIn, LPPart4Error E);`

For asynchronous smart cards, performs a standard ISO 7816 part 4 type command.

Arguments :	PORT_HANDLE port - handle of port to address.
	LPADPU Adpu - pointer to ADPU structure. See Appendix A1.
	LPBYTE BuffOut - pointer to buffer containing data to be sent.
	LPBYTE BuffIn - pointer to buffer into which received data is placed.
	LPPart4Error E - pointer to LPPart4Error structure (identical to LPT0Error). See Appendix A1. This is only valid when the Function returns 1.
Returns :	BOOL - indicating success/failure of the function. 0 - Failure - Call GisSmError 1 - Successful

Note: This function assumes that the Smart card supports the Get Response command, with the same class as ADPU and an instruction number (code) of C0 (hex),

6.2.4 GisSmT0ReadCmd

```
BOOL GisSmT0ReadCmd (PORT_HANDLE port, LPT0Instruction ins,
     LPBYTE buff, LPT0Error err)
```

For asynchronous cards only, performs a standard ISO 7816-3 T=0 read type command (i.e. one which receives data from the card).

Arguments :	PORT_HANDLE port - handle of port to address.
	LPT0Instruction ins - pointer to instruction structure. See Appendix A1.
	LPBYTE buff - Buffer for incoming data
	LPT0Error err - Pointer to structure, contains SW1 and SW2 T=0 error codes after completion. See Appendix A1. This is only valid when the Function returns 1.
Returns :	BOOL - indicating success/failure of the function 0 - Failure - Call GisSmError 1 - Successful

Note: As defined in ISO7816-3 , a P3 value of 0 indicates a read of 256 bytes from the card.

6.2.5 GisSmT0WriteCmd

```
BOOL GisSmT0WriteCmd (PORT_HANDLE port, LPT0Instruction ins,
     LPBYTE buff, LPT0Error err)
```

For asynchronous cards only, performs a standard ISO 7816-3 T=0 write-type command (i.e. one which sends data to the card).

Arguments :	PORT_HANDLE port - handle of port to address.
	LPT0Instruction ins - pointer to instruction structure. See Appendix A1.
	LPBYTE buff - Buffer for outgoing data
	LPT0Error err - Pointer to structure, contains SW1 and SW2 T=0 error codes after completion. See Appendix A1.This is only valid when the Function returns 1.
Returns :	BOOL - indicating success/failure of the function 0 - Failure - Call GisSmError 1 - Successful

6.3 Synchronous Smart Card Functions

6.3.1 Catalyst '3-wire' cards

6.3.1.1 GisCat3Setup

`Bool GisCat3Setup(PORT_HANDLE port, WORD subtype)`

Configures the software to correctly communicate with a card of the specified sub-type (see section 7.2.2).

Arguments:	PORT_HANDLE port - handle of port to address
	WORD value - number of data and address bits see table section 7.2.1.
Returns:	BOOL - indicates the success/failure of the command 0 - Failure 1 - Success

6.3.1.2 GisCat3ReadWord

`Word GisCat3ReadWord(PORT_HANDLE port, WORD addr)`

Reads a single word from the specified address on the card.

Arguments:	PORT_HANDLE port - handle of port to address
	WORD addr - address from which data is read
Returns:	WORD - indicates the failure or the data of the command -1 - Failure other values - Data read from card

6.3.1.3 GisCat3WriteWord

`Bool GisCat3WriteWord(PORT_HANDLE port, WORD addr, WORD val)`

Writes the word in 'val' to the location specified by addr.

Arguments:	PORT_HANDLE port - handle of port to address
	WORD addr - location of write
	WORD val - value to write
Returns:	BOOL - indicates the success/failure of the command 0 - Failure 1 - Success

6.3.1.4 GisCat3EraseWord

`Bool GisCat3EraseWord(PORT_HANDLE port, WORD addr)`

Erases (to state '1') the specified address.

Arguments:	PORT_HANDLE port - handle of port to address
	WORD addr - address to erase
Returns:	BOOL - indicates the success/failure of the command 0 - Failure 1 - Success

6.3.1.5 GisCat3Read

`Bool GisCat3Read(PORT_HANDLE port, WORD addr, LPBYTE buffer, WORD length)`

Reads 'length' bytes starting from address specified by 'addr' and storing the data in 'buffer'.

