4779 Hybrid Smart Card Device

IBM

Device Resident Application Programming Guide

SA34-2361-01

4779 Hybrid Smart Card Device

IBM

Device Resident Application Programming Guide

SA34-2361-01

Note!

Before using this information and the product it supports, be sure to read the general information under “Notices” on page vii.

Second Edition (March 1997)

Changes are made periodically to the information herein; before using this publication in connection with the operation of IBM systems, consult your IBM representative to be sure you have the latest edition and any Technical Newsletters.

IBM does not stock publications at the address given below; requests for IBM publications should be made to your IBM representative or to the IBM branch office serving your locality.

A form for reader’s comments is provided at the back of this publication. If the form has been removed, comments may be addressed to IBM Corporation, Department 56I, 8501 IBM Drive, Charlotte, NC 28262-8563, U.S.A. IBM may use or distribute any of the information you supply in any way it believes appropriate without incurring any obligation to you.

Copyright International Business Machines Corporation 1996, 1997. All rights reserved.
Note to U.S. Government Users — Documentation related to restricted rights — Use, duplication or disclosure is subject to restrictions set forth in GSA ADP Schedule Contract with IBM Corp.

Contents

Notices . . . vii
License . . . viii
Trademarks . . . viii

About This Publication . . . ix
Who Should Read This Book . . . ix
How This Book Is Organized . . . ix
Related Publications . . . x
References . . . x

Chapter 1. Overview of Programming a 4779 Device Resident Application 1-1

Chapter 2. Creating a 4779 Device Resident Application . . . 2-1
Writing the Main Program Module . . . 2-2
  Entry Code . . . 2-2
  Establishing Serial Communications . . . 2-2
  Application Initialization . . . 2-3
  Communications Buffer Read and Message Processing . . . 2-3
  Main Program Considerations . . . 2-5
Accessing the Security Functions . . . 2-6
Security Function Communication Example . . . 2-8
Compiling the Device Resident Application . . . 2-9
Linking the Device Resident Application . . . 2-9

Chapter 3. 4779 Run-Time Library . . . 3-1
How To Use the Libraries . . . 3-1
Interface . . . 3-1
  Serial Communications . . . 3-2
  ICC (Integrated Circuit Card) Reader . . . 3-4
  Display . . . 3-6
  Magnetic Stripe Reader . . . 3-8
  Keypad . . . 3-11
  Tone generator . . . 3-13
  Timer . . . 3-14
  System . . . 3-15
  Interface Control . . . 3-18
  Security Function Interface . . . 3-19
  Modified Compiler Run-Time Functions . . . 3-20

Chapter 4. Loading the 4779 Device Resident Application . . . 4-1
Invoking the 4779 Application Download Program for DOS . . . 4-1
Invoking the 4779 Application Download Program for OS/2 . . . 4-1

Appendix A. 4779 Device-Resident Development Kit Components . . . A-1

Appendix B. Additional Security Functions . . . B-1
Construct Triple-Encrypted Block . . . B-1
Format ANSI PIN Block . . . B-2
Format 3624 PIN Block . . . B-2
Read Security Function Device Information . . . B-3

Copyright IBM Corp. 1996, 1997

Appendix C. 4779 Device Resident Application Sample Code List . . . C-1

Figures

1-1. 4779 Operational Overview 1-1
2-1. 4779 Microcode Architecture 2-1
2-2. Memory Map - 4779 Device without Enhanced Security Feature . . 2-10
2-3. Memory Map - 4779 Device with Enhanced Security Feature 2-11
3-1. LCD pattern 3-7
3-2. MIDS tags 3-16
4-1. 4779 Application Download Programs 4-1
A-1. 4779 Device-Resident Development Kit Components A-1

Copyright IBM Corp. 1996, 1997

Notices

References in this publication to IBM products, programs, or services do not imply that IBM intends to make these available in all countries in which IBM operates. Any reference to an IBM product, program, or service is not intended to state or imply that only IBM's product, program, or service may be used. Any functionally equivalent product, program, or service that does not infringe any of IBM's intellectual property rights or other legally protectable rights may be used instead of the IBM product, program, or service. Evaluation and verification of operation in conjunction with other products, programs, or services, except those expressly designated by IBM, are the user's responsibility.

Licensees of this program who wish to have information about it for the purpose of enabling:

(i) the exchange of information between independently created programs and other programs (including this one) and
(ii) the mutual use of the information which has been exchanged, should contact: IBM Corporation, Department MG39/201, 8501 IBM Drive, Charlotte, NC 28262-8563, U.S.A.

Such information may be available, subject to appropriate terms and conditions, including in some cases, payment of a fee.

IBM may have patents or pending patent applications covering subject matter in this document. The furnishing of this document does not give you any license to these patents. You can send license inquiries, in writing, to the IBM Director of Licensing, IBM Corporation, 500 Columbus Avenue, Thornwood, NY, 10594, USA.

Federal Communications Commission (FCC) Statement

Note: This equipment has been tested and found to comply with the limits for a Class A digital device, pursuant to Part 15 of the FCC Rules. These limits are designed to provide reasonable protection against harmful interference when the equipment is operated in a commercial environment. This equipment generates, uses, and can radiate radio frequency energy and, if not installed and used in accordance with the instruction manual, may cause harmful interference to radio communications. Operation of this equipment in a residential area is likely to cause harmful interference, in which case the user will be required to correct the interference at his own expense.

Properly shielded and grounded cables and connectors must be used in order to meet FCC emission limits. IBM is not responsible for any radio or television interference caused by using other than recommended cables and connectors or by unauthorized changes or modifications to this equipment. Unauthorized changes or modifications could void the user's authority to operate the equipment.

This device complies with Part 15 of the FCC Rules. Operation is subject to the following two conditions: (1) this device may not cause harmful interference, and (2) this device must accept any interference received, including interference that may cause undesired operation.

Copyright IBM Corp. 1996, 1997

License

You may use the files which make up Feature Code 3922 (Programs) with the IBM 4779 only in accordance with the IBM International Program License Agreement that accompanies the Programs.

Trademarks

The following terms, denoted by an asterisk (*) in this publication, are trademarks of the IBM Corporation in the United States or other countries or both:

Personal Security	Personal System/2	PS/2
IBM	Operating System/2	OS/2

About This Publication

This book tells you how to create a 4779 device resident application that uses the IBM* 4779 BIOS.

Who Should Read This Book

The information in this book is intended for people who are creating or modifying the 4779 device resident application.

In order to use this document, the reader should be familiar with the system and application programs that work with the 4779 and with background material related to devices of this type. Specifically, the reader should have a working knowledge of the following areas:

- Basic cryptography, including the use of DES and RSA
- Magnetic stripe cards and their use
- Smart cards

In addition the user should be familiar with writing, compiling, linking and debugging the C programming language. A knowledge of the 8051 microprocessor is also valuable.

How This Book Is Organized

This book contains the following sections:

Chapter 3, "4779 Run-Time Library," documents the I/O run-time library support for the 4779 device according to the categories of tasks they perform.

Chapter 2, "Creating a 4779 Device Resident Application," explains how to create a device resident application for the 4779 device.

Appendix A, "4779 Device-Resident Development Kit Components," lists the components of the 4779 Device Resident Development Kit found on the 4779 Device Resident Development Kit diskette.

Appendix B, "Additional Security Functions," describes the additional security functions available to those creating 4779 Device Resident Application.

Appendix C, "4779 Device Resident Application Sample Code List," lists the device resident sample code files available from the self extracting file 4779AZIP.EXE.

---

* Trademark of IBM

Copyright IBM Corp. 1996, 1997

Related Publications

You might need additional information from one or more of the following publications:

The *DOS Technical Reference*, for your DOS operating system

4779 Programming Guide, SA34-2360

4700 Finance Communication System: System Summary, GC31-2016

4700 Financial I/O Planning Guide, GC31-3762

References

The following references may be useful to those readers who are not familiar with cryptography, magnetic stripe technology, or PIN pads.

Applied Cryptograhpy by Bruce Schneier, ISBN 0-471-59756-2.

Security for Computer Networks by D.W. Davies and W.L. Price, ISBN 0-471-92137-8.

Cryptography and Data Security by Dorothy Denning, ISBN 0-201-10150-5.

Transaction Security System, Concepts and Programming Guide: Volume I, Access Controls and DES Cryptography, IBM publication number GC31-3937.

Transaction Security System, Concepts and Programming Guide: Volume II, Public-Key Cryptography, IBM publication number GC31-2889.

4777 Magnetic Stripe Unit and 4778 PIN-Pad Magnetic Stripe Reader: DOS Programming Guide, IBM publication number SA34-2206.

