MERVA Family

MERVA Connection/6000

SH12-6097-02

MERVA Family

MERVA Connection/6000

SH12-6097-02

Note!

Before using this information and the product it supports, be sure to read the general information under “Appendix C. Notices” on page 79.

Third Edition, November 1997

This edition applies to:

Version 3 Release 3 of IBM MERVA for OS/2 LAN (5622-122)

Version 3 Release 3 of IBM MERVA for OS/2 Standalone (5622-127)

OS/2 based features of products of the MERVA family

Version 1 Release 2 of IBM MERVA for AIX (5765-449)

AIX based features of products of the MERVA family

and to all subsequent releases and modifications until otherwise indicated in new editions.

This edition replaces SH12-6097-01.

Contents

About This Book

This book is intended for application programmers who want to access an installation of Message Entry and Routing with Interfaces to Various Applications for AIX (abbreviated to MERVA AIX in this book) or an installation of Message Entry and Routing with Interfaces to Various Applications for OS/2 Version 3 (abbreviated to MERVA OS/2 V3 in this book) from an application program executing in an IBM RISC System/6000 (RS/6000).

This book can help you to install and customize MERVA Connection/6000, and to write programs using the MERVA Remote Application Program Interface (RAPI).

It is assumed that you have prior knowledge of and experience with:

- RISC System/6000
- Advanced Interactive Executive (AIX) operating system
- Personal System/2 (PS/2)
- Operating System/2 (OS/2)
- Systems Network Architecture (SNA)
- Application Programming Interface (API) of MERVA OS/2
- Application Programming Interface of MERVA AIX.

Chapter 1. Introduction to MERVA Connection/6000

This chapter introduces MERVA Connection/6000 and briefly describes the facilities supported by MERVA Connection/6000.

Objectives of MERVA Connection/6000

There is a wide range of banking applications available for the RISC System/6000 platform. While many of these applications create and process SWIFT messages, they do not provide a connection to public networks.

MERVA OS/2 and MERVA AIX provide connections to the SWIFT network (with SWIFT Link) and to other MERVA systems (with MERVA Link). Additionally, MERVA OS/2 and MERVA AIX provide an Application Program Interface (API) to access some MERVA services.

To use RISC System/6000 applications as banking applications, messages created on the RISC System/6000 must be transferred to a MERVA system. Messages received from one of these networks must be transferred from a MERVA system to the RISC System/6000.

While this can be achieved by saving messages to files and transferring the files, this solution requires operator intervention and can cause message integrity problems. It may also not be transparent to the application. Therefore, the best method is to implement a direct connection from the application on the RISC System/6000 to MERVA OS/2 or MERVA AIX, as if MERVA OS/2 or MERVA AIX was a component of the application.

MERVA Connection/6000 is a tool that makes it easier for you to implement such a solution. MERVA Connection/6000 is not a ready-to-use SWIFT interface on the RISC System/6000. It does not have a user interface.

MERVA Connection/6000 provides an interface for application programs on the RISC System/6000. It is called the Remote MERVA API. Using the Remote MERVA API, you can create an application on the RISC System/6000 to send messages to MERVA and receive messages from MERVA with a minimum effort.

Functions Provided by MERVA Connection/6000

MERVA Connection/6000 provides the complete functionality of the MERVA AIX or MERVA OS/2 API on the RISC System/6000. Additional calls are available for establishing an intersystem connection and making use of MERVA alarms. MERVA Connection/6000 provides a real-time interface to MERVA AIX or MERVA OS/2.

Language Support

Easy portability of MERVA API programs between OS/2 and AIX is provided by the C Language interface.

Security

Security aspects are dealt with by a flexible user exit interface (see “Chapter 8. Security” on page 47).

Message Integrity

A resynchronization mechanism ensures that the remote API program can provide the same level of message integrity as a local API program.

Components of MERVA Connection/6000

Figure 1 names the components of MERVA Connection/6000 and illustrates the programming concepts of MERVA Connection/6000.

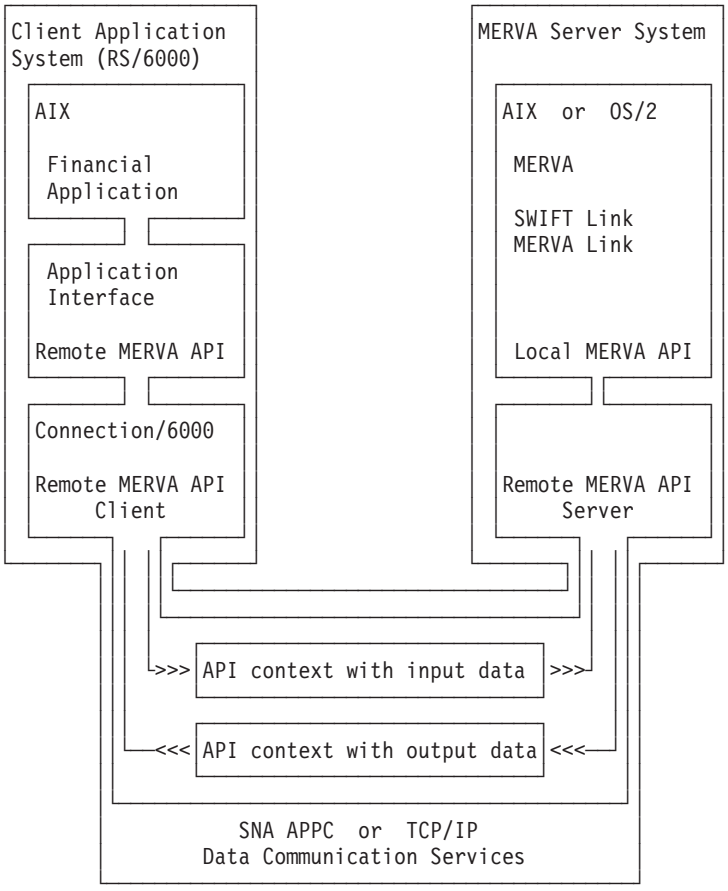

Figure 1. Concept of MERVA Connection/6000

MERVA Connection/6000 has two main components:

- The Remote MERVA API Client is installed and executes in an RS/6000, the Client Application System. MERVA AIX or MERVA OS/2 are not installed in the Client Application System.
- The Remote MERVA API Server is installed and executes in an RS/6000 or a PS/2, the MERVA Server System. The Remote MERVA API Server is a part of the MERVA AIX or MERVA OS/2 system installed in the MERVA Server System.

The *Remote MERVA API Client* provides the calling interface for a Financial Application that must use MERVA services. It forwards the API call with the input parameters to the *Remote MERVA API Server* on the MERVA Server System. The Remote MERVA API Server calls the MERVA API function and passes the received parameters. The output data and the return code of the API function are returned to

the Remote MERVA API Client. The Remote MERVA API Client returns control to the calling program as if the API function had been executed locally.

Chapter 2. MERVA Connection/6000 Client Setup

This chapter describes all aspects for the installation and customization of the Remote MERVA API Client. It starts with a description of the prerequisites for MERVA Connection/6000.

MERVA Connection/6000 Requirements

A number of requirements must be met by an RS/6000 in order to install and execute the Remote MERVA API Client.

Machine Requirements

The Remote MERVA API Client can be installed on any RS/6000 with approximately one megabyte free space on its hard disk.

The MERVA Connection/6000 Client Application System and the MERVA Server System must be interconnected by a Data Communication Link. As specified by the Data Communication Service used (SNA APPC or TCP/IP), Token Ring, SDLC, Twinax, or other types of intersystem links can be used. A corresponding data link adapter must be installed in the RS/6000.

For a list of the alternatives available and the hardware required, refer to the appropriate books listed in the bibliography.

Programming Requirements

The following software must be installed on the RISC System/6000:

- IBM AIX Version 3.2.5, or a subsequent release
- SNA Server for AIX Version 2.1 (if AIX Version 3.2.5 is installed)
- SNA Server for AIX Version 3.1, or Communications Server for AIX Version 4.1, or a subsequent release (if AIX Version 4.1 or a subsequent release is installed)
- The C Compiler XLC or compatible C Compiler.

SNA Server for AIX or Communications Server for AIX is required only if an SNA APPC connection is used for the communication between the Remote MERVA API Client and Server. SNA services are not required if a TCP/IP connection is used for that purpose.

Installing MERVA Connection/6000 Client

MERVA Connection/6000 is installed from the MERVA AIX CD. You need AIX root user authorization for this task.

To install the MERVA Connection/6000 client perform the following steps:

1. Insert the MERVA AIX CD.
2. Enter **smitty**.
3. Select **Software Installation and Maintenance**.
4. Select **Install and Update Software**.
5. Select **Install and Update from Latest Available Software**.

6. In the field **Input device / directory for software**, specify the name of the device in which you inserted the CD, for example, /dev/cd0
7. In the field **SOFTWARE to install**, specify **enmrapi**.

Deinstalling MERVA Connection/6000 Client

If you want to remove MERVA Connection/6000 files from the disk, you can deinstall MERVA Connection/6000. You need AIX root user authorization for this task.

To deinstall the MERVA Connection/6000 client perform the following steps:

1. Ensure that no Remote MERVA API program is active.
2. Enter

   ```
   smitty install
   ```
3. Select **Maintain Installed Software**
4. Select **Remove Software Products**
5. In the field **SOFTWARE name**, specify **enmrapi**.

Customizing SNA Services

MERVA Connection/6000 can use SNA APPC services for the communication between the Remote MERVA API Client and Server. As an alternative, MERVA Connection/6000 can use TCP/IP services for that purpose.

If SNA APPC services must be used, SNA Server for AIX or Communications Server for AIX must be installed in the RS/6000 and be customized for binding APPC sessions between the two partner systems.

Basic SNA Customization

Various methods apply for the interconnection of two RS/6000 systems or a RS/6000 and a PS/2. The applicable customization is described in the books of SNA Server for AIX and Communications Server.

A sample for the SNA customization that is independent of MERVA Connection/6000 is provided in “Appendix A. Sample SNA Definitions” on page 61.

SNA Customization for MERVA Connection/6000

An LU 6.2 Side Information Profile is the only resource that may need to be added to the SNA customization for access by the Remote MERVA API Client. A Side Information Profile defines a Symbolic Destination Name for the Remote MERVA API Server in the partner system. The parameters of a symbolic destination are:

- Symbolic Destination Name (sample: MERVA)
- Local LU Name (sample: LU1)
- Fully Qualified Partner LU Name (sample: APPN1.LUA)
- APPC Session Mode Name (sample: APPCLU62)
- Partner TP Name (sample: ENMRAS or ENM4XECU)

The LU names and the Mode name have been specified by the basic SNA customization. The Partner TP Name is specified by the partner, the Remote

MERVA API Server. The sample Remote MERVA API Server TP name in the MERVA AIX environment is **ENMRAS**. The sample Remote MERVA API Server TP name in the MERVA OS/2 environment is **ENM4XECU**.

If there is already a Symbolic Destination defined with all parameters correct except the TP name, there is no need to define a Symbolic Destination specifically for the Remote MERVA API Server. The existing Symbolic Destination can be used to identify the partner system and the APPC session characteristics, and the applicable TP name is specified in the MERVA Connection/6000 Application Profile. The TP name in the Side Information Profile is disregarded in this case.

Customizing TCP/IP

MERVA Connection/6000 can use TCP/IP services for the communication between the Remote MERVA API Client and Server if the server supports the TCP/IP communication protocol. The TCP/IP support for a Remote API Server has not been initially available with MERVA AIX and MERVA OS/2. This is why you must check whether the applicable Remote MERVA API Server supports TCP/IP.

Basic TCP/IP Customization

The RS/6000 must be customized as a host in an internet, a network of interconnected hosts using TCP/IP communication protocols. No specific MERVA Connection/6000 requirements apply for the basic TCP/IP customization.

TCP/IP Customization for MERVA Connection/6000

TCP/IP customization for MERVA Connection/6000 is not applicable. All information related to the TCP/IP connection to the Remote API Server is provided in the MERVA Connection/6000 Application Profile.

You must, however, ensure that the partner host name specified in the MERVA Connection/6000 Application Profile can be interpreted by the TCP/IP service. A partner host name can be interpreted if it is defined in the AIX hosts file (/etc/hosts), or if it is known by a Name Server in the TCP/IP network.

Customizing MERVA Connection/6000

Any Financial Application that uses the Remote MERVA API must be customized in the Client Application System. The most important customization information is the identification of the applicable Remote MERVA API Server.

A MERVA Connection/6000 application is customized by providing information in a MERVA Connection/6000 Application Profile, a flat ASCII file that is generated and updated using any text editor.

Two formats are supported for an application profile, a fix format profile and a variable format profile. The fix format profile supports only an SNA connection between Remote API Client and Server. The variable format supports additional functions such as TCP/IP interconnection, conversation security, and test environment.

Fix Format Application Profile

The parameters of a MERVA application can be provided in a fix format application profile that contains six parameters. The parameters can be specified in one line separated by at least one blank, in six separate lines, or as parameter groups in 2 to 5 lines.

The sequence of parameters is fix. It must be:

1. Log level (1 to 4)
2. Name of programmer's log
3. Name of diagnosis log
4. SNA symbolic destination name of the Remote API Server
5. Name of the message integrity control file
6. Client system type (RS6000)

A parameter file that starts with a digit (the log level) is interpreted as a fix format application profile. It is interpreted as a variable format application profile otherwise.