[Byte-ordering for cards with two data bytes is LSB first, the same as Intel-based P.C.'s.)

Arguments:	PORT_HANDLE port - handle of port to address
	WORD addr - start address
	LPBYTE buffer - storage area for data
	WORD length - number of bytes to store
Returns:	BOOL - indicates the success/failure of the command 0 - Failure 1 - Success

6.3.1.6 GisCat3Write

```
Bool GisCat3Write(PORT_HANDLE port, WORD addr, LPBYTE buffer,
     WORD length)
```

Writes 'length' bytes starting from address specified by 'addr', using the data in 'buffer'.

[Byte-ordering for cards with two data bytes is LSB first, the same as Intel-based P.C.'s.)

Arguments:	PORT_HANDLE port - handle of port to address
	WORD addr - start address
	LPBYTE buffer - source storage area for data
	WORD length - number of bytes to store
Returns:	BOOL - indicates the success/failure of the command 0 - Failure 1 - Success

6.3.1.7 GisCat3Erase

```
Bool GisCat3Erase(PORT_HANDLE port, WORD addr, WORD length)
```

Erases (to state '1') the addresses starting at 'addr' and covering the next 'length' bytes.

Arguments:	PORT_HANDLE port - handle of port to address
	WORD addr - start address
	WORD length - number of bytes to erase
Returns:	BOOL - indicates the success/failure of the command 0 - Failure 1 - Success

6.3.1.8 GisCat3WriteDisable

```
Bool GisCat3WriteDisable(PORT_HANDLE port)
```

Prevents all writes to the port specified.

Arguments:	PORT_HANDLE port - handle of port to address
Returns:	BOOL - indicates the success/failure of the command 0 - Failure 1 - Success

6.3.1.9 GisCat3WriteEnable

`Bool GisCat3WriteEnable(PORT_HANDLE port)`

Enables writes to the specified port.

Arguments:	PORT_HANDLE port - handle of port to address
Returns:	BOOL - indicates the success/failure of the command 0 - Failure 1 - Success

6.3.1.10 GisCat3EraseAll

`Bool GisCat3EraseAll(PORT_HANDLE port)`

Erases (to state '1') the entire card.

Arguments:	PORT_HANDLE port - handle of port to address
Returns:	BOOL - indicates the success/failure of the command 0 - Failure 1 - Success

6.3.1.11 GisCat3WriteAll

`Bool GisCat3WriteAll(PORT_HANDLE port, WORD value)`

Writes to the entire card.

Arguments:	PORT_HANDLE port - handle of port to address
	WORD value - value to write to all locations
Returns:	BOOL - indicates the success/failure of the command 0 - Failure 1 - Success

6.3.2 I2C Cards

6.3.2.1 GisI2cInit

`Bool GisI2cInit(PORT_HANDLE port)`

Initializes the specified port. This command must be used before any other I2C low-level command.

Arguments:	PORT_HANDLE port - handle of port to address
Returns:	BOOL - indicates the success/failure of the command 0 - Failure 1 - Success

6.3.2.2 GisI2cSetType

`Bool GisI2cSetType(PORT_HANDLE port, BYTE SlaveAddress, BYTE NumAddressBytes, WORD WriteCacheSize)`

Configures the software to correctly communicate with a card with the specified parameters (see section 7.2.1). This must immediately follow a GisI2cInit command.

Arguments:	PORT_HANDLE port - handle of port to address
	BYTE SlaveAddress - Device address as per I2C specification.
	BYTE NumAddressBytes - Number of address bytes (usually 1 or 2).
	WORD WriteCacheSize - Size of card's page write buffer (as per I2C specification).
Returns:	BOOL - indicates the success/failure of the command 0 - Failure 1 - Success

6.3.2.3 GisI2cReadNextByte

`Short GisI2cReadNextByte(PORT_HANDLE port)`

Reads a single byte from the next address on the card.

Arguments:	PORT_HANDLE port - handle of port to address
Returns:	SHORT - indicates the success/failure of the command -1 - Failure other values - Data read from card

6.3.2.4 GisI2cReadByte

`Short GisI2cReadByte(PORT_HANDLE port, WORD addr)`

Reads a single byte from the specified address on the card.