ISO 7811-2: Identification cards - Recording technique - Part 2: Magnetic stripe..

ISO 7816-1: Identification cards - Integrated circuit cards with contacts - Part 1: Physical characteristics.

ISO 7816-2: Identification cards - Integrated circuit cards with contacts - Part 2: Dimensions and location of the contacts.

ISO 7816-3: Identification cards - Integrated circuit cards with contacts - Part 3: Electronic signals and transmission protocols.

Chapter 1. Overview of Programming a 4779 Device Resident Application

The 4779 has two local applications controlling the interface with the end user. The 4779 PC application and required programming is described in the *IBM 4779 Programming Guide*. The 4779 application, a device resident program, is developed as shown in Figure 1-1. A programmer uses the programs supplied with the IBM 4779 Device Resident Application Software Development Kit (SDK) and an 8051C compiler to produce an application that is loaded into the 4779.

Creating a 4779 device resident application requires an understanding of the application requirements and the capabilities of the existing 4779 device resident application. An important design consideration is determining which application, PC, host, or device, performs specific functions.

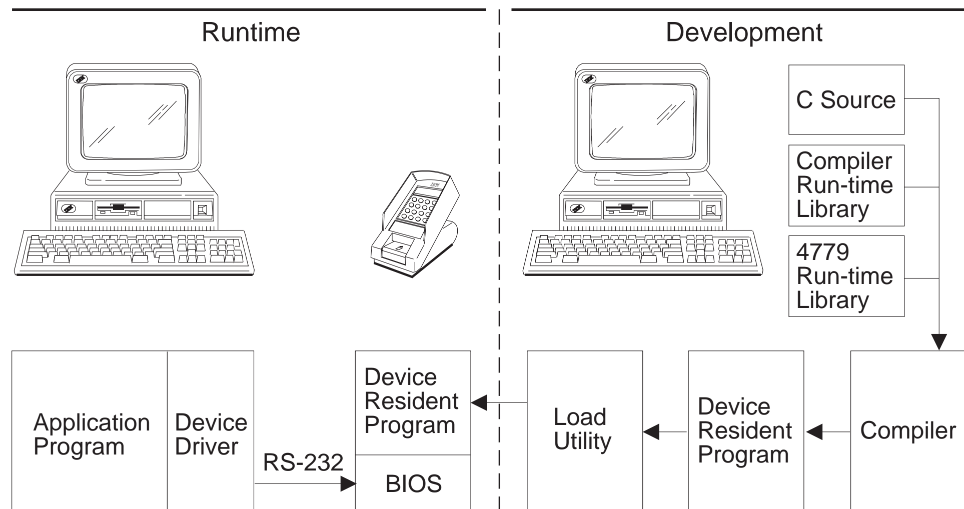


Figure 1-1. 4779 Operational Overview

When more function is performed at the device level, less communication with the PC is required. You should decide if this is important to your application, consider the following:

- You can simplify your PC application by performing primitive operations, such as retries and messages, at the device level.
- If your PC application performs all of the transaction flow control then less function is performed at the device level.

When you create a unique device resident application or modify an existing one then the 4779 Device Resident Application Software Development Kit libraries are required. The device resident application resides in the 4779's 8051C microprocessor.

To compile the device resident application you must use a compiler designed to produce code for an 8051 microprocessor. The 4779 device resident application was created using the C51 Compiler available from Franklin Software, Inc. Similar products are available from other suppliers. These compilers have specific system requirements. You should use your compiler's documentation to determine the minimum system requirements.

Copyright IBM Corp. 1996, 1997

Chapter 2. Creating a 4779 Device Resident Application

This chapter describes the process for creating a 4779 device resident application. These steps include writing source modules that use the 4779 run-time library functions to perform specific application requirements, compiling the source code, and linking the source objects into an executable program.

To assist you in developing an application, the source code used to create the 4779 device resident application supplied with the 4779 is included in Feature Code 3922 as a working sample. You may use this sample code as is, or modified it to achieve your desired application objectives. You are responsible for ensuring that the device resident application meets your required objectives.

A list of the source code provided is found in Appendix C, "4779 Device Resident Application Sample Code List" on page C-1. The files are made available by executing the self-extracting zip file 4779AZIP.EXE. located on the diskette for Feature Code 3922.

The architecture for the 4779 is illustrated in Figure 2-1. While this diagram does not represent all microcode components in the 4779 device, it illustrates what functions are available and how they interact with each other and the rest of the system.

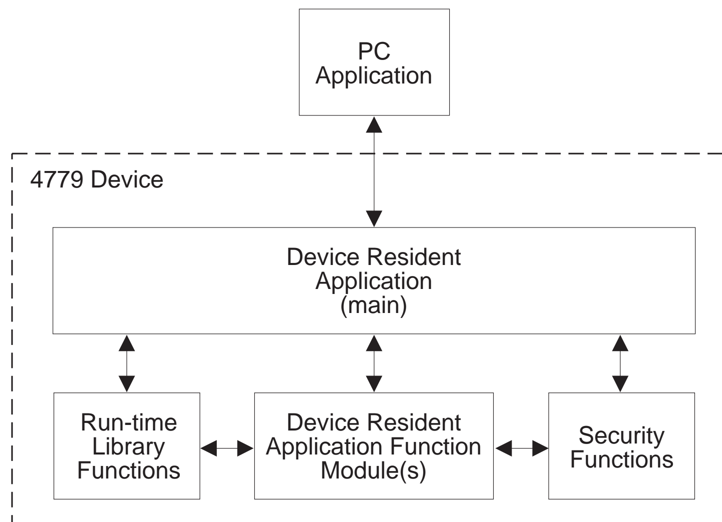


Figure 2-1. 4779 Microcode Architecture

Commands generated by the PC application are sent to the 4779 device via the DOS or OS/2 device driver and placed in the communication buffer. The main program module reads a message in the buffer, interprets the command, calls a function to execute the command, and places the response in a communication buffer. The response is returned to the PC application by the device driver. A number of function modules would typically be required for an entire application. The sample run-time application supplied by IBM is comprised of several modules; a device resident application (main) and subfunction modules. These modules are compiled and linked with the run time library functions described in Chapter 3, "4779 Run-Time Library" to produce a device resident application.

Copyright IBM Corp. 1996, 1997

The modules can also call the security functions through a security function interface, see “Security Function Interface” on page 3-19

Writing the Main Program Module

A basic application must open the serial communication port, perform initialization, read from the communication buffer and write to the communication buffer. See Appendix C, “4779 Device Resident Application Sample Code List” on page C-1 for a complete listing of the main program module. The application program entry point (main) should provide these functions as well as a command interpreter to process the commands sent from the PC application.

Entry Code

The entry code defines required variables and functions for the remainder of the program.

```
/                                                                        /
/                    Main Program Example                        /
/                                                                        /

#define INPUT_MSG_MAX_SIZE 2                  /  Input data buffer size  /
#define OUTPUT_MSG_MAX_SIZE 2                 /  Output data buffer size  /
#define FCN_SUCCESS           /  Run-time library function success value  /

typedef struct            /  Input buffer - messages received from host  /
{
  UINT  length;                   /  # bytes of data received        /
  UCHAR cmd;                      /  Requested appl. command         /
  UCHAR mdata[INPUT_MSG_MAX_SIZE-1];  /  Data for the command            /
} msg_in_struct;

typedef struct                /  Output buffer - messages going to host  /
{
  UINT length;                    /  # bytes of data being sent       /
  UCHAR rc;                       /  Return code                   /
  UCHAR mdata[OUTPUT_MSG_MAX_SIZE-1]; /  Response data                 /
} msg_out_struct;
```

Establishing Serial Communications

Open the serial communication port, this operation uses the run-time function sio_open to specify the baud rate, the word length in bits and the parity.

sio_open

void sio_open (int *baud***, int** *wordlen***, int** *parity***);**

Include	EFT.H
baud	Baud rate to be set (0 = 1200, 1 = 2400, 2 = 4800, 3 = 9600)
wordlen	Word length (0 = 7 bits, 1 = 8 bits, all others reserved)
parity	Parity select (0 = no parity, 1 = odd parity, 2 = even parity, all others reserved)
Returns	No return value

Opens and initializes the serial communication channel.

```
/                                                                        /
/                    Open the Serial Port                        /
/                                                                        /
/  Open the serial communications channel to the host.                 /

 sio_open(BAUD_96   ,             /  at 96   baud                   /
        WORD_LENGTH_8,          /  with 8-bit word length            /
```

```
                PARITY_ODD);          /  and odd parity.                /
```

Application Initialization

Initialize the application, this step is application dependent. It can include such items as variable initialization, writing a start-up message to the display, and clearing the device of any cards.