An example of a Remote API Client application profile in fix format is shown in Figure 2.

```
1
plog.log
dlog.log
MERVA
mip.ctl
RS6000
```

Figure 2. Fix Format Application Profile Sample

The fix format of an application profile is supported for compatibility reasons with older versions of the Remote MERVA API feature only. New functions, such as conversation security and TCP/IP support, are not supported by an application profile in fix format.

Application profiles in variable format must be used to benefit from the full functionality provided by the Remote API Client feature.

Variable Format Application Profile

The variable format application profile provides an extended means to specify environment parameters for a remote MERVA application. The features of the variable format are:

- Parameter Keywords

  An application parameter is specified in a line by its own in the format **parameter_keyword = parameter_value**. Any number of blanks may precede the parameter keyword, and precede and follow the mandatory equal sign.
- Parameter Sequence

  Application parameters can be specified in any sequence. If a parameter is set twice in the profile, the second of the two parameters becomes effective.
- Comments

Comments can be part of an application profile. Any line that does not start with a valid parameter keyword is considered as a comment line. An empty line is also considered as a comment line. The first line of a profile must, however, not start with a digit. According to conventions in other AIX configuration profiles, it is recommended to start a comment line with a hash charater '#'.

A parameter value can be followed by a comment. The comment must be separated from the parameter value by at least one blank.

An example of a Remote API Client application profile in variable format for an SNA connection is shown in Figure 3.

```
#--------------------------------------------------------------------------
# MERVA Connection/6000:  Client Application Profile #1
#--------------------------------------------------------------------------

 log_level            = 1            minimum log level
 log_mode             = append       append new log entries
 system_type          = RS6000       client system type is RS/6000

 programmer_log       = enmra1.plog  programmer's log file name
 diagnosis_log        = enmra1.dlog  diagnosis log file name
 control_file         = enmra1.mip   message integrity control file name

 symbolic_destination = MERVA        SNA side information profile name
 partner_tp_name      = ENMRAS       Remote API server APPC TP name

#partner_host_name    = merva        TCP/IP host name of RAPI server
#rapi_port_number     = 7118         TCP/IP port number of RAPI server
#tcp_nodelay          = on           TCP_NODELAY flag will be set

#client_user_id       = userid       conversation security user id
#client_password      = passwd       conversation security user password

#test_environment     = on           write trace to stdout
```

Figure 3. Variable Format Application Profile Sample enmra1.prf

The application profile parameters that are not supported by a fix format application profile are shown in a comment line. Remove the hash character at the begin of a comment line to activate the parameter in that line.

Variable Format Application Profile Parameters

The parameters supported by the Remote API Client and the corresponding parameter keywords in a variable format profile are described in the following.

Log Level

The parameter keyword for the log level parameter is **log_level**. The parameter value is a single digit, 1, 2, 3, or 4. Log level 4 provides the most detailed information.

Log File Mode

The parameter keyword for the log file mode parameter is **log_mode**. The parameter value is either **append** or **new**. Actually, only the initial characters 'a' or 'n' are relevant. A log file mode parameter that does not start with either of these two characters is ignored. The log file mode applies to both, the programmer's log and the diagnosis log.

Log file mode **append** means that the programmer's and diagnosis log entries are appended to existing log files. This is the default log file mode if this parameter is missing from the application profile.

Log file mode **new** means that existing log files are deleted and the programmer's and diagnosis log entries are written to an empty file.

System Type

The parameter keyword for the client system type parameter is **system_type**. The parameter value for a MERVA Connection/6000 client is **RS6000**. It identifies the type of the client system to the RAPI server.

Name of Programmer's Log

The parameter keyword for the programmer's log is **programmer_log**. The parameter value is the name of an AIX file. A programmer's log is not generated when this parameter is not specified.

Name of Diagnosis Log

The parameter keyword for the diagnosis log is **diagnosis_log**. The parameter value is the name of an AIX file. A diagnosis log is not generated when this parameter is not specified.

Name of Message Integrity Control File

The parameter keyword for the MIP control file is **control_file**. The parameter value is the name of an AIX file. The Message Integrity Control File is a mandatory resource for the Remote MERVA API Client. The Remote API Client cannot be initialized when this parameter is not specified.

SNA APPC Symbolic Destination

The parameter keyword for the SNA APPC symbolic destination is **symbolic_destination**. The parameter value is the name of a Side Information profile. It identifies and describes the APPC partner process in the Remote API Server. The maximum length of a symbolic destination name is 8 characters.

SNA APPC Partner TP Name

The parameter keyword for the SNA APPC partner TP name is **partner_tp_name**. The parameter value is the transaction program name (TPN) of the Remote API Server as it is defined in the partner system. The TP name specified in this parameter takes precedence over the TP name specified in the Side Information profile. If this parameter is not specified, the TP name specified in the Side Information profile applies. The maximum length of a TP name is 8 characters.

TCP/IP Partner Host Name

The parameter keyword for the TCP/IP partner host name is either **partner_host_name** or **partner_host**. The parameter value is the name of the AIX host that houses the RAPI server, or its dotted decimal TCP/IP address. The maximum length of a partner host name is 64 characters.

TCP/IP Port Number

The parameter keyword for the TCP/IP port number of the Remote API Server is either **rapi_port_number** or **port_number**. The parameter value is the number assigned to the MERVA Remote API Server, an inetd subserver, in the applicable partner host system. The maximum value of a TCP/IP port number is 65.535.

TCP NODELAY Option

The parameter keyword for the TCP NODELAY option is **tcp_nodelay**. The parameter value is either **1** or **on** if the TCP_NODELAY flag must be set. It is either **0** or **off** if the TCP_NODELAY flag must not be set. The default parameter value is **1**.

The TCP NODELAY option can have a significant impact on the performance of a remote API program in the TCP/IP communication environment.

Client User Identifier

The parameter keyword for the conversation security client user ID is **client_user_id**. The parameter value is the conversation security user identifier that applies for the conversation with the partner system. The client user must be defined in the partner system and must be authorized to access the Remote API Server transaction program. The maximum length of a user ID is 8 characters.

The client user ID specified in this parameter applies only if the user application program did not provide a user ID before the application profile is handled. The user ID provided by the application program can, however, be erased by **client_user_id =** ″″. A second client user ID statement in this application profile can then set a client user ID of its choice.

Client User Password

The parameter keyword for the conversation security client user password is **client_password**. The parameter value is the conversation security user password that applies for the specified client user. A password is disregarded by the Remote API Client if a user ID is not specified in the application profile or by the application program. The maximum length of a client user password is 8 characters.

The client user password specified in this parameter applies only if the user application program did not provide a user password before the application profile is handled.

The password provided by the application program can, however, be erased by **client_password =** ″″. A second client password statement in this application profile can then set a client user password of its choice.

Client Process Test Environment

The parameter keyword for the client process test environment is **test_environment**. The parameter value is either **on** or **1** to activate the test environment when the client process starts. The test environment is inactive if this parameter is not specified or if any other parameter value is specified.

The Remote API Client function ENMSetTestEnv() is a means to set or reset the client process test environment in a Remote API Client user program for specific phases of the client process.

A Remote API Client process in test environment writes a processing trace to the standard output device (normally the user terminal). This trace can be used for processing and error analysis. The programmer's log and the diagnosis log are other sources of information for error analysis.

Selecting the Communication Type

The Remote API Client can establish a conversation with a Remote API Server using SNA APPC services or using TCP/IP services. The corresponding partner system address information must be provided in the application profile, and the appropriate customization must be applied to the applicable data communication services.

SNA APPC and TCP/IP partner information can be provided in an application profile. If the SNA symbolic destination name of the Remote API Server is available, the Remote API Client tries to establish an APPC conversation with the Remote API Server. TCP/IP partner information is disregarded in this case.

TCP/IP partner information is used to establish a TCP/IP connection to the Remote API Server if an SNA symbolic destination name is not available from the application profile.

The Remote API Client does not support a preferred connection type and an automatic connection type switch if SNA APPC and TCP/IP partner information is available from the application profile.

Chapter 3. Remote API Server Setup in an AIX System

This chapter describes all aspects for the installation and customization of the Remote MERVA API Server in the MERVA AIX environment. The terms **AIX system** and **RS/6000** (RISC System/6000) are used as synonyms.

Remote API Server Requirements

A number of requirements must be met by an AIX system in order to install and execute the Remote MERVA API Server.

Machine Requirements

The MERVA Connection/6000 Client Application System and the MERVA Server System must be interconnected by a Data Communication Link. As specified by the Data Communication Service used (SNA APPC or TCP/IP), Token Ring, SDLC, Twinax, or other types of intersystem links can be used. A corresponding data link adapter must be installed in the RS/6000.

For a list of the alternatives available and the hardware required, refer to the appropriate books listed in the bibliography.

Programming Requirements

The following software must be installed on the RISC System/6000:

- IBM AIX Version 4.1.3, or a subsequent release
- SNA Server for AIX Version 3.1, or Communications Server for AIX Version 4.1, or a subsequent release

SNA Server for AIX or Communications Server for AIX is required only if an SNA APPC connection must be used for the communication between the Remote MERVA API Client and Server. SNA Server for AIX or Communications Server for AIX is not required if a TCP/IP connection is used for that purpose.

Installing the Remote API Server

The Remote MERVA API Server is automatically installed when MERVA AIX is installed in an RS/6000. No specific installation tasks apply for the Remote API server in the MERVA AIX environment.

Customizing SNA Services

MERVA Connection/6000 can use SNA APPC services for the communication between the Remote MERVA API Client and Server. As an alternative, MERVA Connection/6000 can use TCP/IP services for that purpose.

If SNA APPC services are used, SNA Server for AIX or Communications Server for AIX must be installed in the RS/6000 and be customized for binding APPC sessions between the two partner systems.

Basic SNA Customization

Various methods apply for the interconnection of two RS/6000 systems. The applicable customization is described in the books of SNA Server for AIX or Communications Server for AIX.

A sample for the SNA customization that is independent of MERVA Connection/6000 is provided in “Appendix A. Sample SNA Definitions” on page 61.

SNA Customization for the Remote API Server

An LU 6.2 TP Name Profile is the only resource that must be added to the SNA customization to support the Remote MERVA API Server. A TPN Profile defines the characteristics of an inbound APPC Transaction Program. The characteristics of an inbound TP are, for example:

- TP Name (sample: ENMRAS)
- Full Path Name of the Executable
- Command Line Parameters
- TP Access Security
- AIX User for the TP Process

To define a TPN Profile in SNA Server for AIX or Communications Server for AIX call smitty sna and select:

1. **Configure SNA Profiles**
2. **Advanced Configuration**
3. **Sessions**
4. **LU 6.2**
5. **LU 6.2 Transaction Program Name (TPN)**
6. **Add a Profile**

A sample LU 6.2 TPN profile for ENMRAS in the Communications Server for AIX environment is shown in Figure 4 on page 15.

```
                          Add LU 6.2 TPN Profile

Type or select values in entry fields.
Press Enter AFTER making all desired changes.

[TOP]                                                  [Entry Fields]
* Profile name                                        [ENMRAS]
  Transaction program name (TPN)                      [ENMRAS]
  Transaction program name (TPN) is in hexadecimal?    no                        +
  PIP data?                                            no                        +
     If yes, Subfields (0-99)                         [0]                        #
  Use command line parameters?                         yes                       +
  Command line parameters                             [trace]
  Conversation type                                    mapped                    +
  Sync level                                           confirm                   +
  Resource security level                              none                      +
     If access, Resource Security Acc List Prof.      []
  Full path to TP executable                          [/u/merva1/ipc/enmtpi.cmd]
  Multiple instances supported?                        yes                       +
  Use user id from attach?                             no                        +
  User ID                                             [210]                      #
  Server synonym name                                 [ENMRASRV]
  Restart action                                       once                      +
  Communication type                                   signals                   +
     If IPC, communication IPC queue key              [0]                        #
  Time out Attaches?                                   yes                       +
    If yes, time-out value (0-3600 seconds)           [60]                       #
  Standard INPUT file/device                          [/dev/null]
  Standard OUTPUT file/device                         [/dev/null]
  Standard ERROR file/device                          [/dev/null]



F1=Help          F2=Refresh         F3=Cancel            F4=List
F5=Reset         F6=Command         F7=Edit              F8=Image
F9=Shell         F10=Exit           Enter=Do
```

Figure 4. LU 6.2 Transaction Program Name Profile for ENMRAS

The sample TPN profile in Figure 4 defines that both the sample TPN **Profile name** and the sample **Transaction program name (TPN)** are **ENMRAS**.

Command line parameter keyword **trace** is used to request an inbound conversation trace. It is written to a file in the /tmp file system starting with the name /tmp/enmtpi.t. If you specify a trace directory name starting and ending with a forward slash instead of the keyword **trace**, a conversation trace is written to a trace file starting with enmtpi.t. in that directory.

The sample **Full path to TP executable** specifies an AIX shell script that calls the Remote API Server program **enmtpi** via a symbolic link from the MERVA AIX IPC directory (for example, /u/merva1/ipc) to the MERVA installation directory (for example, /usr/lpp/merva/bin).