Arguments:	PORT_HANDLE port - handle of port to address
	WORD addr - address from which data is read.
Returns:	SHORT - indicates the success/failure of the command -1 - Failure other values - Data read from card

6.3.2.5 GisI2cWriteByte

`Bool GisI2cWriteByte(PORT_HANDLE port, WORD addr, BYTE Value)`

Writes a single byte to the specified address on the card.

Arguments:	PORT_HANDLE port - handle of port to address
	WORD addr - address to which data is to be read.
	BYTE Value - value to be stored.
Returns:	BOOL - indicates the success/failure of the command 0 - Failure 1 - Success

6.3.2.6 GisI2cRead

`Bool GisI2cRead(PORT_HANDLE port, WORD addr, LPBYTE buffer, WORD length)`

Reads 'length' bytes starting from address specified by 'addr' and storing the data in 'buffer'.

Arguments:	PORT_HANDLE port - handle of port to address
	WORD addr - start address
	LPBYTE buffer - storage area for data
	WORD length - number of bytes to store
Returns:	BOOL - indicates the success/failure of the command 0 - Failure 1 - Success

6.3.2.7 **GisI2cWrite**

`Bool GisI2cWrite(PORT_HANDLE port, WORD addr, LPBYTE buffer, WORD length)`

Writes 'length' bytes starting from address specified by 'addr', using the data in 'buffer'.

Arguments:	PORT_HANDLE port - handle of port to address
	WORD addr - start address
	LPBYTE buffer - source storage area for data
	WORD length - number of bytes to store
Returns:	BOOL - indicates the success/failure of the command 0 - Failure 1 - Success

7. APPENDIX 1.

7.1 Asynchronous Smart Card Data Structures

ISO 7816-3 T=0 and ISO 7816-4 type commands are sent to a Smart Mouse smart card reader/writer using either the functions GisSmT0ReadCmd and GisSmT0WriteCmd, or the function GisSmPart4Cmd. If the former is used the command must be as a T0Instruction structure, and if the latter then the ADPU structure must be used.

7.1.1 T0Instruction

```
typedef struct
{
    WORD  cla;        /* Class            */
    WORD  ins;        /* Instruction      */
    WORD  p1;         /* Param 1          */
    WORD  p2;         /* Param 2          */
    WORD  p3;         /* Length of data   */
} T0Instruction;

typedef T0Instruction *LPT0Instruction;
```

7.1.2 ADPU

```
typedef struct
{
    WORD cla;         /* Class            */
    WORD ins;         /* Instruction      */
    WORD p1;          /* Param 1          */
    WORD p2;          /* Param 2          */
    WORD Lc;          /* Outgoing Length  [0..255] */
    WORD Le;          /* Incoming Length  [0..256] */
} ADPU;

typedef APDU *LPAPDU;
```

7.1.3

T0Error

```
typedef union
{
   struct
   {
               BYTE sw2;
               BYTE sw1;
               WORD err;
   } b;
   struct
   {
               WORD sw12;
               WORD err;
   } w;
} T0Error;

typedef T0Error *PT0Error;
typedef T0Error FAR *LPT0Error;
```

7.2 Synchronous Smart Card Data Structures

7.2.1 CARD_TYPE

Used to define a card type.

I2C cards use ‘AddrBytes’ and ‘Wcachesize’ others use ‘param1’ and ‘param2’.

```
typedef WORD   MINOR_TYPE;
typedef int    MAJOR_TYPE;

typedef struct
{
   MAJOR_TYPE  major;
   MINOR_TYPE  minor;
   union
   {
               WORD            AddrBytes;    //Address Bytes
               WORD            param1;
   };
   union
   {
               WORD            WCacheSize;   //Write Cache
size
               WORD            param2;
   };

} CARD_TYPE;
```

The structure is used slightly differently for 3-wire and I^2C cards.

3-Wire

The number of address bits and data bits are set by choosing the correct sub-type from the table below and setting this as the minor type in the structure.

i.e. a 3-wire card with 9-bit address and 8 data bits is sub-type 4 and therefore minor type 4.

Major	**CAT_CARD**		
Minor	**Sub-Type**	*Address bits*	*Data bits*
(one of)	**0**	*7*	*8*
	1	*6*	*16*
	2	*8*	*8*
	3	*7*	*16*
	4	*9*	*8*
	5	*8*	*16*
	6	*10*	*8*
	7	*9*	*16*
	8	*11*	*8*
	9	*10*	*16*

param1 and param2 are reserved for future expansion.