Communications Buffer Read and Message Processing

Read the communication buffer, this process involves using the run-time function sio_get_message to read the PC application message placed in the communication buffer by the device driver.

sio_get_message

int sio_get_message (unsigned char ****buffer*, unsigned int ****len***);**

Include	EFT.H
buffer	User buffer where a message is to be placed
len	Length of the message buffer
Returns	0 if successful; *len* is updated with the actual number of bytes read if successful.
	-1 if not successful

Reads a single message from the communication receive buffer and stores it in *buffer* as per the link level protocol.

Prior to this call, the message length should be initialized to the maximum length. Messages read from the communication buffer contain an initial two byte length field that is created by the device driver.

If the communication buffer cannot be read, the buffer is cleared with the run-time function sio_rx_flush and the message length and return code are updated to reflect an error condition.

sio_rx_flush

void sio_rx_flush (void);

Include	EFT.H
Returns	No return value

Flushes the receive buffer of the serial communication channel.

```
/                                                                      /
/                   Read the Communication Buffer                 /
/                                                                      /
msg_in.length = INPUT_MSG_MAX_SIZE;        /  Indicate recv. buffer size   /

if (FCN_SUCCESS != sio_get_message(&msg_in.cmd,     /  Buffer addr.        /
                   (UCHAR  ) &msg_in.length)) /  #bytes recd. /
{
  /  The message was not read into our buffer successfully.  Flush it      /
  /  from the buffer and send an errror response message.               /

  (void) sio_rx_flush();       /  Flush the message                    /

  msg_out.rc = arc_DATA_LENGTH_ERROR;
```

```
  msg_out.length =  ;
}
else                           /  Message was received successfully   /
{
  /  A message is available, and we've read it into the buffer           /
  /  msg_in.  Decide what the message is, and call the appropriate       /
  /  processing function.  Pass pointers to the incoming and outgoing     /
  /  message buffers.  The called functions are responsible for          /
  /  setting up a response message in msg_out, including the data (if     /
  /  any), the return code, and the length -- which is the length of      /
  /  only the data, not including the return code.                      /

  /  Note that all commands with codes in the range 8 -EF are for the     /
  /  security function.  These are passed through to that processor      /
  /  for handling, and the responses are passed back to the requester.    /

  /  Set the default response message return code to indicate there      /
  /  were no errors.  Set the default output length to indicate the       /
  /  command had no data to return.  Note that the length is             /
  /  incremented after the command is complete in order to add in the     /
  /  length of the return code - until then, the length is considered     /
  /  to be just the number of bytes of data generated by the command      /
  /  itself.                                                       /

  msg_out.rc = arc_OK;           /  No errors                        /
  msg_out.length =  ;            /  No data                          /
  /                                                                     /
  /                   Interpret Messages Received                  /
  /                                                                     /
  /  Decode the incoming command code.  If it is for the security        /
  /  processor, pass it on to that processor for handling.  Otherwise,    /
  /  decode here and call the appropriate routine.                      /

  /  Check if command is a security function                          /

  if ((msg_in.cmd >= ac_FIRST_SP_CMD) && (msg_in.cmd <= ac_LAST_SP_CMD))
  {

    /  Send request on to the security processor for handling           /

    /  Decrease the response length by one, so it is only the length     /
    /  of the data returned by the command.  The supervisor will add     /
    /  one (see below) to add in the length of the return code.         /

    msg_out.length--;
  }
  else switch (msg_in.cmd)       /  Call required command             /
  {

    case COMMAND_ONE              :
         call_command_one(&msg_in, &msg_out);
         break;
    default:
         unknown_command(&msg_in, &msg_out);
       break;
  }                        /  End switch                         /
}
```

Interpret messages received, this process involves matching the command in a message with a the predefined command set for the application. The following hex command identifiers are reserved : F0, F1, F2, F3, F4, and F5. Commands that are not identifiable as application commands or security function commands are handled as a default. Valid commands initiate a call to an application module that executes the desired function. The command interpreter is a case statement enclosed in a do-while loop that continues until the device is reset.
H3.Communications Buffer Write

Write to the communication buffer, the run-time function sio_put_message is used to return a message to the PC application.

sio_put_message

int sio_put_message (unsigned char **buffer*, **unsigned int** *len***);**

Include	EFT.H
buffer	User message buffer containing the data to be transmitted
len	Length of the data to be transmitted
Returns	0 if successful
	-1 if not successful

Writes *len* bytes from *buffer* to the serial communication channel as per the link level protocol.

In order to maintain synchronization, each message read from the communication buffer must be followed by a return message. Prior to sending a message, the length of the message is incremented to reflect the return code size. If an attempt to write a message to the buffer is unsuccessful, you should re-initialize the device with the reset run-time function.

```
/                                                                              /
/                    Write to the Communication Buffer                 /
/                                                                              /
/  We have finished processing the command, and a response has been        /
/  formatted in the output buffer, msg_out.  Send the response            /
/  message to the PC application.  Note that we increase msg_out.length,    /
/  to account for the return code byte. The application functions only to    /
/  set msg_out.length to the length of the returned data, not including     /
/  the return code.                                                   /

/  If we are unable to communicate with the host, reset the box.           /

msg_out.length += sizeof(msg_out.rc);      /  Add length of return code    /

if (FCN_SUCCESS != sio_put_message(&msg_out.rc,    /  Data to send        /
                             msg_out.length))       /  # bytes      /
{
   reset();                     /  Can't communicate!  Reset.          /
}
```

Main Program Considerations

An application in this context is a set of high-level functions. Each function is a module called by a command interpreter using a predefined command set. Each module typically contains a number of run-time functions combined to achieve a specific result. Your modules should include the following:

- Message verification - This involves verification that the received message length is correct for the particular command sent and that any parameters are valid. Since the device drivers for this device function primarily to pass messages, it is the responsibility of the application program to verify the content of the messages received and sent.
- Device compatibility - This step verifies that the particular 4779 device model or type will support the command being executed. The device information is obtained by a run-time function call to MIDS.

  For more detailed information please refer to the Machine Information Data Structure Appendix of the 4779 Hybrid Smart Card Device Programming Guide.
- Issue run-time functions - This involves execution of the run-time functions required to perform the expected module function.

Error response - This step involves interpreting the result of a call to a run-time function.

Message construction - This step constructs a message to be returned to the PC application. The minimum message contains a return code and message length. Data is also be returned if applicable.

Accessing the Security Functions

Access to the security functions described in the 4779 Hybrid Smart Card Device Programming Guide is through the 4779 device resident application program. Because the security functions cannot be modifiable, a predefined interface must be adhered to when these functions are called. Four additional security function definitions are available when creating a modified 4779 device resident application.

If your application uses any of the security functions, the following definitions must be included in the application program.

	Command Name	Command Code
#define	ac_FIRST_SP_CMD	0x80
#define	ac_READ_SER_NO	0x80
#define	ac_GENERATE_RAND_NO	0x81
#define	ac_LOAD_CLEAR_RSA_PRIV_KEY	0x82
#define	ac_LOAD_DES_KEK_PART	0x83
#define	ac_LOAD_RSA_ENCR_KEK	0x84
#define	ac_LOAD_DES_ENCR_KEK	0x85
#define	ac_LOAD_DES_ENCR_KEK_WITH_CV	0x86
#define	ac_LOAD_DES_ENCR_KD	0x87
#define	ac_VISA	0x88
#define	ac_GENERATE_MAC	0x8D
#define	ac_VERIFY_MAC	0x8E
#define	ac_INITIALIZE_SECPROC	0x8F
#define	ac_FORMAT_PIN_ANSI	0x90
#define	ac_FORMAT_PIN_3624	0x91
#define	ac_RD_SP_DEV_INFO	0x93
#define	ac_COMPUTE_VERIF_PATTERN	0x94
#define	ac_VERIFY_DES_KEY	0x95
#define	ac_LAST_SP_CMD	0xEF

Security functions are accessed by sending a message containing the desired command information to the security functions, waiting for the command to be processed, and then receiving the output message.

This process is outlined below and followed by a code segment example.

Send a message to the security functions

This is accomplished by implementing the run-time function spc_put_message.

spc_put_message

int spc_put_message(unsigned char **buffer*, **unsigned int** *length***);**

Include EFT.H

buffer The user buffer containing the data to be transmitted.

length The length of the user buffer containing the data to be transmitted.

Returns

0 = successful
-1 = not successful

This function transmits a message to the security interface.