The AIX shell script **enmtpi.cmd** reads, for example,

```
#!/bin/bsh
/u/merva1/ipc/enmtpi $1 $2 $3 $4 $5 $6 &
```

This sample AIX shell script ensures that the program **enmtpi** is called with the full path name as the first parameter. The full path name is needed to identify the applicable MERVA AIX instance (MERVA AIX IPC directory name).

Multiple instances of this TP must be enabled to allow concurrent inbound conversations with multiple clients.

The **User ID** 210 represents the MERVA Instance Owner **merva1** in this sample.

Customizing TCP/IP Services

MERVA Connection/6000 can use TCP/IP services for the communication between the Remote MERVA API Client and Server. TCP/IP customization applies in the Remote MERVA API Server environment.

Basic TCP/IP Customization

The RS/6000 must be customized as a host in an internet, a network of interconnected hosts using TCP/IP communication protocols. No specific MERVA Connection/6000 requirements apply for the basic TCP/IP customization.

TCP/IP Customization for the Remote API Server

The Remote MERVA API Server executes as an inetd subserver if TCP/IP services are used for the communication between the Remote MERVA API Client and Server. An inetd subserver is defined in two steps, the definition of the internet service and the definition of the inetd subserver.

Customizing Client Network Services (/etc/services)

To add an entry to the file /etc/services, call smit tcpip or smitty tcpip and select:

1. **Further Configuration**
2. **Client Network Services**
3. **Services**
4. **Add a Service**

A sample internet services profile for the Remote API Server **enmras** is shown in Figure 5. The sample TCP socket port number for the Remote MERVA API Server is **7118**. This port number aligns with other sample port numbers in the MERVA AIX environment.

```
                          Add an Internet Service

Type or select values in entry fields.
Press Enter AFTER making all desired changes.

                                                    [Entry Fields]
* Official Internet SERVICE Name                   [enmras]
* Transport PROTOCOL                                tcp                    +
* Socket PORT Number                               [7118]                   #
  Unofficial Internet SERVICE NAMES                []
  (separate names with blanks)
```

Figure 5. Internet Service Profile Sample

Customizing Super Daemon Services (/etc/inetd.conf)

To add an entry to the file /etc/inetd.conf, call smit tcpip or smitty tcpip and select:

1. **Further Configuration**
2. **Server Network Services**

3. **Other Available Services**
4. **Super Daemon (inetd)**
5. **inetd Subservers**
6. **Add an inetd Subserver**

You must specify the service name and the transport protocol in that initial screen before you can enter the other subserver parameters. A sample for the initial screen is shown in Figure 6.

```
                          Add an inetd Subserver

     Please refer to help for information concerning subserver dependencies


Type or select values in entry fields.
Press Enter AFTER making all desired changes.

                                                    [Entry Fields]
* Available Subservers                             [enmras tcp]             +
```

Figure 6. inetd Subserver Profile Identification Sample

When you have specified the subserver name and the transport protocol you must press the ENTER key to get the screen that provides the full set of subserver parameters. A sample inetd subserver profile is shown in Figure 7.

```
                          Add an inetd Subserver

     Please refer to help for information concerning subserver dependencies


Type or select values in entry fields.
Press Enter AFTER making all desired changes.

                                                    [Entry Fields]
* Internet SERVICE Name                            [enmras]
* Transport PROTOCOL                               tcp                      +
* SOCKET Type                                      stream                   +
* WAIT for Server to Release Socket                nowait                   +
* USER Name                                        [merva1]
* Service Program PATH Name                        [/usr/lpp/merva/bin/enmtci]
  Service Program Command Line ARGUMENTS           [/u/merva1/ipc/ trace]
```

Figure 7. inetd Subserver Profile Sample

The **USER Name** (for example, merva1) is the name of the MERVA Instance Owner. The service program **enmtci** executes under the identifier of this AIX user. The program must, however, have root authority at its begin to check whether the client is authorized to access the server system. This is why program **enmtci** must be owned by the root user, must be executable by members of its owning group, and must have set the AIX **setuid** flag. The user specified as **USER Name** must be a member of the group that owns program **enmtci**.

You can also use the following path name as the **Service Program PATH Name**:

```
/home/<merva instance owner>/ipc/enmtci
```

<merva instance owner> denotes the home directory of the MERVA instance owner.

The service program **enmtci** switches to normal user authority (user merva1) as soon as the client authorization has been executed.

The **Service Program Command Line ARGUMENTS** are as follows:

- The first command line argument must be the IPC directory of the applicable MERVA AIX instance.
- An inbound conversation trace can be requested in another command line argument. The command line parameter keyword 'trace' causes a conversation trace to be written to a file in the /tmp file system starting with the name /tmp/enmtci.t. If you specify a trace directory name starting and ending with a forward slash, a conversation trace is written to a trace file starting with enmtci.t. in that directory.
- Command line parameter keyword **delay** can be specified (separated from other parameters by a blank) to keep the inbound TCP/IP socket as it was passed to the inbound TP. By default, the Remote API Server TP for TCP/IP (enmtci) sets the TCP_NODELAY flag for the inbound socket. A significantly decreased execution time of a remote API application program can be the effect of that flag. Depending on the size of the API Response Data Units, the flag can have no effect at all, or save up to 90% of the execution time.

Chapter 4. Remote API Server Setup on OS/2

This chapter describes all aspects for the installation and customization of the Remote API (RAPI) Server in the MERVA OS/2 environment.

Remote API Server Requirements

A number of requirements must be met by a OS/2 system in order to install and execute the Remote MERVA API Server.

Machine Requirements

The MERVA Connection/6000 Client Application System and the MERVA Server System must be interconnected by a Data Communication Link. As specified by the Data Communication Service used (SNA APPC or TCP/IP), Token Ring, SDLC, Twinax, or other types of intersystem links can be used. A corresponding data link adapter must be installed o the OS/2 system.

For a list of the alternatives available and the hardware required, refer to the appropriate books listed in the “Bibliography” on page 89.

Programming Requirements

The following software must be installed on your OS/2 PC:

- IBM OS/2 Warp Version 3, or a subsequent release
- Personal Communications 4.1 or Communication Server 4.1 *or*
  TCP/IP for OS/2 Version 3.0, or a subsequent release
- MERVA OS/2 V3.3, including the latest available PTF level, or a subsequent release.

Installing Remote API Server

The Remote MERVA API Server is automatically installed when MERVA OS/2 is installed on a OS/2 system. Specific customization tasks apply for the Remote API Server in the OS/2 environment.

Installing the Remote MERVA API Server Program

Refer to “Customizing SNA Services” on page 20 or “Customizing TCP/IP Services” on page 22 on how to install MERVA Connection/6000 with SNA or TCP/IP.

Refer to “Chapter 10. Replacing Security User Exits” on page 55 on how to replace the Security User Exits provided with MERVA Connection/6000.

Installing the Sample Communications Server Configuration Files

The directory \SAMPLES\CONNECT\PC\ on the MERVA OS/2 V3.3 CD contains a sample set of Communications Server configuration files. If you want to use these, unzip them to the directory where Communications Server is installed:

unzip enmsrvnn.zip *:* -d x :\CMLIB *or*

unzip enmsrvpp.zip *:* -d x :\CMLIB

in which *x* is the drive where Communications Server is installed on (usually the CMLIB directory). The zip file **enmsrvnn.zip** contains a Communications Server configuration file using APPN on the Server side. The zip file **enmsrvpp.zip** contains a Communications Server configuration file using a peer-to-peer connection for the Server side. Change the name of the default configuration file in Communications Server to **enmsrvnn** or **enmsrvpp** by double-clicking on the icon "Replace Default Configuration" of the Communications Server folder. The files with the extension **NDF** are plain text files and can be read by any text editor. All other files can only be read by the Communications Server setup program.

Customizing SNA Services

The Remote API Server can use SNA APPC services for the communication between the Remote MERVA API Client and Server. As an alternative, the Remote API Server can use TCP/IP services for that purpose.

If SNA APPC services are used, one of the following programs must be installed and be customized for binding APPC sessions between the two partner systems on the OS/2 system:

- Personal Communications 4.1
- Communication Server 4.1

Whenever the term **Communications Server** is used in this book, **Personal Communications** is also meant (both are sufficient for MERVA Connection/6000).

If TCP/IP is used, TCP/IP for OS/2 Version 3.0 or higher must be installed on the OS/2 system.

Basic SNA Customization

Various methods apply for the interconnection of two OS/2 system. The applicable customization is described in the books of Communications Server.

SNA Customization for the Remote API Server

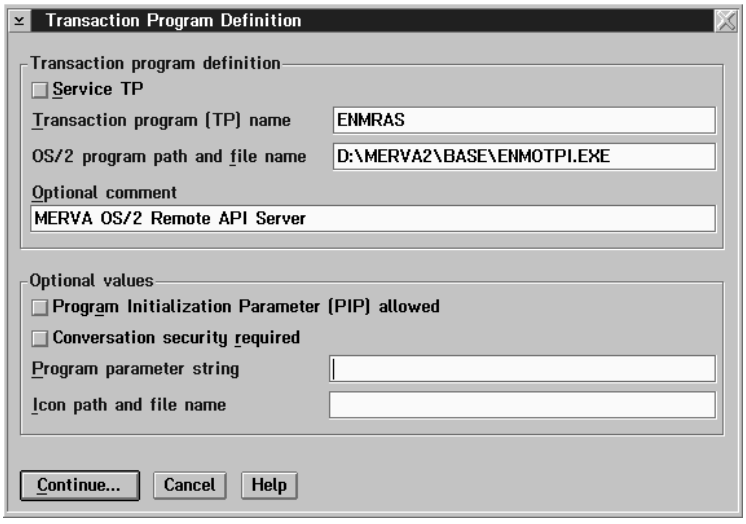

Figure 8. Transaction Program Definition in Communications Server Setup

An LU 6.2 TP is the only resource that must be added to the Communications Server customization to support the Remote MERVA API Server. A TP defines the characteristics of an inbound APPC Transaction Program. The characteristics of an inbound TP are, for example:

- TP Name (example: ENMRAS)
- Full Path Name of the Executable
- Command Line Parameters
- TP Access Security

A sample LU 6.2 TP definition is shown in Figure 8.

If **Conversation security** is used, an appropriate entry must be added to *Conversation security* in the SNA Features List. It is recommended to use the *Utilize User Profile Management* option. Therefore, any user registered in the Server's User Profile Management has automatically the access right for the Transaction Program.

Customizing the Trace File for SNA

The trace file path must be set with an Environment variable. Add the following line to CONFIG.SYS:

```
SET ENM_TRC_DIR=C:\TRACE\
```

The path name given is the path to a directory where all trace files will be written. Replace 'C:\TRACE\' with an appropriate path name. The trace file names will be of the type

```
<partner host name>.trc
```

If the ENM_TRC_DIR environment variable is not set, no trace file will be written.

Note that this change to CONFIG.SYS does not take effect until the OS/2 system is rebooted.

Customizing TCP/IP Services

MERVA Connection/6000 can also use TCP/IP services for the communication between the Remote MERVA API Client and Server. As prerequisite, TCP/IP for OS/2 Version 3.0 must be installed on the Remote API Server and all Remote API Clients.

The Remote MERVA API Server executes as an **inetd** subserver if TCP/IP services are used for the communication between the Remote MERVA API Client and Server. An inetd subserver is defined in two steps, the definition of the internet service and the definition of the inetd subserver.

Customizing Client Network Services

The file **C:\MPTN\ETC\SERVICES** contains all TCP/IP services available on the Remote API Server. It is used to map a service to a specific port and a transport protocol.

Add the following line to the **C:\MPTN\ETC\SERVICES** file:

```
enmras          7118/tcp                # MERVA Connection Remote Api Server
```

This defines the TCP/IP service ‘enmras’, maps it to the port 7118, and defines ‘tcp’ as transport protocol for this service. The name of the service and the port number may be freely choosen. The service name should match the name given in **INETD.LST** (see below) and the port number must match the port number given in the profile of the Client.

Customizing Super Daemon Services

The file

```
%ETC%\INETD.LST
```

contains all services that were started with the **inetd** daemon. Next to the service name, the transport protocol and the executable file to start are indicated.

Add the following line to the %ETC%\INETD.LST file:

```
enmras tcp <drive>:\merva2\base\enmotci.exe
```

Note: The implementation of **inetd** in TCP/IP for OS/2 3.0 is somewhat incomplete in comparison with TCP/IP on AIX. No parameters are allowed in %ETC%\INETD.LST and the Configuration program supplied with TCP/IP for OS/2 3.0 doesn’t allow selfdefined services for inetd. Particulary, take the following into consideration: **The TCP/IP Configuration program overrides all changes made in %ETC%\INETD.LST! That is, if the configuration is saved with the Configuration program, all changes made to %ETC%\INETD.LST must be done again!**

Customizing the Trace File for TCP/IP

The trace file path must be set with an Environment variable. Add the following line to CONFIG.SYS:

```
SET ENM_TRC_DIR=C:\TRACE\
```

The path name given is the path to a directory where all trace files will be written. Replace ‘C:\TRACE\’ with an appropriate path name. The trace file names will be of the type

```
<partner host name>.trc
```

If the ENM_TRC_DIR environment variable is not set, no trace file will be written.