I^2C

I^2C cards are accessed on a device by device basis, that is if two or more devices occupy a card then they are regarded as separate. A GisSmSynCard command with a different minor type is required to select a device with a different slave address.

Major	**I2C_CARD**
Minor	**slave address**

In addition the address bytes (1 or 2) and the write cache must also be correctly set.

N.B. If the write cache in particular is not set correctly the software is unable to detect this and the error may lead to write errors - proper use of the memory images should enable the correct card type and hence write cache size to be set.

7.2.2 MEM_IMG

Used to hold a 'memory image' consisting of a data buffer, its length and the location (address) on the card at which the data starts.

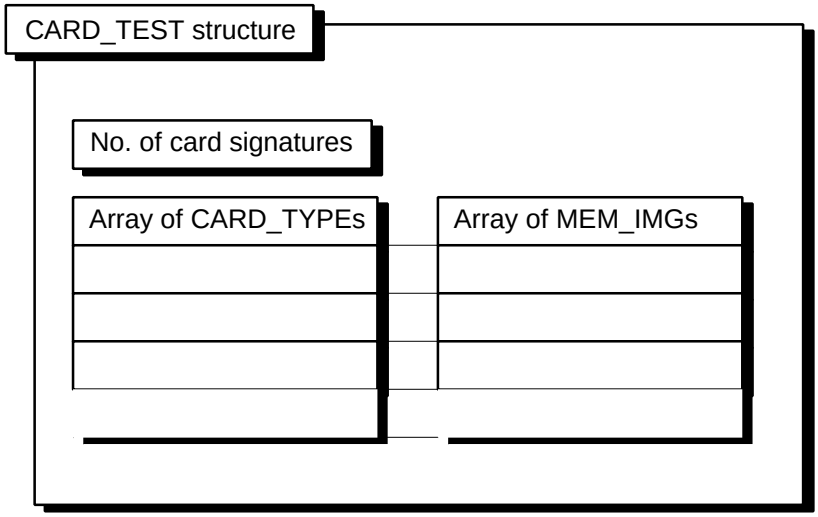
```
typedef struct
{
   LPBYTE       buffer;
   WORD         buffer_count;
   WORD         location;
} MEM_IMG;
```

Note that the buffer contains byte-wise data, if the card uses a wider data size then the correct byte ordering must be set otherwise the image will not be recognised.

7.2.3 CARD_TEST

Holds a list of memory images and a list of card types.

(A CARD_TYPE structure with an associated MEM_IMG structure is sometimes referred to as a card signature.)




```
typedef struct
{
   WORD card_sig_count   //number of card signatures
   LPCARD_TYPE card_type_list;//ptr to first card type
   LPMEM_IMG    mem_img_list;    //ptr to first mem_img
} CARD_TEST;
```

7.2.4 GEN_ERR

Holds full error breakdown, see 0 GisSmError for further details.

```
typedef struct
{

   BYTE    type;   //Error type
   WORD    no;     //Error number
   BYTE    qual;   //Qualifier
   WORD    smerr;  //Reader error from GisSmError()

} GEN_ERR;
```

8. APPENDIX 2.

8.1 PC Software Installation

The supplied disc contains software to operate the SM1,SM2 and SM3 under Windows 3.1 or Windows NT. All the directories should be copied to a suitable location on the target machine.

a:\include

`gis_sync.h` Header file for asynchronous card functions.

`gis_asyn.h` Header file for other synchronous card specific functions.

a:\sm1\16bit\bin\gissm12w.dll

The 16bit DLL file for SM1 and SM2 should be copied into a directory on the DOS path or the windows system directory. Alternatively edit the PATH command in the `autoexec.bat` file to include a new directory holding this file.

a:\sm1\16bit\lib\gissm12w.lib

This LIB file should be linked into any application which calls the DLL when implicit linking is being used.

a:\sm1\include\gis_sm1.h

Header file for SM1 and SM2 card functions.

a:\sm3\16bit\bin\gissm3w.dll

The 16bit DLL file for SM3 should be copied into a directory on the DOS path or the windows system directory. Alternatively edit the PATH command in the `autoexec.bat` file to include a new directory holding these files.