The origin of a message sent may be the PC application or the device resident application. This run-time function must be used to send a message to the security functions.

```
/
/                    Send the Security Message                          /
/
/  The incoming data in msg_in contains length, command, and data fields,
/  all in contiguous memory.  From the message's length field, we compute
/  the length of the message to be passed to the security function, and
/  we then pass the entire cmd/length/data structure with this associated
/  length using the run-time function spc_put_message.

ok = (FCN_SUCCESS == spc_put_message((byte  ) &msg_in->cmd, msg_in->lengt
```

Initialize return code

Involves initializing the return code of a message received from the security functions to non-error state.

Initialize message return length

The length of the message returned must be initialized to 1999 in order to insure compatibility with the non-modifiable security code.

Receive a message from the security functions

This is done by use of the run-time function spc_get_message.

spc_get_message

int spc_get_message (unsigned char **buffer*, **unsigned int** **length***);**

Include EFT.H

buffer The user buffer for the returned security function response

length On input, the length of *buffer*; on output, the length of the returned security function response.

Returns 0 = successful
-1 = not successful

This function reads a response from the security interface. If no response is available, this function returns immediately with an unsuccessful return code.

This function is called within a do-while loop that either reads the return message when it was available or times out after some defined amount of time.

```
/                                                                           /
/                    Receive the Security Message                      /
/                                                                           /
/  Keep trying to get a response, until we're successful, or until      /
/  we have tried the maximum number of times.                          /

do
{

   ok = (FCN_SUCCESS == spc_get_message((byte  ) &msg_out->rc,
                               (int_or_bytes  ) &msg_out->length));
   if (!ok) timeout_count--;

} while ((!ok) && (timeout_count !=  ));
```

Security Function Communication Example

```
/                                                                           /
/            Security Function Communication Example                  /
/                                                                           /
#define INPUT_MSG_MAX_SIZE 2                 /  Input data buffer size  /
#define OUTPUT_MSG_MAX_SIZE 2                /  Output data buffer size  /
#define FCN_SUCCESS            /  Run-time library function success value  /

typedef struct            /  Input buffer - messages received from host  /
{
  UINT  length;                   /  # bytes of data received        /
  UCHAR cmd;                      /  Requested appl. command         /
  UCHAR mdata[INPUT_MSG_MAX_SIZE-1];  /  Data for the command            /
} msg_in_struct;

typedef struct                /  Output buffer - messages going to host /
{
  UINT length;                    /  # bytes of data being sent       /
  UCHAR rc;                       /  Return code                  /
  UCHAR mdata[OUTPUT_MSG_MAX_SIZE-1]; /  Response data                 /
} msg_out_struct;

int ok;                              /  Communication return value  /
UNIT timeout_count;    /  Number of retries waiting for security response  /
/                                                                           /
/                  Send the Security Message                       /
/                                                                           /
/  The incoming data in msg_in contains length, command, and data fields,   /
/  all in contiguous memory.  From the message's length field, we compute   /
/  the length of the message to be passed to the security function, and    /
/  we then pass the entire cmd/length/data structure with this associated   /
/  length using the run-time function spc_put_message.              /

ok = (FCN_SUCCESS == spc_put_message((byte  ) &msg_in->cmd, msg_in->length));
/                                                                           /
/                  Initialize the Return Code                      /
/                                                                           /
/  Initialize the return code in the response message to zero, indicating   /
/  that no errors were detected.  It will be replaced if errors occur.     /

msg_out->rc = arc_OK;

/  We now call spc_get_message to wait for a response from the security    /
/  function.  If the security function has not yet responded, the         /
/  request will time out and return with an error.  In this case, we keep   /
/  retrying up to the maximum number of times for this command.  That      /
/  maximum is obtained with a call to get_sp_cmd_timeout().             /

if (ok)                         /  If no errors sending the message...  /
{

  timeout_count =                /  Number defined by the application  /

  /                                                                         /
  /                  Initialize the Return Length                 /
```

```
/                                                                        /
/  Initialize msg_out length to the maximum to insure compatibility     /
/  with the security functions.                                   /

msg_out->length =  OUTPUT_MSG_MAX_SIZE - 1;

/                                                                        /
/                   Receive the Security Message                  /
/                                                                        /
/  Keep trying to get a response, until we're successful, or until      /
/  we have tried the maximum number of times.                      /

do
{

  ok = (FCN_SUCCESS == spc_get_message((byte  ) &msg_out->rc,
                               (int_or_bytes  ) &msg_out->length));
  if (!ok) timeout_count--;

} while ((!ok) && (timeout_count !=  ));
}
/  If there were errors, set the length of the response to zero, and put    /
/  an appropriate error code in the response message.                /

if (!ok)
{
  msg_out->rc = arc_SECURITY_ERROR;
  msg_out->length = 1;
}
```

Compiling the Device Resident Application

Creating object code for the application is similar to using a conventional 'C' compiler. The application must be compiled using the compiler options specifing a large memory model without debug, and optimization based upon size. Refer to the documentation provided with the compiler you are using for how to implement these options.

Linking the Device Resident Application

The link step will combine the compiled object modules into an executable program that is in Intel OMF-51 absolute object module format. The Enhanced Security Feature will determine the specifics of the link process in terms of the Run-Time Libraries called and the code location. 4779 devices without this feature will call the Run-Time Libraries 4779RTL.LIB and 4779RTLS.LIB and will link the code at a location of 7000H. Devices with this feature will call the Run-Time Libraries 4779RTL.LIB and 4779RTLD.LIB and will link the code at 4000H. Please refer to the memory maps in figures 2-1 and 2-2. Bit addressing begins at 40H, the stack address is 30H, and the external data address begins at 08000H. To link start-up code refer to the documentation provided with the compiler you are using.

In order to load the linked executable into the 4779 device, convert it into an Intel HEX format with a HEX file extension. Refer to the documentation provided with the compiler that you are using for a description of how to accomplish this.