Note that this change to CONFIG.SYS does not take effect until the OS/2 system is rebooted.

Chapter 5. Verifying Correct Installation and Customization

To verify that the installation and customization of MERVA Connection/6000 was successful, run the sample program smp6le4.

Before you can run this program, the user ID SAMPLE with the password SAMPLE1 has to be defined in MERVA. This user ID must be approved for application programs. The program also checks that the queues API_IN and API_OUT have been customized.

Chapter 6. The Application Programming Interface

The following description of the API is based on the descriptions in *MERVA OS/2 V3 Application Programming* and *MERVA AIX Application Programming*. This chapter describes only the differences between the MERVA API programming on the RISC System/6000 and the MERVA OS/2 API programming on the PS/2 or the MERVA AIX API programming on the RISC System/6000.

Structure of the MERVA API Program on the Client Side

One major task of the MERVA API program on the RISC System/6000 is that it must call functions that deal with connecting and disconnecting to and from the PS/2 or the RISC System/6000:

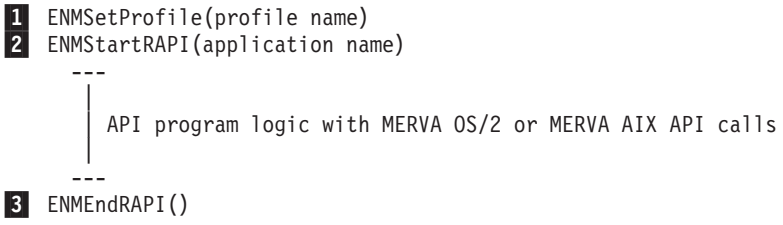

Figure 9. Remote MERVA API Program Structure

1 Before the API functions can be called, the Remote MERVA API Client on the RISC System/6000 must be initialized by calling the function ENMSetProfile. This function tells the Remote MERVA API Client the name of the profile. The profile is described in “Customizing MERVA Connection/6000” on page 7.

2 After having set the profile name, the connection to the Remote MERVA API Server on the MERVA OS/2 or MERVA AIX side can be established. To do this, call the function ENMStartRAPI. When this function is called, the Remote MERVA API Client is initialized and the network connection to the Remote MERVA API Server is established.

After the ENMStartRAPI call, the MERVA OS/2 or MERVA AIX API functions can be called as if the program ran locally on a MERVA OS/2 or MERVA AIX machine.

3 Before terminating the program, the connection to the Remote MERVA API Server must be released by calling the function ENMEndRAPI. It is important to call this function even if an error occurs in the API program, otherwise, the Remote MERVA API Server misses the termination and is not ready to receive the next connection request when the API program is started again.

C Language Data Types

The file enmrapi.h contains the data types and prototypes of the MERVA OS/2 and MERVA AIX API functions. When compiling a MERVA OS/2 or MERVA AIX API program locally on the MERVA machine, the file enmoapi.h is included. When compiling an API program on the RISC System/6000, the file enmrapi.h is included instead.

For the description of the API calls in this book, some data types defined in the supplied include file enmrapi.h are used. Their meanings are as follows:

Type	Definition
USHORT	unsigned short
UCHAR	unsigned char
PUCHAR	unsigned char*
PUSHORT	unsigned short*
ULONG	unsigned long
PULONG	unsigned long*

Additional Functions

MERVA Connection/6000 provides more API calls than the MERVA OS/2 and MERVA AIX API. They are divided into the following categories:

- Functions for starting and ending the conversation
- Functions enabling the API program to be triggered by MERVA OS/2 alarms
- Functions for error handling.

Starting and Ending the Conversation

If you want that the API program starts and ends the conversation between the Remote MERVA API Client and the Remote MERVA API Server, use the following functions:

- ENMSetProfile - Select a Profile
- ENMStartRAPI - Establish Connection to MERVA
- ENMRestartRAPI - Reconnect Remote API Program to MERVA
- ENMEndRAPI - Disconnect from MERVA
- ENMSetSecurity - Set Conversation Security Information
- ENMSetTestEnv - Set test environment

Each function is described in the following.

ENMSetProfile - Select a Profile

Specify the name of the profile you want to use.

C Definition

```
void ENMSetProfile (PUCHAR pucProfileName);
```

Parameter Description

The following parameter is required:

- **pucProfileName**(PUCHAR)

  Pointer to a null-terminated string with a maximum length of 80 characters. This is the full path name of the profile.

  Note: If several API programs run concurrently, each must use a different profile name.

Remarks

The format and contents of the profile file are described in “Customizing MERVA Connection/6000” on page 7.

C Language Example:

```
#include "enmrapi.h"

ENMSetProfile ("enm6r1.prf");
```

ENMStartRAPI - Establish Connection to MERVA Server

C Definition

```
USHORT ENMStartRAPI ( PUCHAR pucApplicationName );
```

Parameter Description

The following parameters are required:

- **retCode**(USHORT) - output

Code	Meaning
2	An internal error has occurred. The API program receives further information by calling the function ENMGetReason, described in “Handling Errors” on page 39. The reason code is also written to the diagnosis log on the RISC System/6000 (see “Chapter 11. Diagnosis Information” on page 59). If it is an internal error of the MERVA OS/2 or MERVA AIX API, the reason code is zero.

- **pucApplicationName**(PUCHAR) - input

  A pointer to a null-terminated string of up to 8 characters. This name is registered by the Remote MERVA API Server.

  Note: If several API programs run concurrently, each must use a different name.

Remarks

This call establishes the conversation with MERVA (Remote MERVA API Server) and initializes internal buffers and variables. After this function was called, the program must not end without calling ENMEndRAPI.

C Language Example

```
#include "enmrapi.h"

USHORT rc = 0;

if ((rc = ENMStartRAPI ( "APPLID" )) == 0 )
    puts("Conversation is up\n");
else
    puts("Error in ENMStartRAPI");
```

ENMRestartRAPI - Reconnect to MERVA Server

C Definition

```
USHORT ENMRestartRAPI ( PUCHAR pucApplicationName );
```

Parameter Description

The following parameters are required:

- **retCode**(USHORT) - output

Code	Meaning
2	An internal error has occurred. The API program receives further information by calling the function ENMGetReason, described in “Handling Errors” on page 39. The reason code is also written to the diagnosis log on the RISC System/6000 (see “Chapter 11. Diagnosis Information” on page 59). If it is an internal error of the MERVA OS/2 or MERVA AIX API, the reason code is zero.

- **pucApplicationName**(PUCHAR) - input

  A pointer to a null-terminated string of up to 8 characters. This name is registered by theRemote MERVA API Server.

  Note: If several API programs run concurrently, each must use a different name.

Remarks

If the connection is established with this call instead of ENMStartRAPI, the resynchronization described in Figure 10 on page 43 is provided for the following API calls:

- ENMAdd
- ENMDelete
- ENMPut
- ENMRouteAdd
- ENMRoutePut

If the connection was not interrupted within the critical time period in a previous session, this call has the same functions as ENMStartRAPI. Therefore, you can also use it if the previous connection did not end abnormally.

C Language Example

```
#include "enmrapi.h"

USHORT rc = 0;

if ((rc = ENMRestartRAPI ( "APPLID" )) == 0 )
    puts("Conversation is up\n");
else
    puts("Error in ENMRestartRAPI");
```

ENMEndRAPI - Disconnect from MERVA Server

C Definition

```
USHORT ENMEndRAPI ( void );
```

Parameter Description

The following parameter is required:

- **retCode**(USHORT) - output

  Code Meaning

2 An internal error has occurred. The API program receives further information by calling the function ENMGetReason, described in “Handling Errors” on page 39. The reason code is also written to the diagnosis log on the RISC System/6000 (see “Chapter 11. Diagnosis Information” on page 59). If it is an internal error of the MERVA OS/2 or MERVA AIX API, the reason code is zero.

Remarks

The RAPI conversation to MERVA is terminated.

C Language Example

```
#include "enmrapi.h"

USHORT rc = 0;

if ((rc = ENMEndRAPI ()) == 0 )
    puts("Conversation successfully terminated\n");
else
    puts("Error in ENMEndRAPI");
```

ENMSetSecurity - Set Conversation Security Information

C Definition

```
VOID ENMSetSecurity ( PUCHAR pucUserID,
                      PUCHAR pucPassword );
```

Parameter Description

The following parameters are required:

- **pucUserID**(PUCHAR) - input

  A pointer to a null-terminated string of up to 8 characters containing the client user ID.

- **pucPassword**(PUCHAR) - input

  A pointer to a null-terminated string of up to 8 characters containing the client password.

Remarks

A MERVA application program can provide conversation security information to be used for client authorization in the Remote API Server system. The function **ENMSetSecurity()** must be used for that purpose. The parameters of this function are a client user ID and a password. Either of these parameters or both can be empty.

Conversation security information must be provided before **ENMStartRAPI()** or **ENMRestartRAPI()** is issued. An **ENMSetSecurity()** function call has no effect thereafter.

Conversation security information can also be provided via application profile parameters. Normally, the information provided by **ENMSetSecurity()** takes precedence over profile parameters. There is, however, a means to overwrite the security information set by **ENMSetSecurity()** via application profile parameters.

C Language Example

```
#include "enmrapi.h"

ENMSetSecurity ("SAMPLE1", "SAMPLEPW");
```

ENMSetTestEnv - Set Test Environment

C Definition

```
VOID ENMSetTestEnv ( UCHAR ucTestEnvIndicator );
```

Parameter Description

The following parameter is required:

- **ucTestEnvIndicator**(UCHAR) - input

  Function parameter 1 activates the test environment, 0 inactivates it.

Remarks

A MERVA application program can activate or inactivate the Remote API Client test environment for specific sections of the application program. The function **ENMSetTestEnv()** must be used for that purpose. It can be called as often as required.

The variable **ENMTestEnv** is provided as part of the Remote MERVA API to test whether the Remote API Client test environment is active or inactive. The instruction **ENMSetTestEnv(!ENMTestEnv);** toggels the test environment setting either from active to inactive, or from inactive to active.

C Language Example

```
#include "enmrapi.h"

#define TESTENV_ON  '1'

ENMSetTestEnv (TESTENV_ON);
```

Functions Enabling the API Program to Be Triggered

If you want that the API program is triggered by MERVA alarms, use the following functions (the semaphores reside on the MERVA system):

- ENMWaitSemList - Wait for a List of Semaphores
- ENMCloseSem - Close a Semaphore
- ENMSetSem - Set a Semaphore
- ENMClearSem - Clear a Semaphore
- ENMCreateSem - Create a Semaphore
- ENMOpen - Open a Semaphore.

Each function is described in the following.

ENMWaitSemList - Wait for a List of Semaphores

This call blocks the current process until one of the specified semaphores is cleared. It allows the API program to wait for a list of up to 16 semaphores and up to 16 different MERVA alarms.

C Definition

```
USHORT ENMWaitSemList(PUSHORT Index,
                      ULONG   timeout,
                      ULONG   SemHandle,
                              ...,
                     (ULONG)  0);[1]
```

Parameter Description

The following parameters are required:

- **retCode**(USHORT) - output

Code	Meaning
2	An internal error has occurred. The API program receives further information by calling the function ENMGetReason, described in “Handling Errors” on page 39. The reason code is also written to the diagnosis log on the RISC System/6000 (see “Chapter 11. Diagnosis Information” on page 59). If it is an internal error of the MERVA OS/2 or MERVA AIX API, the reason code is zero.
121	No semaphore is cleared. The timeout was reached.
Others	See the description of OS/2 system call DosMuxSemWait in the *OS/2 Programming Tools and Information Version 1.3 Control Program Programming Reference*.

- **Index**(PUSHORT) - output

  In this parameter, ENMWaitSemList returns an index (0..15) that tells you which of the semaphores is cleared.

- **timeout**(ULONG) - input

Code	Meaning
-1	Wait indefinitely for a semaphore to be cleared.
0	Return immediately.
>1	Wait the indicated number of milliseconds for a semaphore to be cleared before resuming execution.

- **SemHandle**(ULONG) - input

  Up to 16 semaphore handles that were created by the calls of ENMCreateSem or ENMOpenSem.

- **(ULONG)0** - input

  This parameter terminates the list of semaphores. Its value must be zero and a 4-byte value. If the parameter is missing, ENMWaitSemList is not able to recognize the end of the semaphore list.

1. The last parameter ((ULONG)0) is not part of the C function prototype. It is only mentioned here to show that the list of SemHandle parameters must be terminated by the value 0 (4 bytes).

C Language Example

```
/*
   If a connection to MERVA OS/2 should be used,
   use the following define statements:
*/
#define TRIGGER "\\SEM\\SAMPLE2"
#define STOP    "\\SEM\\STOP.SMP"
/*
   If a connection to MERVA AIX should be used,
   use the following define statements:
*/
/*
  #define TRIGGER "SAMPLE2"
  #define STOP    "STOP.SMP"
*/
#include "enmrapi.h"

   USHORT      rc = 0;
   ULONG       SemTrigger;
   ULONG       SemStop;
   USHORT      Index = 0;

   if ((rc = ENMCreateSem (&SemStop, STOP)) == 0)
      if ((rc = ENMCreateSem (&SemTrigger, TRIGGER)) == 0)
         if (( rc = ENMSetSem (SemStop)) == 0)
            if ((rc = ENMSetSem(SemTrigger)) == 0)
               rc = ENMWaitSemList(&Index, -1L,
                                   SemStop,
                                   SemTrigger,
                                   (ULONG)0);
```

ENMCloseSem - Close a Semaphore

This call closes a semaphore obtained with an ENMCreateSem or ENMOpenSem call.