a:\sm3\16bit\lib\gissm3w.lib

This LIB file should be linked into any application which calls the DLL when implicit linking is being used.

a:\sm3\32bit\bin\gissm3w.dll

The 32bit DLL file for SM3 for use with Windows

a:\sm3\32bit\lib\gissm3w.lib

This LIB file should be linked into any application which calls the DLL when inplicit linking is being used.

a:\sm3\include\gis_sm3.h

Header file for SM3 card functions.

a:\ sm3\flash

The `sm3.trm` file customises the Windows Terminal program to allow easy updates to the SM3's 'flash' memory (see 10.1 SM3 Internal Software Upgrade)

The `rpcshell.txt` file contains the current version V2_0 of SM3 flash code.

9. APPENDIX 3.

9.1 SM3 Software Architecture

9.1.1 Structure

The SM3 driver software follows a broadly hierarchical structure in which basic functions are supplied to value-adding modules which in turn supply more advanced functions.

An important source of the system's flexibility is the provision of an internal Remote Procedure Calling (RPC) system which allows the PC to use functions on the SM3 as if they were running locally. This has allowed the functions of the system and the component modules to be split between the processor on the SM3 and the processor on the PC, optimising the system's performance especially with respect to low-level card functions.

The standard package includes a variety of drivers for asynchronous and synchronous smart card types following the ISO 7816 standard. All of these or any combinations can be loaded into the SM3. Advanced developers are able to write additional modules to be loaded into the SM3 and called via the RPC mechanism from applications on the PC.

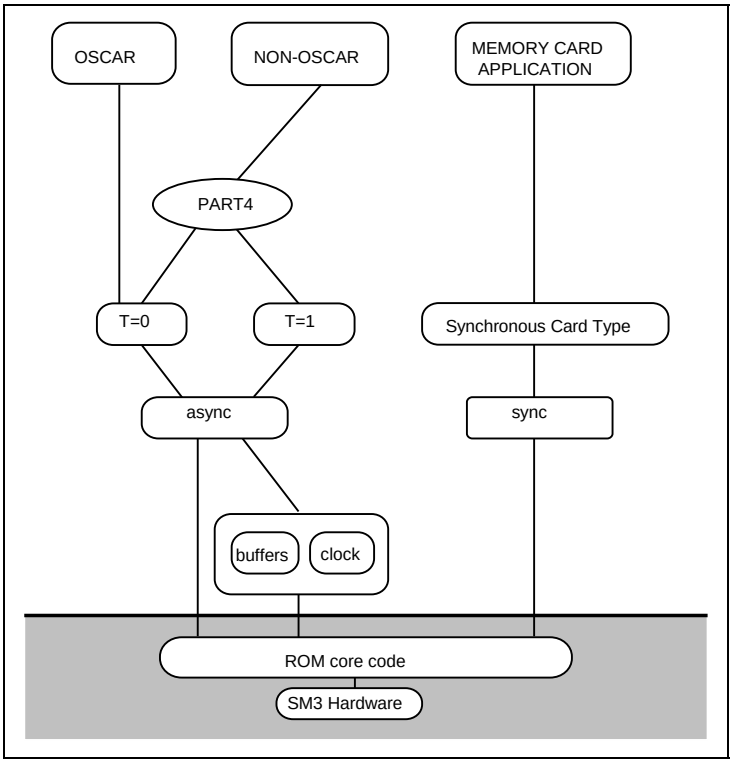


Figure 5. Functional overview of SM3 software.

9.1.2 The Remote Procedure Call Mechanism

The Remote Procedure Call mechanism (RPC) provides a reliable method for the invocation of functions running in the processor of the SM3 based on call-reply-acknowledge semantics with retries and time-outs.