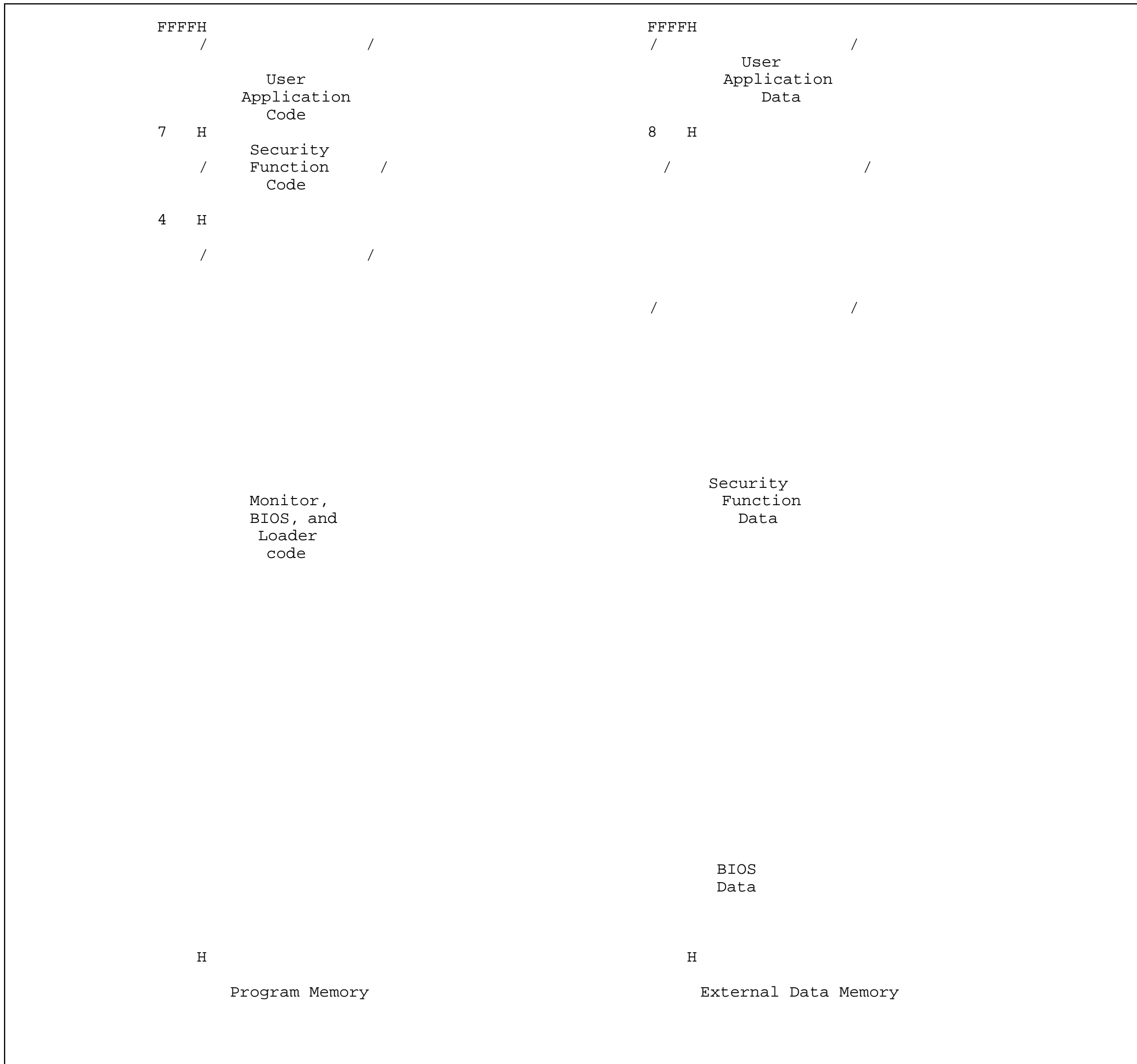


Figure 2-2. Memory Map - 4779 Device without Enhanced Security Feature

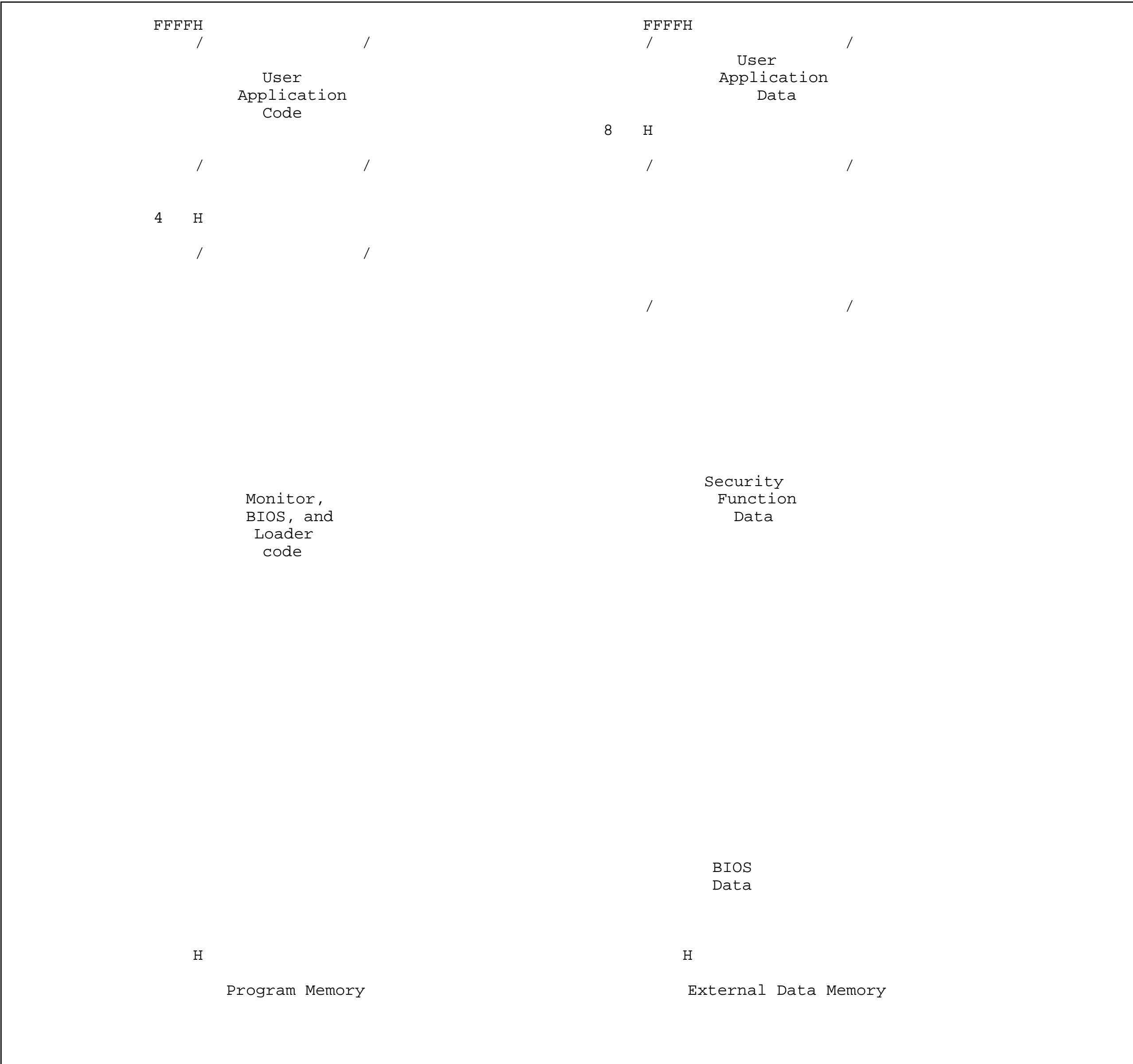


Figure 2-3. Memory Map - 4779 Device with Enhanced Security Feature

Chapter 3. 4779 Run-Time Library

This chapter describes the I/O run-time library support for the 4779 device. The three libraries provided with this feature are:

4779RTL.LIB
4779RTLS.LIB
4779RTLD.LIB

These libraries support 4779 devices with or without the Enhanced Security Feature (Feature Codes 3923, 3924). Each type of device requires calling two of the above libraries when the application is linked.

The Run-Time Library **4779RTL.LIB** contains function categories that are common to either type of device. These categories are listed below.

Serial communications
Integrated circuit card (ICC) reader
Display
Magnetic stripe reader
Keypad
Tone generator
Timer
System
Interface control

In addition, the compiler supplied run-time library may require modification to include functions used with the device resident application. If your application uses any of the functions in this category, refer to the documentation supplied with the compiler related to library utilities. The modified library must be compatible with your compiler's large memory model. This library is used for either type of 4779 device. Refer to the following category for specific functions.

Modified compiler run-time functions

How To Use the Libraries

When you create or modify a 4779 resident application the Run-Time Libraries previously described are statically linked with the application object modules. The libraries you link with the compiler are determined by the run-time functions called by the application and the presence or absence of the Enhanced Security Feature. For information on linking the libraries refer to the documentation provided with the compiler that you are using and Chapter 2, “Creating a 4779 Device Resident Application” on page 2-1.

Interface

Copyright IBM Corp. 1996, 1997

Serial Communications

sio_open

void sio_open (int *baud*, **int** *wordlen*, **int** *parity***);**

Include	EFT.H
baud	Baud rate to be set (0 = 1200, 1 = 2400, 2 = 4800, 3 = 9600)
wordlen	Word length (0 = 7 bits, 1 = 8 bits, all others reserved)
parity	Parity select (0 = no parity, 1 = odd parity, 2 = even parity, all others reserved)
Returns	No return value

Opens and initializes the serial communication channel.

sio_rx_flush

void sio_rx_flush (void);

Include	EFT.H
Returns	No return value

Flushes the receive buffer of the serial communication channel.

sio_in

int sio_in (void);

Include	EFT.H
Returns	A single data byte from the communication receive buffer
	-1 if no data available

Reads and returns a data byte from the serial communication channel receive buffer.

sio_out

int sio_out (unsigned char *data***);**

Include	EFT.H
data	Data byte to be transmitted
Returns	0 if successful
	-1 if not successful

Transmits a data byte across the serial communication channel.

sio_get_message

int sio_get_message (unsigned char ***buffer*, unsigned int ***len***);**

Include	EFT.H
buffer	User buffer where a message is to be placed
len	Length of the message buffer
Returns	0 if successful; *len* is updated with the actual number of bytes read if successful.
	-1 if not successful

Reads a single message from the communication receive buffer and stores it in *buffer* as per the link level protocol.

sio_put_message

int sio_put_message (unsigned char ***buffer*, unsigned int** *len***);**

Include	EFT.H
buffer	User message buffer containing the data to be transmitted
len	Length of the data to be transmitted
Returns	0 if successful
	-1 if not successful

Writes *len* bytes from *buffer* to the serial communication channel as per the link level protocol.

sio_in_status

int sio_in_status (void);

Include	EFT.H
Returns	0 if data available
	-1 if no data available

Returns the status of the serial communication receive channel.

ICC (Integrated Circuit Card) Reader

icc_arm

void icc_arm (int *switch***);**

Include	EFT.H
switch	control switch 0 = off 1 = on
Returns	No return value

Arms or disarms the ICC (Integrated Circuit Card) reader.