C Definition

```
USHORT ENMCloseSem (ULONG SemHandle);
```

Parameter Description

The following parameters are required:

- **retCode**(USHORT) - output

Code	Meaning
2	An internal error has occurred. The API program receives further information by calling the function ENMGetReason, described in “Handling Errors” on page 39. The reason code is also written to the diagnosis log on the RISC System/6000 (see “Chapter 11. Diagnosis Information” on page 59). If it is an internal error of the MERVA OS/2 or MERVA AIX API, the reason code is zero.
102	A semaphore is set and therefore cannot be closed.
Others	See the description of the OS/2 system call DosCloseSem in the *OS/2 Programming Tools and Information Version 1.3 Control Program Programming Reference*.

- **SemHandle**(ULONG) - input

  Generated by ENMCreateSem or ENMOpenSem.

C Language Example

```
/*
   If a connection to MERVA OS/2 should be used,
   use the following define statements:
*/
#define TRIGGER "\\SEM\\SAMPLE2"
/*
   If a connection to MERVA AIX should be used,
   use the following define statements:
*/
/*
  #define TRIGGER "SAMPLE2"
*/
#include "enmrapi.h"

   USHORT      rc = 0;
   ULONG       SemTrigger;

   if ((rc = ENMCreateSem (&SemTrigger, TRIGGER)) == 0)
      rc = ENMCloseSem (SemTrigger);
```

ENMSetSem - Set a Semaphore

Remarks

ENMSetSem sets a semaphore unconditionally. For MERVA OS/2 or MERVA AIX this means that the semaphore can be cleared by raising an alarm.

C Definition

```
USHORT ENMSetSem (ULONG SemHandle);
```

Parameter Description

The following parameters are required:

- **retCode**(USHORT) - output

Code	Meaning
2	An internal error has occurred. The API program receives further information by calling the function ENMGetReason, described in “Handling Errors” on page 39. The reason code is also written to the diagnosis log on the RISC System/6000 (see “Chapter 11. Diagnosis Information” on page 59). If it is an internal error of the MERVA OS/2 or MERVA AIX API, the reason code is zero.
100	The limit of open semaphores in the system is exceeded.
103	Too many semaphore requests on the system.
Others	See the description of the OS/2 system call DosSemSet in the *OS/2 Programming Tools and Information Version 1.3 Control Program Programming Reference*.

- **SemHandle**(ULONG) - input

  Generated by ENMCreateSem or ENMOpenSem.

C Language Example

```
/*
   If a connection to MERVA OS/2 should be used,
```

```
   use the following define statements:
*/
#define TRIGGER "\\SEM\\SAMPLE2"
/*
   If a connection to MERVA AIX should be used,
   use the following define statements:
*/
/*
  #define TRIGGER "SAMPLE2"
*/
#include "enmrapi.h"

   USHORT      rc = 0;
   ULONG       SemTrigger;

   if ((rc = ENMCreateSem (&SemTrigger, TRIGGER)) == 0)
      rc = ENMSetSem (SemTrigger);
```

ENMClearSem - Clear a Semaphore

This call clears a semaphore unconditionally. If processes are blocked on the semaphore, they are restarted.

C Definition

```
USHORT ENMClearSem (ULONG SemHandle);
```

Parameter Description

The following parameters are required:

- **retCode**(USHORT) - output

Code	Meaning
2	An internal error has occurred. The API program receives further information by calling the function ENMGetReason, described in “Handling Errors” on page 39. The reason code is also written to the diagnosis log on the RISC System/6000 (see “Chapter 11. Diagnosis Information” on page 59). If it is an internal error of the MERVA OS/2 or MERVA AIX API, the reason code is zero.
101	A semaphore cannot be cleared because it is owned by another process.
Others	See the description of the OS/2 system call DosSemClear in the *OS/2 Programming Tools and Information Version 1.3 Control Program Programming Reference*.

- **SemHandle**(ULONG) - input

  Generated by ENMCreateSem or ENMOpenSem.

C Language Example

```
/*
   If a connection to MERVA OS/2 should be used,
   use the following define statements:
*/
#define TRIGGER "\\SEM\\SAMPLE2"
/*
   If a connection to MERVA AIX should be used,
   use the following define statements:
*/
/*
  #define TRIGGER "SAMPLE2"
```

```
*/
#include "enmrapi.h"

   USHORT      rc = 0;
   ULONG       SemTrigger;

   if ((rc = ENMCreateSem (&SemTrigger, TRIGGER)) == 0)
      rc = ENMClearSem (SemTrigger);
```

ENMCreateSem - Create a Semaphore

This call creates an OS/2 or AIX semaphore. The semaphore is used by several API programs to synchronize their access to resources or to wait for MERVA alarms.

C Definition

```
USHORT ENMCreateSem (PULONG SemHandle,
                     PUCHAR SemName);
```

Parameter Description

The following parameters are required:

- **retCode**(USHORT) - output

Code	Meaning
2	An internal error has occurred. The API program receives further information by calling the function ENMGetReason, described in “Handling Errors” on page 39. The reason code is also written to the diagnosis log on the RISC System/6000 (see “Chapter 11. Diagnosis Information” on page 59). If it is an internal error of the MERVA OS/2 or MERVA AIX API, the reason code is zero.
87	One of the parameters is not valid.
100	The limit of open semaphores in the system is exceeded.
123	The name of the semaphore is not valid.
183	The semaphore already exists.
Others	See the description of the OS/2 system call DosCreateSem in the *OS/2 Programming Tools and Information Version 1.3 Control Program Programming Reference*.

- **SemHandle**(PULONG) - output

  Address of the semaphore handle.

- **SemName**(PUCHAR) - input

  Pointer to a null-terminated string containing the name of the semaphore to be created. In MERVA AIX the semaphore name is a logical name without path details. In MERVA OS/2 it must be a full OS/2 path name starting with \SEM\.

C Language Example

```
/*
   If a connection to MERVA OS/2 should be used,
   use the following define statements:
*/
#define TRIGGER "\\SEM\\SAMPLE2"
/*
```

```
   If a connection to MERVA AIX should be used,
   use the following define statements:
*/
/*
  #define TRIGGER "SAMPLE2"
*/
#include "enmrapi.h"

   USHORT      rc = 0;
   ULONG       SemTrigger;

   rc = ENMCreateSem (&SemTrigger, TRIGGER);
```

ENMOpenSem - Open a Semaphore

This call opens an existing semaphore created by another process with ENMCreateSem. The other process can also run on the PS/2 or the RISC System/6000.

C Definition

```
USHORT ENMOpenSem (PULONG SemHandle,
                   PUCHAR SemName);
```

Parameter Description

The following parameters are required:

- **retCode**(USHORT) - output

Code	Meaning
2	An internal error has occurred. The API program receives further information by calling the function ENMGetReason, described in “Handling Errors” on page 39. The reason code is also written to the diagnosis log on the RISC System/6000 (see “Chapter 11. Diagnosis Information” on page 59). If it is an internal error of the MERVA OS/2 or MERVA AIX API, the reason code is zero.
100	Limit of open semaphores in the system is exceeded.
123	The name for the semaphore is not valid.
187	The semaphore does not exist.
Others	See the description of the OS/2 system call DosOpenSem in the *OS/2 Programming Tools and Information Version 1.3 Control Program Programming Reference*.

- **SemHandle**(PULONG) - output

  Address of the handle of the opened semaphore.

- **SemName**(PUCHAR) - input

  Pointer to a null-terminated string containing the name of the semaphore to be opened.

C Language Example

```
/*
   If a connection to MERVA OS/2 should be used,
   use the following define statements:
*/
#define TRIGGER "\\SEM\\SAMPLE2"
```

```
/*
   If a connection to MERVA AIX should be used,
   use the following define statements:
*/
/*
  #define TRIGGER "SAMPLE2"
*/
#include "enmrapi.h"

   USHORT      rc = 0;
   ULONG       SemTrigger;

   rc = ENMOpenSem (&SemTrigger, TRIGGER);
```

Handling Errors

If you want that the API call returns reason codes, use the function ENMGetReason - Get Reason Code for Internal Error. This function is described in the following.

ENMGetReason - Get Reason Code for Internal Error

This call returns the reason code for an internal error in MERVA Connection/6000.

If an internal error occurs either in MERVA Connection/6000 or in the local MERVA OS/2 or MERVA AIX API, an API call returns the return code 2. If it is an error of the MERVA Connection/6000, ENMGetReason returns a more specific reason code. Otherwise, the reason code is 0.

C Definition

```
USHORT ENMGetReason (void);
```

Parameter Description

The following parameter is required:

- **retCode**(USHORT) - output

Code	Meaning
2xxx	All reason codes between 2000 and 2999 indicate communication problems.
2110	The APPC conversation cannot be established or is canceled.
2120	The Communications Side Information object is not found.
2130	Connection to Remote MERVA API Server program failed.
2140	Deallocation failed because the conversation has already been terminated.
2150	Conversation is interrupted while trying to receive data.
2200	An empty data buffer was received.
28xx	xx is a return code of the MERVA Connection/6000 TCP/IP services program.
29xx	xx is a return code of the CPI-C call.
2999	A general communication problem occurred (see diagnosis log).
7006	The Remote MERVA API Server failed while allocating memory.

7012 The Remote MERVA API Server does not accept further API calls due to a previous error.

7013 The Remote MERVA API Server received a negative return code from user exit ENM4ExitDecrypt.

7014 The Remote MERVA API Server received a negative return code from user exit ENM4ExitEncrypt.

7015 The Remote MERVA API Server received a negative return code from user exit ENM4ExitMacVerify or ENM4ExitMacGen.

7016 The Remote MERVA API Server received an incorrect API request.

7018 The Remote MERVA API Server received an error when converting ASCII to EBCDIC. See the diagnosis log of MERVA OS/2 or MERVA AIX.

7019 The Remote MERVA API Server received an error while accessing the message integrity control file.

7030 Internal message space has not been created.

8002 The Remote MERVA API Client cannot open the programmer's log file specified in the profile.

8003 The Remote MERVA API Client cannot close the programmer's log file specified in the profile.

8004 The Remote MERVA API Client cannot open the diagnosis log file specified in the profile.

8005 The Remote MERVA API Client cannot close the diagnosis log file specified in the profile.

8006 The Remote MERVA API Client could not allocate memory.

8007 The Remote MERVA API Client cannot write to the diagnosis log file specified in the profile.

8008 The Remote MERVA API Client cannot write to the programmer's log file specified in the profile.

8010 The Remote MERVA API Client failed because the profile name in ENMSetProfile was incorrect or was not specified.

8011 The Remote MERVA API Client failed because the profile specified in ENMSetProfile does not exist.

8013 The Remote MERVA API Client received a negative return code from user exit ENM4ExitDecrypt.

8014 The Remote MERVA API Client received a negative return code from user exit ENM4ExitEncrypt.

8015 The Remote MERVA API Client received a negative return code from user exit ENM4ExitMacVerify.

8016 The Remote MERVA API Client received a negative return code from user exit ENM4ExitMacGen.

8017 Conversation has not been started with ENMStartRAPI (or with ENMStartAPPC).

8019 The Remote MERVA API Client could not access the message integrity control file.

C Language example

```
#include "enmrapi.h"

   USHORT   rc     = 0;
   USHORT   reason = 0;

   rc = ENMFree();
   if (rc) {
      reason = ENMGetReason();
      if (reason) {
      printf ("Internal error in Connection/6000 occurred, reason code %d",
            reason);
      }
   }
```

ENMGetReason

Chapter 7. Resynchronization

If a network connection is interrupted, the recovery procedure must ensure that all changes of message status in MERVA (such as Add, Route, or Delete) are done only once. This affects both programs using the remote API and programs calling the local MERVA OS/2 or MERVA AIX API.

During normal processing, an API call is transferred from the Remote MERVA API Client to the Remote MERVA API Server (positions (1) and (2) in Figure 10). The return data from MERVA is transferred back from the Remote MERVA API Server to the Remote MERVA API Client (positions (3) and (4)) and to the calling program.

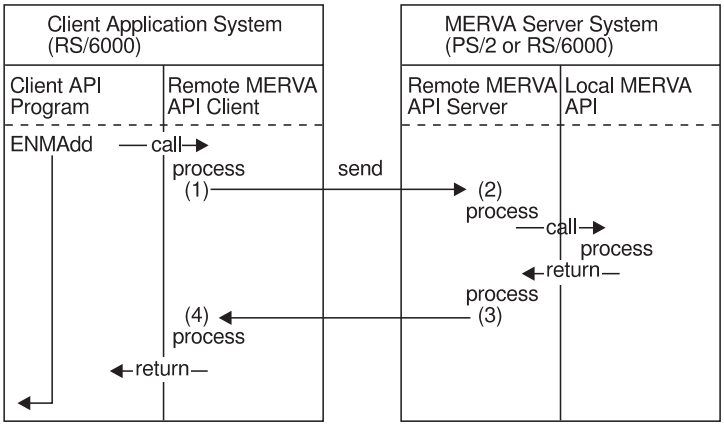

Figure 10. Resynchronization Support

The return code ERR_SYSTEM of the API call and a corresponding reason code (2000 to 2999) of an additional ENMGetReason call indicates whether the network connection is interrupted. MERVA Connection/6000 does not know whether the call completed successfully, unsuccessfully, or whether it is not executed on the MERVA system. In the example shown in Figure 10 this means that the API program does not know whether the message has been added to the MERVA queue.