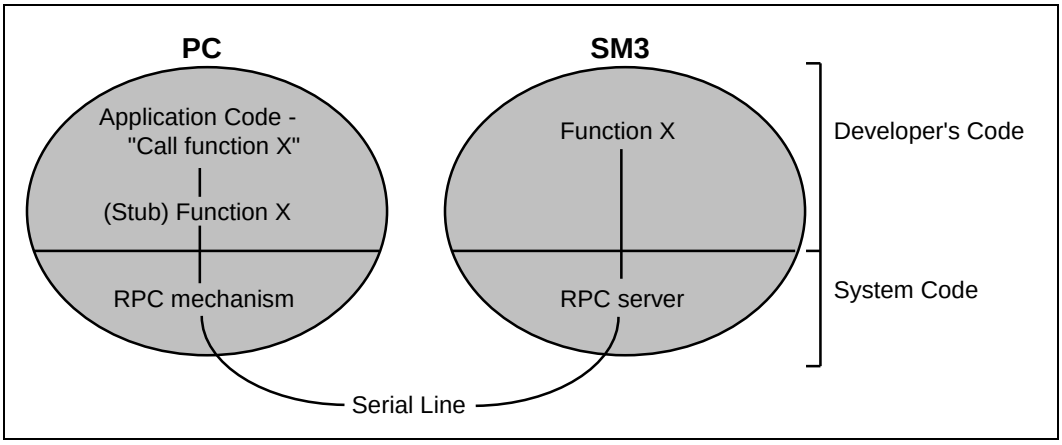


Figure 6. RPC architecture

The diagram above shows the flow of control from an application program running on the P.C. as it calls a remote function "X".

- On the local machine a surrogate for X called a "stub" is invoked by the call.
- (This stub has an identical prototype to the remote function and therefore allows compile time type-checking.)
- The stub calls the system RPC function with the required function and arguments.
- The RPC mechanism transports the call to the server on the SM3 which passes the call to the code body of function X.

All the standard protocol modules are supplied with their stub functions as a 16-bit DLLs for Windows 3.1 and 32-bit DLLs for Windows NT/Win32. This means that RPC mechanism is almost invisible to the application developer.

The RPC mechanism allows applications to access multiple SM3s on all the available serial ports.

Although remote procedure calls follow the semantics of local procedure calls quite closely, unlike local calls, RPCs have the possibility of failure. This may be caused by invalid module or function IDs, failure of the serial link, reader failure or other system failures but should always be regarded as a possibility and checked for.

The functions in the GisSm3 module have been written with this possibility in mind and all return a RPC error code in some form.

10. APPENDIX 4.

10.1 SM3 Internal Software Upgrade

This appendix covers the simple user installation of upgrades to the internal software of the SM3.

From time to time GIS may find it necessary to provide software upgrades to the modules resident in the SM3's 'flash' (non-volatile) memory. This software will be provided in the form of a file containing Intel hex format code. The instructions below should be carefully followed to install this new code.

1. Connect the SM3 to the host machine.

2. Start Windows.

3. Start the 'terminal' application.

4. Load the terminal configuration file "a:\V2_0\flash\sm3.trm" from the supplied disc into the Windows Terminal program. Check that the following values are set - 9600 baud, hardware flow control, no parity, 8 data bits, 1 stop bit and that the serial port selected is the one to which the SM3 is attached.

5. Press the 'reset' switch on the SM3 (when present) or power the SM3 off then on. In each case the LED should briefly flash amber.

6. Immediately after the initial message 'SM3:' is displayed, type ESC e. This will halt the boot routine and start an administration shell.

7. Click on the “Zero Flash” button at the bottom of the terminal window.

8. After a short pause click the “Erase Flash” button. If the value shown is more than five less than (hex) 8000 then click the button again until the value is larger.

9. Select "Transfers" "Send text file" from the menu, select the upgrade file (something like `rpcshell.txt)`. This will spool the upgrade file into the SM3.

10. Quit the terminal program once the transfer is complete and reset the SM3 (power off then on).

11. APPENDIX 5.

11.1 Features Supported

SM3 does not support Cat3Wire. Furthermore, I2C maybe somewhat sluggish on the NT, Win95 platforms for SM1 and SM2.

12. APPENDIX 6.

12.1 Bibliography

ISO 7816 PART 1 [1987]

Identification cards - Integrated circuit(s), cards with contacts part 1 : Physical characteristics

ISO 7816 PART 2 [1988]

Identification cards - Integrated circuit(s), cards with contacts part 3 : Dimensions and location of contacts.

ISO 7816 PART 3 [1989]

Identification cards - Integrated circuit(s), cards with contacts part 3 : Electronic signals and transmission protocols.

ISO 7816 PART 4 [1992]

Identification cards - Integrated circuit(s), cards with contacts part 4 : Inter-industry commands for interchange