Note: This function enables the model 2 transport motor. Before issuing this command to the model 2, make sure a card is not partially inserted into the reader. When using a model 1, it is recommended that prior to issuing this function call it is confirmed that a card is present in the device. This would be done by reading status.

icc_reset

int icc_reset (unsigned char ***buffer*);**

Include	EFT.H
buffer	User buffer where the answer to reset data is to be stored (the length of the reset data is self-defining)
Returns	0 = successful -1 = not successful

Resets the ICC (Integrated Circuit Card).

icc_cmd0

int icc_cmd0 (unsigned char ***buffer, int** *length***, unsigned int ***icrc***);**

Include	EFT.H
buffer	User buffer that on input contains the command to be sent to the ICC and on output where the ICC response data is to be stored (if required)
length	Length of the user command buffer to be sent to the ICC
icrc	User buffer where the return code from the ICC is to be stored
Returns	0 = successful -1 = not successful

Sends a command and receives a response from the ICC using protocol T=0.

icc_cmd

int icc_cmd (unsigned char **cmdbuff*, **int** *cmdlen*, **unsigned char** **rspbuff*, **unsigned char** **rsplen*, **unsigned int** **icrc*);

Include EFT.H

cmdbuff User buffer containing the command to be sent to the ICC

For a T=0 card, this buffer contains the command data
For a T=1 card, this buffer contains the information field

cmdlen Length of the user command buffer to be sent to the ICC

rspbuff User buffer where the ICC response data is to be stored

rsplen Length of the response received from the ICC

icrc User buffer where the return code from the ICC is to be stored

Returns

0 = successful
-1 = not successful

Sends a command and receives a response from the ICC using either protocol T=0 or T=1, depending on the type of card present in the reader.

icc_status

int icc_status (void);

Include EFT.H

Returns

0 = no ICC present
1 = ICC present
3 = ICC present and locked
-1 = card motor timeout

Senses the presence of a card in the reader.

Display

cld

void cld (void);

Include EFT.H

Returns No return value

Clears the display and moves the cursor to the home position.

cursor_ctrl

void cursor_ctrl (int *mode***);**

Include EFT.H

switch mode

0 = cursor off/no blink
1 = cursor on/no blink
2 = cursor off/blink
3 = cursor on/blink

Returns No return value

Enables or disables display and blink attribute of the display cursor.

gotoxy

void gotoxy (int *x***, int** *y***);**

Include EFT.H

x x coordinate (0 - 19, all others reserved)

y y coordinate (0 - 3, all others reserved)

Returns No return value

Moves the cursor to the designated position.

void lcd_pattern (unsigned char *cp***, unsigned char** **buffer***);**

Include	EFT.H
cp	character codepoint to be loaded 0 = character 1 1 = character 2 2 = character 3 3 = character 4 4 = character 5 5 = character 6 6 = character 7 7 = character 8
buffer	User buffer containing character defined data to be loaded to the LCD character generator
Returns	No return value

Loads a user defined character to one of the 8 user definable codepoints in the LCD character generator.

The LCD is a 5x8 matrix with cursor. The following example shows the format of the user buffer to define an "up arrow".

Buffer Index	b7	b6	b5	b4	b3	b2	b1	b0
0	X	X	X	0	0	1	0	0
1	X	X	X	0	1	1	1	0
2	X	X	X	1	0	1	0	1
3	X	X	X	0	0	1	0	0
4	X	X	X	0	0	1	0	0
5	X	X	X	0	0	1	0	0
6	X	X	X	0	0	1	0	0
7	X	X	X	0	0	0	0	0

where ***X*** is ignored, ***0*** is a dot that is to be off, and *1* is a dot that is to be on.

Figure 3-1. LCD pattern

Magnetic Stripe Reader

msr_arm

void msr_arm (int *switch***);**

Include EFT.H

switch Enable/disable switch

0 = disable
1 = enable for read
2 = enable for encode

Returns No return value

Arms/disarms the magnetic stripe reader. All 3 tracks are armed if arm for read is requested.

Notes:

1. Only the model 2 supports encode.
2. This function enables the transport motor on the model 2. Before issuing this command to the model 2, make sure a card is not partially inserted into the reader. Refer to the Run-Time Library function msr_status to obtain this information.
3. The arm for read function causes a read operation to occur in the model 2 if a card is present and locked in the reader. Refer to the Run-Time Library function msr_status to obtain this information.
4. When using a model 1, it is recommended that prior to issuing this function call it is confirmed that a card is present in the device or a card is present and locked. Refer to the Run-Time Library function msr_status to obtain this information.

msr_read_data

int msr_read_data (int *trk***, char** **buffer***);**

Include EFT.H

trk track data to be read

0 = track 1
1 = track 2
2 = track 3
all others reserved

buffer Storage location for the returned track data

Returns

0 = successful
-1 = not successful

Reads magnetic data from *trk* and stores them in *buffer*. SOM, EOM, and LRC are validated but not returned as part of the data; the data is converted to ASCII and is a null terminated string. This function returns to the caller immediately if no data is available.

msr_read

int msr_read (int *trk*, **char** **buffer*);

Include EFT.H

trk track data to be read

0 = track 1
1 = track 2
2 = track 3
all others reserved

buffer Storage location for the returned track data

Returns

0 = successful
-1 = not successful

Waits for a magnetic card to be inserted and places the magnetic data from *trk* into *buffer* (if a successful read). The SOM, EOM, and LRC are validated but not returned as part of the data; the data is converted to ASCII and is a null terminated string. If no data is available this function waits for a card to be inserted; if data is available in the buffer, this function returns that data immediately.

Note: This function returns an error if the read operation is unsuccessful.

msr_status

int msr_status (void);

Include EFT.H

Returns

0 = MSR card not present in device
1 = card is present
3 = card is present and locked
-1 = card motor timeout

Queries the MSR system for the presence of a magnetic stripe card in the reader.

msr_eject

void msr_eject (void);

Include EFT.H

Returns No return value

Disarms the magnetic circuitry and ejects a card from the reader.

int msr_write (int *trk*, **char ****buffer***);**

Include	EFT.H
trk	track data to be encoded (1 = track 2, all others reserved)
buffer	Storage location containing the data to be encoded to *trk*.
Returns	

0 = successful
-1 = unsuccessful
-2 = card motor timeout

Encodes *buffer* to *trk*. The encode data on input must be a null terminated ASCII string. A readback verification is performed for this operation.

Notes:

1. Only the model 2 supports encode operations.
2. The model 2 only supports track 2 encode operations.
3. This function enables the transport motor on the model 2.

Keypad

clr_key_buf

void clr_key_buf (void);

Include EFT.H

Returns No return value

Clears the keypad input buffer.

read_key

int read_key (void);

Include EFT.H

Returns Scan/ASCII code returned (high byte = scan, low = ASCII)
-1 if no key is available

Reads and converts a keystroke from the keypad buffer into a scan and ASCII character code.

Keycap	Scan Code	ASCII Code
0	70H	30H
1	71H	31H
2	72H	32H
3	73H	33H
4	74H	34H
5	75H	35H
6	76H	36H
7	77H	37H
8	78H	38H
9	79H	39H
*	7AH	08H
#	7BH	0DH
F1	7CH	41H
F2	7DH	42H
F3	7EH	43H
F4	7FH	44H

peek_key

int peek_key (void);

Include EFT.H

Returns Scan/ASCII code returned (high byte = scan, low = ASCII)
-1 if no key is available

This function performs a nondestructive read and conversion of a keystroke from the keypad buffer into a scan and ASCII character code - the keystroke is not removed from the keypad buffer.

key_tone_ctrl

void key_tone_ctrl (int *switch***);**

Include	EFT.H
switch	Enable/disable keypad tone feedback 0 = disable 1 = enable
Returns	No return value

This function enables/disables audio feedback on keypad data entry.

Note: If an application wants visual feedback, it must be provided by the application.

Tone generator

beep

void beep (unsigned int *len***);**

Include	EFT.H
len	Length of time (in 10 millisecond intervals) for tone (0 - 65535)
Returns	No return value

Generates a tone for *len* * 10 milliseconds.

Timer

timer_set

void timer_set (unsigned int *len***);**

Include	EFT.H
len	Length of time (in 10 millisecond intervals) to be set (0 - 65535)
Returns	No return value

Sets the timer value *len* * 10 milliseconds and enables the timer function (i.e., this function does not wait for the timeout to occur and returns to the caller immediately). The timeout condition can be checked using ***timer_check***.

timer_check

int timer_check (void);

Include	EFT.H
Returns	0 = no timeout -1 = timeout

Returns the status of the current timer function.

timer_wait

void timer_wait (unsigned int *len***);**

Include	EFT.H
len	Length of time to wait (in 10 millisecond intervals)
Returns	No return value

Waits *len* * 10 milliseconds before returning to application.