With MERVA Connection/6000the API program reestablishes the connection in the next run using ENMRestartRAPI. It recreates the message with the same contents and fields, and repeats the call that failed. This mechanism is provided for those API calls that are important for the integrity of the message database:

- ENMAdd
- ENMDelete
- ENMPut
- ENMRouteAdd
- ENMRoutePut

How Resynchronization Is Implemented

The Remote MERVA API Client generates an internal unique identifier when it receives a call from the application program. The identifier is saved locally and also sent to the Remote MERVA API Server. The Remote MERVA API Server deletes the identifier after the API call has been executed and the return data is passed back to the Remote MERVA API Client.

If the network connection terminates before the return data is passed back, identifier and return data are saved. After the connection is reestablished, the same identifier arrives with the first of the above mentioned API calls. The saved return data is passed back as if the call was executed now.

The necessary control data is saved in files. On the Remote MERVA API Client you can specify the file name in the MERVA Connection/6000 profile as described in “Customizing MERVA Connection/6000” on page 7. On the Remote MERVA API Server the file name must be the same as the application name specified in the ENMStartRAPI or ENMRestartRAPI call.

To ensure that resynchronization works correctly, note the following:

- Specify unique file names for the Message Integrity Control file (MIP) in the profiles of your application programs.
- Use unique application names for the ENMStartRAPI and ENMRestartRAPI calls if you run more than one remote API program.

Using the Resynchronization Mechanism

The following is a program that issues calls in a loop:

```
ENMSetProfile
ENMRestartRAPI
ENMAttach
do
   ENMCreate
   ENMWriteField
   read message from application
   ENMRouteAdd
until (no more message to send)
ENMDetach
ENMEndRAPI
```

If the network connection breaks down after the ENMRouteAdd call is issued, the API program terminates. When it is restarted, the loop is entered as if there had been no interruption in the previous run.

Notes:

1. Use the same profile as in the previous run.
2. Call ENMRestartRAPI using the same application name.
3. Call ENMCreate and ENMWriteField using the same data as in the previous run (same message, same field contents).
4. Call ENMRouteAdd using the same queue name.
5. After resynchronization continue with the loop as in normal processing.

If the program runs like that, it does not have to check how far processing went in the previous run when the ENMRouteAdd call was interrupted.

Hints and Tips

Recovering after a Failed Call

If calling ENMAdd or ENMRouteAdd fails, you usually call ENMClear to clear the message space (see *MERVA AIX Application Programming*).

If these calls fail after reestablishing the connection as described before because of other reasons than network problems, calling ENMClear may return the return code

`ERR_NO_MSG_CREATED`

(that is, 202).

This means that the API call was executed in the first run. The error message can be ignored.

The same applies to an ENMFree call returning the message

`ERR_NO_MSG_LOCKED`

(that is, 201)

after calling of ENMDelete, ENMPut, or ENMRoutePut failed.

Not Using Resynchronization

If you do not use the resynchronization option, call ENMStartRAPI instead of ENMRestartRAPI. ENMStartRAPI deletes the internal control information for resynchronization. Then each API call is considered as a new one.

MERVA Connection/6000 does not save the type or input data of the API call that failed due to the network failure. Therefore, when using ENMRestartRAPI, you must ensure that the same call is repeated after reconnecting to the MERVA system if one of the above mentioned calls failed.

MERVA Connection/6000 does not recognize an inappropriate API call. The call is not executed if the internal state indicates that the last API call from the previous run was executed. If this is not considered, an API call with new data could be treated as a repeated call from a previous run.

Chapter 8. Security

Security is a fundamental requirement for all financial institutions. When discussing the security of message transfers, a number of different aspects must be considered:

- Encryption of transferred information
- Authentication of transferred information.

These requirements are supported by MERVA Connection/6000.

Encryption of Transferred Information

Using MERVA Connection/6000 you can encrypt all information that is exchanged.

You do this by activating user exits. User exits allow you to include your own algorithm or even other products that support encryption and decryption routines.

There are two user exits:

- ENM4ExitEncrypt for encryption
- ENM4ExitDecrypt for decryption.

See “User Exit Interfaces” for more information on how to implement these routines.

Authentication of Transferred Information

Using MERVA Connection/6000 you can generate an authentication key covering all exchanged information. You do this by activating user exits. User exits allow you to include your own algorithm or even other products that support authentication routines.

There are two user exits:

- ENM4ExitMacGen for MAC generation
- ENM4ExitMacVerify for MAC verification.

See “User Exit Interfaces” for more detailed information on how to implement these routines.

User Exit Interfaces

The following introduces the user exit interfaces of MERVA Connection/6000.

Introduction

There is a fundamental difference between an API call and a user exit:

- For an API call, you write a program that calls the API routine provided by MERVA Connection/6000.
- A user exit is a routine written by you and called by MERVA Connection/6000. The user exit routines must contain the declaration for the function name and formal parameter list, as described in the following.

User Exit Points

Figure 11 shows what happens when an API function is called by an API program on the RISC System/6000. You can see who is calling a user exit at which processing step. In the figure, the following abbreviations are used for the user exits:

ENCRYP ENM4ExitEncrypt

DECRYP ENM4ExitDecrypt

MACGEN ENM4ExitMacGen

MACVFY ENM4ExitMacVerify

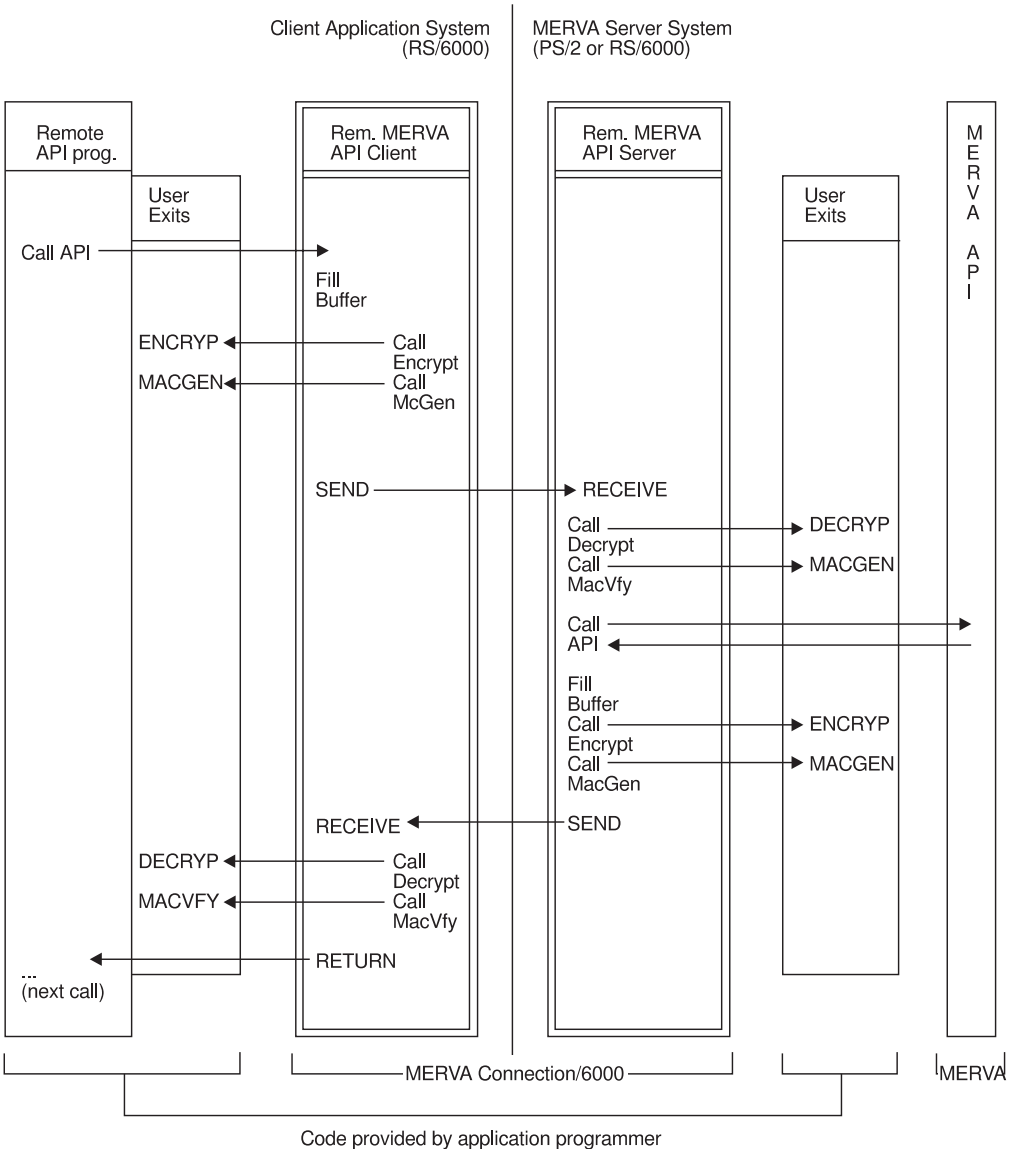

Figure 11. User Exit Points

User Exit Interfaces in C Language

The data types used in these routines can be different, depending on whether they are implemented on the PS/2 or the RISC System/6000. See the coded samples (“Appendix B. Sample Security User Exits” on page 73) for more information.

User Exit for Encryption

C Definition

```
unsigned short ENM4ExitEncrypt ( unsigned char* pucApplId,
                                 unsigned char* pucBuffer,
                                 unsigned short usBufferLen );
```

Purpose of the User Exit Routine

Encrypts the passed data buffer.

Parameter Description

The following parameters are required:

- **pucApplId**(unsigned char*) - input

  Address of a null-terminated string with a maximum length of 8. The string contains the application identifier that you passed as a parameter of the API call ENMStartRAPI. You can use this string to provide different encryption keys for different partner connections, or decide for which connections or for which API programs the information is to be encrypted.
- **pucBuffer**(unsigned char*) - input/output

  Address of the data buffer to be encrypted.
- **usBufferLen**(unsigned short) - input

  Length of the data buffer to be encrypted.

User Exit for Decryption

C Definition

```
unsigned short ENM4ExitDecrypt ( unsigned char* pucApplId,
                                 unsigned char* pucBuffer,
                                 unsigned short usBufferLen );
```

Purpose of the User Exit Routine

Decrypts the passed data buffer.

Parameter Description

The following parameters are required:

- **pucApplId**(unsigned char*) - input

Address of a null-terminated string with a maximum length of 8. The string contains the application identifier that you passed as a parameter of the API call ENMStartRAPI. You can use this string to provide different decryption keys for different partner connections, or to decide for which connections or for which API programs the information is to be decrypted.

- **pucBuffer**(unsigned char*) - input, output

  Address of the data buffer to be decrypted.
- **usBufferLen**(unsigned short) - input

  Length of the data buffer to be decrypted.

User Exit for MAC Generation

C Definition

```
unsigned short ENM4ExitMacGen  ( unsigned char* pucApplId,
                                 unsigned char* pucBuffer,
                                 unsigned short usBufferLen,
                                 unsigned char* pucMacBuffer);
```

Purpose of the User Exit Routine

Generates a MAC (Message Authentication Code) for the passed data buffer.

Parameter Description

The following parameters are required:

- **pucApplId**(unsigned char*) - input

  Address of a null-terminated string with a maximum length of 8. The string contains the application identifier you passed as a parameter of the API call ENMStartRAPI. You can use this string to provide different MAC generation algorithms for different partner connections, or to decide for which connections or for which API programs a MAC shall be generated.
- **pucBuffer**(unsigned char*) - input

  Address of the data buffer for which to generate a MAC.
- **usBufferLen**(unsigned short) - input

  Length of the data buffer for which to generate a MAC.
- **pucMacBuffer**(unsigned char*) - output

  Address of the area to copy the generated MAC to. The address can be up to 32 bytes in length.

User Exit for MAC Verification

C Definition

```
unsigned short ENM4ExitMacVerify  (unsigned char* pucApplId,
                                   unsigned char* pucBuffer,
                                   unsigned short usBufferLen,
                                   unsigned char  pucMacBuffer);
```

Purpose of the User Exit Routine

Generates a MAC for the passed data buffer and compares it with the passed MAC. Set the return code to 0 if the MAC matches, and otherwise to 1.

Parameter Description

The following parameters are required:

- **pucApplId**(unsigned char*) - input

  Address of a null-terminated string with a maximum length of 8. The string contains the application identifier you passed as a parameter of the API call ENMStartRAPI. You can use this string to provide different MAC verification algorithms for different partner connections, or to decide on which connections or for which API programs a MAC is to be verified.
- **pucBuffer**(unsigned char*) - input

  Address of the data buffer for which to generate a MACand for which the passed MAC has been generated on the partner side.
- **usBufferLen**(unsigned short) - input

  Length of the data buffer for which to generate a MAC.
- **pucMacBuffer**(unsigned char*) - input

  Address of the area holding the MAC key that has been received from the partner. The address can be up to 32 bytes in length.

ENM4ExitMacVerify

Chapter 9. Building API Programs

This chapter describes how to compile MERVA Connection/6000 programs in the C programming language.

Compiling Your Own Program

To generate your API program on the RISC System/6000, issue the following commands:

- **cc -o <name> <name>.o -lcpic -lsxit -lrapi**
- **xlc <name>.c -c -o<name>.o -qenum=small**

Compiling the Sample Programs

To generate the executable files for the delivered sample programs, copy all files from the directory /usr/lpp/enm6rapi/samples (see the following list) to a directory of your choice.

List of Sample Files

enm6rsmp.mak
Make file to generate sample API programs

enm6sgen.mak
Make file to generate sample user exit libraries

enmrapi.exp Export file containing the Remote API user program interface function names

enmsxit.exp Export file containing the Remote API user exit function names

smp6le1.c Sample program to send and receive messages to and from MERVA

smp6le2.c Sample program to receive messages from MERVA, uses alarms

smp6le2s.c Sample program to stop SMP6LE2 program

smp6le3.c Sample program to send telex messages to MERVA

smp6le4.c Sample program to verify correct installation

enmra1.prf File containing first sample profile

enmra2.prf File containing second sample profile

enmssec.c File containing sample security user exit routines

enmsnil.c File containing empty security user exit routines.

Run enm6rsmp.mak with the following command:

make -f enm6rsmp.mak

Chapter 10. Replacing Security User Exits

This chapter describes how you can replace the provided security user exits by generating and activating your own security user exits on the RAPI client and server systems.

Security User Exits

Two sets of sample security user exits are provided (see “User Exit Interfaces” on page 47):

enmssec These routines contain sample code for encryption and authentication. They show how to access the variables of the formal parameter list in the function call but do not provide genuine security. The provided code in the RAPI client is the shared library libssec.a. In the RAPI server, it is the the shared library libenmssec.a (MERVA AIX) or the dynamic link library enmssec.dll (MERVA OS/2).

enmsnil These routines do not contain any code. Use this file if no encryption or authentication is desired. The provided code in the RAPI client is the shared library libsnil.a. In the RAPI server, it is the shared library libenmsnil.a (MERVA AIX) or the dynamic link library enmsnil.dll (MERVA OS/2).

In the RAPI client the shared library containing the user exits must have the name **libsxit.a**. The shipped version of libsxit.a is a copy of the sample library libsnil.a.

In the RAPI server (MERVA AIX) the shared library containing the user exits must have the name **libenmsxit.a**. The shipped version of libenmsxit.a is a copy of the sample library libenmsnil.a.

In the RAPI server (MERVA OS/2) the dynamic link library containing the user exits must have the name **enmsxit.dll**. The shipped version of enmsxit.dll is a copy of the sample library enmsnil.dll.

If you want to use the second set (enmssec) of sample user exit routines, copy the following files:

- libssec.a to libsxit.a in the RAPI client
- libenmssec.a to libenmsxit.a in the RAPI server (MERVA AIX)
- enmssec.dll to enmsxit.dll in the RAPI server (MERVA OS/2)

Generating and Activating Security User Exits

Security user exits must be generated and activated in the Remote API client system and in the MERVA server system.

Security User Exits on the RAPI Client System

On the RAPI client system the user exit routines must be placed in shared libraries.

To replace the sample user exits by your own routines, use enmssec.c as a skeleton. Generate a shared library to replace /usr/lib/libsxit.a. For example:

```
xlc enmssec.c -c -o enmssec.o -qenum=small
cc -o enmsxit.o enmssec.o -bE:enmsxit.exp -bM:SRE -e_nostart -O
ar -vq libssec.a enmsxit.o
```

The files that you need for the generation reside in the directory /usr/lpp/enm6rapi/samples. You can also use **enm6rgen.mak** which generates the sample user exit libraries as a sample. **emnsxit.h** must be in the local directory when making the library.

Replace /usr/lib/libsxit.a with your new library using the following command:

cp libssec.a /usr/lib/libsxit.a

Security User Exits on the MERVA Server System

Security User Exits must be generated and activated differently in a MERVA OS/2 and a MERVA AIX server system.

Security User Exits for MERVA OS/2

In the MERVA OS/2 environment the user exit routines must be placed in DLLs.

If you want to replace the sample user exits with your own routines, use enmssec.c as a skeleton. The following file generates a new enmssec.dll from enmssec.c:

```
#-------------------------------------------------------------------------
#  ENMSSEC.MAK  - Make file to generate a DLL with Security User Exits
#-------------------------------------------------------------------------
#-------------------------------------------------------------------------
#  Compile-Options:
#  /C+     Compile, do not link
#  /Gd-    Static linking of the runtime library
#  /Sp1    Structure alignment set to 1-byte boundaries to be compatible
#          with the 16-bit code ( /ZP option ) of MERVA/2
#  /Se     Allow all C Set/2 language extensions except migration
#  /Ss+    Allow use of double slashes (//) for comments
#  /Re     Generate executable code for C Set/2 runtime environment
#  /Gm+    Use the multithread version of the libraries
#  /Kb+    Produce basic diagnostic messages
#  /Fo+    Create an object file
#  /Ti+    Generate debugger information
#  /Ge-    Build a .DLL file
#
#  /DOS2   enable 'define INCL_BASE', 'include <OS2.H>' - see ENMSSEC.C
#
#  Link-Options:
#  /NOE[XTDICTIONARY]       Ignore Extended Dictionary
#  /NOD[EFAULTLIBRARYSEARCH] Ignore Default Libraries
#  /NOI[GNORECASE]          Case sensitive
#  /CO[DEVIEW]              Include symbolic debugging information
#  /BATCH                   Return error, if input file name is missing
#  /A[LIGNMENT]             Alignment Factor in the executable, 512 is default
#  /E[XEPACK]               Packing Executable Files
#  /STACK                   Stack size
#-------------------------------------------------------------------------

C_OPT= /C+ /Gd- /Sp1 /Se /Ss+ /Re /Gm+ /Kb+ /Fo+ /Ti+ /Ge- /DOS2
L_OPT= /NOE /NOD /NOI /CO /BATCH /A:512 /E /STACK:16384

#-------------------------------------------------------------------------
#  Objects which are to be generated in this make file
#-------------------------------------------------------------------------
ALL:  ENMSSEC.DLL ENMSXIT.LIB

#-------------------------------------------------------------------------
#  Link ENMSSEC.DLL
#-------------------------------------------------------------------------
ENMSSEC.DLL: ENMSSEC.OBJ ENMSXIT.DEF
             -LINK386 $(L_OPT) ENMSSEC.OBJ,ENMSSEC.DLL,ENMSSEC.MAP,\
                  OS2386.LIB + DDE4MBS.LIB,ENMSXIT.DEF >>ENMSSEC.LOG;2>&1

#-------------------------------------------------------------------------
#  Compile ENMSSEC
#-------------------------------------------------------------------------
ENMSSEC.OBJ: ENMSSEC.C  ENMSXIT.H
             icc $(C_OPT) ENMSSEC.C >>ENMSSEC.LOG;2>&1

#-------------------------------------------------------------------------
#  Generate LIB file for exit DLL
#-------------------------------------------------------------------------
ENMSXIT.LIB: ENMSXIT.DEF
             IMPLIB ENMSXIT.LIB ENMSXIT.DEF
```

Figure 12. Make File to Generate a DLL

Use the following command to create the new file: **make /f enmssec.mak**

Error messages are written to the file enmssec.log. Copy the newly generated enmssec.dll to enmsxit.dll.

If your source file name is different from enmssec.c, replace every occurrence of enmssec within the make file enmssec.mak with your program name.

Security User Exits for MERVA AIX

The sample security user exits can be accessed by the MERVA AIX Remote API server if you copy the library libenmssec.a to the library libenmsxit.a. If you want to replace the sample user exits by your own routines, use the enmssec.c as a skeleton. The file can be retrieved from directory */usr/lpp/merva/samples*. The file enmssec.mak generates a new library libenmssec.a from the source file enmssec.c.

Use the following command:

make -f enmssec.mak all

Replace /usr/lpp/merva/lib/libenmsxit.a with your new library using the following command:

cp libenmssec.a /usr/lpp/merva/lib/libenmsxit.a

Chapter 11. Diagnosis Information

This chapter describes the diagnosis information that is written to log files on the RAPI client system and on the RAPI server system (MERVA OS/2 or MERVA AIX).

Log Files on the RAPI Client System

Two logs are written. You can choose their names and directories by setting them in the MERVA Connection/6000 profile (see “Customizing MERVA Connection/6000” on page 7).

Each message written to the logs consists of two parts, the message header and the message body, as shown in Figure 13.

```
*  19970603170116ENMRAPI ENMTrace
ENM9151: Trace: ON

*  19970603170116ENMRAPI ENMAttach
ENM9153: API function ENMAttach called.
          Parameters:
          Uid: SAMPLE
          Pwd: SAMPLE1
          Fid: API

*  19970603170117ENMRAPI ENMQueryQueue
ENM9153: API function ENMQueryQueue called.
          Parameters:
          Qn : API_IN

*  19970603170117ENMRAPI ENMQueryQueue
ENM9153: API function ENMQueryQueue called.
          Parameters:
          Qn : API_OUT

*  19970603170117ENMRAPI ENMDetach
ENM9153: API function ENMDetach called.
          Parameters: no input parms

*  19970603170118ENMRAPI ENMEndRAPI
ENM9153: API function ENMEndRAPI called.
          Parameters: no input parms
```

Figure 13. Example of Diagnosis Log with API Trace Entries

Diagnosis Log

The diagnosis log provides you with:

- Error messages that help you recover from errors when using the API calls or errors concerning the communication with the MERVA system.
- Trace information when the API Trace is switched on with the call ENMTrace (see *MERVA AIX Application Programming*).

Programmer's Log

The programmer's log is a general debugging tool. It contains all entries of the diagnosis log and additional more detailed information to be analyzed by your IBM representative.

The layout of the header is as follows:

Date The date is in the form YYYYMMDD, where YYYY is the year, MM is the month, and DD is the day.

Time The time is in the form HHMMSS, where HH is the hour, MM are the minutes, and SS are the seconds.

Module name The module name is an 8-character code identifying the module the message originated from.

Function name
The function name is a 15-character code identifying the function the message originated from.

The layout of the message is as follows:

Message The variable-length message to be recorded. See *MERVA Messages and Codes* for the meaning of the messages.

Note: Logging entries are appended to the existing files. If you want that MERVA Connection/6000 creates new log files, delete the old log files.

Log Files on the RAPI Server System (MERVA AIX)

Diagnosis information concerning the Remote MERVA API Server program is provided by the MERVA AIX log files. Error and trace information is written to the diagnosis log. IBM service information is written to the programmer's trace log. You can list or browse the diagnosis log file using the Display Diagnosis Log function in the MERVA AIX menu program.

The log files are located in the MERVA AIX instance logging directory as selected in the "Create MERVA AIX Instance" step described in the *MERVA AIX Installation and Customization Guide*.

Log Files on the RAPI Server System (MERVA OS/2)

Diagnosis information concerning the Remote MERVA API Server program is provided by the MERVA OS/2 log files. Error and trace information is written to the diagnosis log. IBM service information is written to the programmer's log. You can list or browse the diagnosis log file using the Display/Print Diagnosis Log (DPD) function of MERVA OS/2.

The log files are located on the disk and directory *x*:\MERVA2\, where *x* is the drive on which MERVA OS/2 is installed. See the *MERVA OS/2 V3 Diagnosis Guide* for further information.

Appendix A. Sample SNA Definitions

MERVA Connection/6000 uses LU 6.2 sessions for the communication between the Remote MERVA API Client and Server in the SNA Data Communication environment. There are many ways how the data communication subsystems in the client and server systems can be customized to bind the required sessions.

This appendix provides one example for these many ways. It is based on a customization example shown in the *SNA Server for AIX User's Guide*. (Configuring an APPN Network of Two Nodes), and uses the naming conventions of the latter example.

The MERVA Connection/6000 SNA customization sample defines an APPN network of two nodes in a token ring. The name of the sample network is **APPN1**. The MERVA Server System is defined as an APPN Network Node (NN), and the Client Application System is defined as an APPN End Node (EN). A second Client Application System can be easily added to this sample network.

The naming conventions for the SNA resources in the sample network node (MERVA Server System) are:

NNA The network node name. A second network node in this APPN network would be named NNB.

CPA The name of the Control Point in NNA.

LUA The name of an independent LU 6.2 in NNA.

The sample token ring address of NNA is **10005aa88fe0**.

The naming conventions for the SNA resources in the sample end node (Client Application System) are:

EN1 The end node name. A second end node in this APPN network would be named EN2.

CP1 The name of the Control Point in EN1.

TR1 The name of the Token Ring Link Station in EN1 that provides the link to the network node server (NNA).

LU1 The name of an independent LU 6.2 in EN1.

The sample customizations of an APPN End Node (MERVA Connection/6000 Client Application System), an APPN Network Node (AIX Server System), and an APPN Network Node (OS/2 Server System) are provided in the following sections.