System

reset

void reset (void);

Include EFT.H

Returns No return value

This function performs a long jump to the POR vector and will cause a reinitialization of the system.

warm_reset

void warm_reset (void);

Include EFT.H

Returns No return value

This function performs a long jump to the user application (*main*).

download

void download (void);

Include EFT.H

Returns No return value

This function performs a long jump to the system's bootstrap loader. This function allows an application to dynamically initiate a program download to the device. Once the device application has issued this function, the host must send the application program across the communication interface. Following the application download, the system will begin executing from the POR vector.

Machine Information Data Structure (MIDS)

unsigned char MIDS (unsigned char *index*, **unsigned char** **buffer***);**

Include — EFT.H

Index — Value from 1 to 9. To retrieve the entire object use 1, for elements of the object use 2 through 9.

buffer — The retrieved data size will not exceed X'80'(128) bytes.

Returns — Number of bytes retrieved from the MIDS object

Zero when there was a error, such as invalid *index*

This function will retrieve information concerning the capabilities, features and model number of the device.

The machine information is separated into functional categories, where each category is given a separate tag. The categories are as follows.

Figure 3-2. MIDS tags

Tag value	Category
X'8001'	This is the tag for the constructed MIDS object, which contains all the other objects in its data field.
X'0002'	General device information which does not fall into the other categories.
X'0003'	Magnetic stripe read and write information.
X'0004'	Keypad information
X'0005'	Smart card reader information.
X'0006'	Display information.
X'0007'	Communications interface information.
X'0008'	Information about the version of BIOS resident in the device.
X'0009'	Information about the model type of 4779.

Notice that the constructed tag X'8001' has the high order bit set to 1, while all the simple objects carry tags with this bit set to 0. The high order bit is used to distinguish simple and constructed tags, allowing the parsing software to determine whether to expect the data field to contain raw data or a series of Tag-Length-Value objects.

For more detailed information please refer to the *4779 Hybrid Smart Card Device Programming Guide*.

CRC

int CRC (int *address*, **int** *length***);**

Include	EFT.H
address	The starting memory address in hexadecimal. The CRC will be evaluated beginning with the byte in this memory address.
length	The length of memory in hexadecimal over which the CRC is to be evaluated. If the low order byte of this parameter is zero, invalid CRC values will be returned.
Returns	The CRC value in hexadecimal.

This function returns a CRC value based on the device memory defined by the starting address and evaluation length.

Interface Control

interface_status

int interface_status (void);

Include EFT.H

Returns Communication interface status

0 = not busy
1 = busy

This function returns the status of the serial communication interface.

interface_request

int interface_request (void);

Include EFT.H

Returns

0 = successful
-1 = not successful

This function requests ownership of the serial communication channel.

interface_release

void interface_release (void);

Include EFT.H

Returns No return value

This function releases ownership of the serial communication channel.

Security Function Interface

spc_put_message

int spc_put_message(unsigned char **buffer*, **unsigned int** *length***);**

Include — EFT.H

buffer — The user buffer containing the data to be transmitted.

length — The length of the user buffer containing the data to be transmitted.

Returns

0 = successful
-1 = not successful

This function transmits a message to the security interface.

spc_get_message

int spc_get_message (unsigned char **buffer*, **unsigned int** **length***);**

Include — EFT.H

buffer — The user buffer for the returned security function response

length — On input, the length of *buffer*; on output, the length of the returned security function response.

Returns — 0 = successful
-1 = not successful

This function reads a response from the security interface. If no response is available, this function returns immediately with an unsuccessful return code.

spc_reset

viod spc_reset (void);

Include — EFT.H

Returns — 0 if successful

This function reinitializes the security function in 4779 devices with the Enhanced Security Feature.

spc_rcv_byte

int spc_rcv_byte (void);

Include — SP_COMM.H

Returns — A single data byte from the security function communication buffer.

This function is used to read a single byte of data from the security function communication buffer. When implemented this function will degrade security function performance and is not recommended for use. It is included as a library function to insure compatibility with early versions of code.

Modified Compiler Run-Time Functions

putchar

char putchar (char *c*);

Include	STDIO.H
c	Character to be written.
Returns	The character written, *c*

Writes a single character to the LCD at the current cursor position.

_getkey

char _getkey (void);

Include	STDIO.H
Returns	An ASCII code from the keypad if successful -1 if not successful

This function returns an ASCII code from the keypad buffer if keypad data is available. If no data is available, this function returns immediately with an unsuccessful return code.

Chapter 4. Loading the 4779 Device Resident Application

Once you have compiled and linked your device resident application program, you will need to load it into the 4779 device. The 4779 Application Download Programs provide this service. The 4779 Application Download Programs reside on the 4779 DOS and OS/2 device drivers diskette and are identified in the following table.

Figure 4-1. 4779 Application Download Programs

Component	Description
4779APD.EXE	4779 Application Download Program for DOS
4779APD2.EXE	4779 Application Download Program for OS/2

Invoking the 4779 Application Download Program for DOS

Before you invoke the 4779 Application Download Program for DOS, copy file *4779apd.exe* from the 4779 DOS and OS/2 Device Driver Diskette to a directory on your PC workstation. In order to execute the 4779 Application Download Program for DOS, the 4779 DOS device driver (4779DOS.SYS) must be installed and loaded in your workstation. To invoke the 4779 Application Download Program for DOS, enter the following on the DOS command line:

```
[drive:path\]4779apd [drive:path\]filename.hex [/c:n or /a:xxxx]

   where
     filename = file name of device application to be downloaded
     n       = serial port to which 4779 is attached,
                syntax:  1 for COM1 (default), or 2 for COM2
     xxxx     = hexidecimal COM port base address
```

Invoking the 4779 Application Download Program for OS/2

Before you invoke the 4779 Application Download Program for OS/2, copy file *4779apd2.exe* from the 4779 DOS and OS/2 Device Driver Diskette to a directory on your PC workstation. In order to execute the 4779 Application Download Program for OS/2, the 4779 OS/2 physical device driver (4779OS2.SYS) and dynamic link library (x4779OS2.DLL) must be installed and loaded in your workstation. To invoke the 4779 Application Download Program for OS/2, enter the following on the OS/2 command line:

```
4779apd2  [drive:path\]filename.hex

  where
     filename = file name of device application to be downloaded
```

Copyright IBM Corp. 1996, 1997

Appendix A. 4779 Device-Resident Development Kit Components

The 4779 Device-Resident Development Kit includes the following components.

Figure A-1. 4779 Device-Resident Development Kit Components

Component	Description
4779RTL.LIB	4779 Runtime Library for models with or without the Enhanced Security Feature
4779RTLS.LIB	4779 Runtime Library for models without the Enhanced Security Feature
4779RTLD.LIB	4779 Runtime Library for models with the Enhanced Security Feature
4779APS.HEX	4779 Default Application for models without the Enhanced Security Feature
4779APD.HEX	4779 Default Application for models with the Enhanced Security Feature
4779AZIP.EXE	4779 Resident Application Program Source

Copyright IBM Corp. 1996, 1997

Appendix B. Additional Security Functions

The following security functions are available for use by the 4779 device resident application program. Unlike the security functions described in chapter 5 of 4779 Hybrid Smart Card Device Programming Guide these may not be called from the PC application.

Construct Triple-Encrypted Block

This command is used to construct and return an 8-byte triple encrypted block of data. This function receives the following information: the Data Key pair to be used, a pad character for the buffer to be triple encrypted in the event that the buffer length is less than 16 bytes, the length of the buffer to be triple encrypted (between 1 and 16 bytes), and the buffer to be triple encrypted.

The format of the command is as follows.

Command - X'88'

Key number pair to be used - X'00' to X'03' defined as follows:

- X'00'
  - Use Data Key 0 as left half of a 16 byte Key for triple encryption
  - Use Data Key 1 as right half of a 16 byte Key for triple encryption
- X'01'
  - Use Data Key 2 as left half of a 16 byte Key for triple encryption
  - Use Data Key 3 as right half of a 16 byte Key for triple encryption
- X'02'
  - Use Data Key 4 as left half of a 16 byte Key for triple encryption
  - Use Data Key 5 as right half of a 16 byte Key for triple encryption
- X'03'
  - Use Data Key 6 as left half of a 16 byte Key for triple encryption
  - Use Data Key 7 as right half of a 16 byte Key for triple encryption

Reserved byte X'00'

Pad character (1 byte)

Buffer length to be triple encrypted

Buffer to be triple encrypted (16 ASCII digits maximum)

The format of the response is as follows.

Expected Return Codes

- X'00' - No Error
- X'02' - Data Length Error
- X'04' - Invalid Value
- X'08' - Bad Key

Data - Encrypted Block - 8 Bytes

Copyright IBM Corp. 1996, 1997

Format ANSI PIN Block

This command is used to construct and return an encrypted PIN block using the ANSI 9.8 format. This function receives the following information: the PAN, generally read from track 2 of the magnetic stripe; the PIN, collected from the keypad; and the ID of the single length DES key to be used to encrypt the PIN block.

The format of the command is as follows.

Command - X'90'

Key Number X'00' to X'07'

Reserved byte X'00'

PIN length (a value between 4 and 12 inclusive)

PAN length

Primary Identification Number (12 ASCII digits maximum)

Primary Account Number (19 ASCII digits maximum)

The format of the response is as follows.

Expected Return Codes

- X'00' - No Error
- X'02' - Data Length Error
- X'04' - Invalid Value
- X'08' - Bad Key

Data - Encrypted PIN Block - 8 Bytes

Format 3624 PIN Block

This command is used to construct and return an encrypted PIN block using the IBM 3624 format. This function receives the following information: the PIN, collected from the keypad; and the ID of the single length DES key to be used to encrypt the PIN block.

The format of the command is as follows.

Command - X'91'

Key Number X'00' to X'07'

Reserved byte X'00'

Pad character (1 byte)

PIN length (a value between 1 and 16 inclusive)

Primary Identification Number (16 ASCII digits maximum)

The format of the response is as follows.

Expected Return Codes

- X'00' - No Error
- X'02' - Data Length Error
- X'04' - Invalid Value

– X'08' - Bad Key

Data - Encrypted PIN Block - 8 Bytes

Read Security Function Device Information

This command returns the device information structure for the security function. This includes the serial number, the microcode version, and the application ID string.

The format of the command is as follows.

Command - X'93'

The format of the response is as follows.

Expected Return Codes

– X'00' - No Error

Serial number - 8 bytes

Microcode version - 14 bytes

Application ID - 16 byte

Appendix C. 4779 Device Resident Application Sample Code List

The following is a list of the sample code provided with the Run-Time Library feature. This is the sample code used to generate the 4779 device resident application that is supplied with the device. A brief description of each module is provided as well as the hex command code implemented by the module when applicable.

Module Name	Command Executed	Description
RD_DVINF.C	00	Obtain basic device information
RD_STAT.C	01	Return card status
WRT_DISP.C	02	Write message to the display
RD_KPD.C	03	Read the keypad
MED_ARM.C	04	Arm the device - msr or icc
EJECT.C	05	Eject the card
RD_MAG.C	06	Read magnetic stripe card
WRT_MAG.C	07	Encode track 2 of magnetic stripe card
SC_XCHNG.C	08	Pass message to or from icc
PIN_GET.C	09, 0B	Format PIN - ANSI or 3624
PIN_GETP.C	12	Format ANSI PIN using parameters
SCPINCHK.C	0A, 30	Check icc password - SAISS or MFC
LOADPARM.C	0D	Load application variables
RD_STRCT.C	0E	Read Machine Information Data Structure
OFFSET.C	0C	Generate offset - 3624
OFFSETP.C	13	Generate offset using parameters - 3624
OFFSETC.C	21	Generate offset comprehensive version
PIN_VER.C	0F	Verify PIN - 3624
PIN_VERP.C	14	Verify PIN using parameters - 3624
PIN_VERC.C	22	Verify PIN comprehensive version
MAN_CARD.C	20	Insert card in device
LDV_PARM.C	10	Load application parameters
UTL_FUNC.C	11	Execute useful application functions
DIAG.C	F1	Application download (Reserved)
ABORTDEV.C	F2	Cancel keypad operation (Reserved)
REST_DEV.C	F3	Reset the device (Reserved)
REST_SPC.C	F4	Reset the security processor (Reserved)
SUPERVSR.C	NA	Application entry point - main
UNKN_CMD.C	NA	Response to unknown command
CHK_MIDS.C	NA	Check compatibility of device
GLBLDATA.C	NA	Global data
SP_CMD.C	NA	Handles messages with security function
ATR.C	NA	Decode answer to reset for icc
DEFCHARS.C	NA	Define special display characters
ICC_RQST.C	NA	Communicate with icc
MSG.C	NA	Initialize display prompt messages
UTILITY.C	NA	Support functions
CMDCODES.H	NA	Command definitions
RETCODES.H	NA	Return code definitions
EFT.H	NA	Run-time library prototypes
VERSION.H	NA	Version information
SP_TYPES.H	NA	Definition file
COMMUNIC.H	NA	Definition file
TYPEDEFS.H	NA	Definition file
GLBLDATA.H	NA	Definition file

Copyright IBM Corp. 1996, 1997

Module Name	Command Executed	Description
RUNTIME.H	NA	Definition file
UTILITY.H	NA	Definition file
CMDPROCS.H	NA	Definition file
DISPLAY.H	NA	Definition file
KEYPAD.H	NA	Definition file
MAGNETIC.H	NA	Definition file
SMT_CARD.H	NA	Definition file
MISCCMDS.H	NA	Definition file
SP_GLBDT.H	NA	Definition file
SP_COMM.H	NA	Definition file
MP_BLOCK.H	NA	Definition file
TIMERS.H	NA	Definition file
DIAG.H	NA	Definition file

Communicating Your Comments to IBM

4779 Hybrid Smart Card Device
Device Resident
Application Programming Guide

Publication No. SA34-2361-01

If you especially like or dislike anything about this book, please use one of the methods listed below to send your comments to IBM. Whichever method you choose, make sure you send your name, address, and telephone number if you would like a reply.

Feel free to comment on specific errors or omissions, accuracy, organization, subject matter, or completeness of this book. However, the comments you send should pertain to only the information in this manual and the way in which the information is presented. To request additional publications, or to ask questions or make comments about the functions of IBM products or systems, you should talk to your IBM representative or to your IBM authorized remarketer.

When you send comments to IBM, you grant IBM a nonexclusive right to use or distribute your comments in any way it believes appropriate without incurring any obligation to you.

If you are mailing a readers' comment form (RCF) from a country other than the United States, you can give the RCF to the local IBM branch office or IBM representative for postage-paid mailing.

- If you prefer to send comments by mail, use the RCF at the back of this book.
- If you prefer to send comments by FAX, use this number:

  United States & Canada: 1-800-955-5259

Make sure to include the following in your note:

- Title and publication number of this book
- Page number or topic to which your comment applies.

Readers' Comments — We'd Like to Hear from You

4779 Hybrid Smart Card Device
Device Resident
Application Programming Guide

Publication No. SA34-2361-01

Overall, how satisfied are you with the information in this book?

	Very Satisfied	Satisfied	Neutral	Dissatisfied	Very Dissatisfied
Overall satisfaction					

How satisfied are you that the information in this book is:

	Very Satisfied	Satisfied	Neutral	Dissatisfied	Very Dissatisfied
Accurate					
Complete					
Easy to find					
Easy to understand					
Well organized					
Applicable to your tasks					

Please tell us how we can improve this book:

Thank you for your responses. May we contact you? Yes No

When you send comments to IBM, you grant IBM a nonexclusive right to use or distribute your comments in any way it believes appropriate without incurring any obligation to you.

Name

Address

Company or Organization

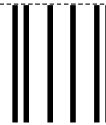
Phone No.

Readers' Comments — We'd Like to Hear from You
SA34-2361-01

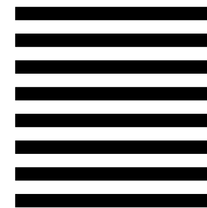
IBM

Cut or Fold
Along Line

Fold and Tape | **Please do not staple** | Fold and Tape



NO POSTAGE
NECESSARY
IF MAILED IN THE
UNITED STATES



BUSINESS REPLY MAIL

FIRST-CLASS MAIL PERMIT NO. 40 ARMONK, NEW YORK

POSTAGE WILL BE PAID BY ADDRESSEE

International Business Machines Corporation
RDS Solutions Development
Department 56I
8501 IBM Drive
Charlotte NC 28262-8563

Fold and Tape | **Please do not staple** | Fold and Tape

SA34-2361-01

Cut or Fold
Along Line

IBM

Part Number: 84H8595

Printed in the United States of America
on recycled paper containing 10%
recovered post-consumer fiber.