Customizing an APPN End Node (AIX)

A detailed description how to configure an end node in a two-node APPN network can be found in the *SNA Server for AIX User's Guide*. In the following it is assumed that you are familiar with this description.

Initial Node Setup

The initial node setup can be performed by entering the applicable parameters in the SNA Server for AIX Initial Node Setup menu or by entering the following command:

mk_qcinit -w APPN1 -d CP1 -y token_ring -q TR1 -N yes -s 10005aa88fe0

The w-flag defines the APPN network name (APPN1). The d-flag defines the control point name (CP1). The y-flag defines the data link type (token_ring). The q-flag defines the name of the SNA DLC profile and the name of the Link Station profile (TR1). The N-flag specifies whether the link station is a calling link station (yes).

The s-flag specifies the sample link address (10005aa88fe0). It is the token ring address of the APPN network node server. The sample link address must be replaced by the actual address of the network node server. For the correct value to enter, see your system administrator.

If the APPN Network Node executes in the AIX environment, you can ask for its token ring address using the following AIX command:

/usr/sbin/lscfg/ -l tok0 -v

This command must be entered in the network node.

The initial node setup modifies the APPN Control Point Profile node_cp, and creates the SNA DLC profile TR1 and the Token Ring Link Station profile TR1.

Check and Modify the Initial Node Setup

You may wish to check the initial node setup for the APPN end node by comparing the modified and generated profiles with following figures. A small number of parameters in the SNA DLC profile TR1 and in the Token Ring Link Station profile TR1 are modified in this example.

Control Point Profile

To display the sample SNA Control Point Profile call smitty sna and select:

1. **Configure SNA Profiles**
2. **Advanced Configuration**
3. **Control Point**
4. **Change/Show a Profile**

The sample SNA Control Point profile for node EN1 is shown in Figure 14 on page 63.

```
                    Change/Show Control Point Profile

Type or select values in entry fields.
Press Enter AFTER making all desired changes.

                                                      [Entry Fields]
* Profile name                                        node_cp
  XID node ID                                        [*]
  Network name                                       [APPN1]
  Control Point (CP) name                            [CP1]
  Control Point alias                                [CP1]
  Control Point type                                  appn_end_node            +
  ....
  ....
```

Figure 14. SNA Control Point in Node EN1

The Control Point profile modified by the initial node setup is not modified by the MERVA Connection/6000 configuration sample.

Token Ring SNA DLC Profile

To display the sample Token Ring SNA DLC Profile call smitty sna and select:

1. **Configure SNA Profiles**
2. **Advanced Configuration**
3. **Links**
4. **Token Ring**
5. **Token Ring SNA DLC**
6. **Change/Show a Profile**

Part of the sample Token Ring SNA DLC profile TR1 for node EN1 is shown in Figure 15 on page 64.

```
                    Change/Show Token Ring SNA DLC Profile

Type or select values in entry fields.
Press Enter AFTER making all desired changes.

[TOP]                                                 [Entry Fields]
  Current profile name                                TR1
  New profile name                                   []
  Data link device name                              [tok0]                    +
  ....
  ....
  ....
  Dynamic link stations supported?                    yes                      +

  Link Recovery Parameters
     Retry interval (1-10000 seconds)                [30]                      #
     Retry limit (0 or 1-500 attempts)               [0]                       #


  ....
  ....
  ....

F1=Help          F2=Refresh         F3=Cancel            F4=List
F5=Reset         F6=Command         F7=Edit              F8=Image
F9=Shell         F10=Exit           Enter=Do
```

Figure 15. Token Ring SNA DLC Profile TR1 in Node EN1

The modified Link Recovery Parameters provide for faster and permanent automatic recovery of the link to the APPN network server when the network server has been temporarily inactive.

Token Ring Link Station Profile

To display the sample Token Ring Link Station Profile call smitty sna and select:

1. **Configure SNA Profiles**
2. **Advanced Configuration**
3. **Links**
4. **Token Ring**
5. **Token Ring Link Station**
6. **Change/Show a Profile**

The sample Token Ring SNA Link Station profile **TR1** for node EN1 is shown in Figure 16 on page 65.

```
                  Change/Show Token Ring Link Station Profile

Type or select values in entry fields.
Press Enter AFTER making all desired changes.

[TOP]                                              [Entry Fields]
  Current profile name                             TR1
  New profile name                                 []
  Use Control Point's XID node ID?                  yes                      +
     If no, XID node ID                            [*]
* SNA DLC Profile name                             [TR1]

:
:

  Adjacent Node Address Parameters
     Access routing                                 link_address             +
     If link_name, Rmote link name                 []
     If link_address,
        Remote link address                        [10005aa88fe0]            X
        Remote SAP address                         [04]                      X

  Adjacent Node Identification Parameters
     Verify adjacent node?                          yes                      +
     Network ID of adjacent node                   [APPN1]
     CP name of adjacent node                      [CPA]
     XID node ID of adjacent node (LEN node only)  [*]
     Node type of adjacent node                     learn                    +

  Link Activation Parameters
     Solicit SSCP sessions?                         yes                      +
     Initiate call when link station is active?     yes                      +
     Activate link station at SNA startup?          yes                      +
     Activate on demand?                            no                       +
     CP-CP sessions supported?                      yes                      +
     If yes,
        Adjacent network node preferred server?     yes                      +
     Partner required to support CP-CP sessions?    no                       +
     Initial TG number (0-20)                      [0]                       #

  Restart Parameters
     Restart on activation?                         no                       +
     Restart on normal deactivation?                yes                      +
     Restart on abnormal deactivation?              yes                      +
```

Figure 16. Token Ring Link Station Profile TR1 in Node EN1

The modified Restart Parameters provide for an automatic recovery of the link to the APPN network server when the network server has been temporarily inactive.

Defining Additional Resources

A Local LU, an APPC Session Mode, and a Side Information Profile are the additional resources required for the communication with the MERVA Server System.

Local LU Profile

To add the sample Local LU Profile call smitty sna and select:

1. **Configure SNA Profiles**
2. **Advanced Configuration**
3. **Sessions**
4. **LU 6.2**

5. **LU 6.2 Local LU**
6. **Add a Profile**

The sample LU 6.2 Local LU profile for node EN1 is shown in Figure 17.

```
                        Add LU 6.2 Local LU Profile

Type or select values in entry fields.
Press Enter AFTER making all desired changes.

                                                      [Entry Fields]
* Profile name                                       [LU1]
  Local LU name                                      [LU1]
  Local LU alias                                     [LU1]
  Local LU is dependent?                              no                      +
     If yes,
  ....
  ....
  Conversation Security Access List Profile name     []
  Recovery resource manager (RRM) enabled?            no                      +


F1=Help          F2=Refresh         F3=Cancel            F4=List
F5=Reset         F6=Command         F7=Edit              F8=Image
F9=Shell         F10=Exit           Enter=Do
```

Figure 17. LU 6.2 Local LU Profile in Node EN1

Mode Profile

To add the sample Mode Profile call smitty sna and select:

1. **Configure SNA Profiles**
2. **Advanced Configuration**
3. **Sessions**
4. **LU 6.2**
5. **LU 6.2 Mode**
6. **Add a Profile**

The sample LU 6.2 Mode profile for application sessions (APPCLU62) is shown in Figure 18 on page 67.

```
                          Add LU 6.2 Mode Profile

Type or select values in entry fields.
Press Enter AFTER making all desired changes.

                                                    [Entry Fields]
* Profile name                                     [APPCLU62]
  Mode name                                        [APPCLU62]
  Maximum number of sessions (1-5000)              [2]                          #
  Minimum contention winners (0-5000)              [1]                          #
  Minimum contention losers (0-5000)               [0]                          #
  Auto activate limit (0-500)                      [1]                          #
  Upper bound for adaptive receive pacing window   [16]                         #
  Receive pacing window (0-63)                     [7]                          #
  Maximum RU size (128,...,32768: multiples of 32) [1024]                       #
  Minimum RU size (128,...,32768: multiples of 32) [256]                        #
  Class of Service (COS) name                      [#CONNECT]


F1=Help          F2=Refresh         F3=Cancel            F4=List
F5=Reset         F6=Command         F7=Edit              F8=Image
F9=Shell         F10=Exit           Enter=Do
```

Figure 18. LU 6.2 Mode Profile in Node EN1

The sample Mode profile in Figure 18 defines an SNA logon mode that can be used for APPC sessions to all kinds of partner systems.

LU 6.2 Side Information Profile

To add the sample Side Information Profile call smitty sna and select:

1. **Configure SNA Profiles**
2. **Advanced Configuration**
3. **Sessions**
4. **LU 6.2**
5. **LU 6.2 Side Information**
6. **Add a Profile**

The sample LU 6.2 Side Information profile for the Remote MERVA API Server is shown in Figure 19.

```
                     Add LU 6.2 Side Information Profile

Type or select values in entry fields.
Press Enter AFTER making all desired changes.

                                                    [Entry Fields]
* Profile name                                     [MERVA]
  Local LU or Control Point alias                  [LU1]                      +
  Provide only one of the following:
     Partner LU alias                              []                         +
     Fully qualified partner LU name               [APPN1.LUA]
  Mode name                                        [APPCLU62]                 +
  Remote transaction program name (RTPN)           [ENMRAS]
  RTPN in hexadecimal?                              no                        +
```

Figure 19. LU 6.2 Side Information Profile ENMRAS

Customizing an APPN Network Node (AIX)

A detailed description how to configure a network node in a two-node APPN network can be found in the *SNA Server for AIX User's Guide*. In the following it is assumed that you are familiar with this description.

Initial Node Setup

The initial node setup can be performed by entering the applicable parameters in the SNA Server for AIX Initial Node Setup menu or by entering the following command:

mk_qcinit -w APPN1 -d CPA -y token_ring -N yes -t appn_network_node

The w-flag defines the APPN network name (APPN1). The d-flag defines the control point name (CPA). The y-flag defines the data link type (token_ring). The N-flag specifies whether the link station is a calling link station (yes). The t-flag specifies the APPN network node type (appn_network_node).

The initial node setup modifies the APPN Control Point Profile node_cp and creates the SNA DLC profile tok0.00001. A Token Ring Link Station profile is not generated. The sample Network Node uses only dynamically generated link stations.

Check and Modify the Initial Node Setup

You may wish to check the initial node setup for the APPN end node by comparing the modified and generated profiles with following figures. The profiles modified or generated by the initial node setup are not modified in this example.

Control Point Profile

To display the sample SNA Control Point Profile call smitty sna and select:

1. **Configure SNA Profiles**
2. **Advanced Configuration**
3. **Control Point**
4. **Change/Show a Profile**

The sample SNA Control Point profile for node NNA is shown in Figure 20 on page 69.

```
                    Change/Show Control Point Profile

Type or select values in entry fields.
Press Enter AFTER making all desired changes.

                                                      [Entry Fields]
* Profile name                                        node_cp
  XID node ID                                        [*]
  Network name                                       [APPN1]
  Control Point (CP) name                            [CPA]
  Control Point alias                                [CPA]
  Control Point type                                  appn_network_node        +
  ....
  ....
  ....
```

Figure 20. SNA Control Point in Node NNA

The Control Point profile modified by the initial node setup is not modified by the MERVA Connection/6000 configuration sample.

Token Ring SNA DLC Profile

To display the sample Token Ring SNA DLC Profile call smitty sna and select:

1. **Configure SNA Profiles**
2. **Advanced Configuration**
3. **Links**
4. **Token Ring**
5. **Token Ring SNA DLC**
6. **Change/Show a Profile**

The sample Token Ring SNA DLC profile for node NNA is shown in Figure 21 on page 70.

```
                    Change/Show Token Ring SNA DLC Profile

Type or select values in entry fields.
Press Enter AFTER making all desired changes.

[TOP]                                                 [Entry Fields]
  Current profile name                               tok0.00001
  New profile name                                   []
  Data link device name                              [tok0]                    +
  ....
  ....
  ....
  Dynamic link stations supported?                    yes                      +

  Link Recovery Parameters
     Retry interval (1-10000 seconds)                [60]                      #
     Retry limit (0 or 1-500 attempts)               [20]                      #


  ....
  ....
  ....

F1=Help          F2=Refresh         F3=Cancel            F4=List
F5=Reset         F6=Command         F7=Edit              F8=Image
F9=Shell         F10=Exit           Enter=Do
```

Figure 21. Token Ring SNA DLC Profile in Node NNA

The Token Ring SNA DLC profile generated by the initial node setup is not modified by the MERVA Connection/6000 configuration sample.

Defining Additional Resources

A Local LU, a Partner LU, an APPC Session Mode, and a TPN Profile are the additional resources required for the communication with the Client Application System.

Local LU Profile

To add the sample Local LU Profile call smitty sna and select:

1. **Configure SNA Profiles**
2. **Advanced Configuration**
3. **Sessions**
4. **LU 6.2**
5. **LU 6.2 Local LU**
6. **Add a Profile**

The sample LU 6.2 Local LU profile for node NNA is shown in Figure 22 on page 71.

```
                        Add LU 6.2 Local LU Profile

Type or select values in entry fields.
Press Enter AFTER making all desired changes.

                                                      [Entry Fields]
* Profile name                                       [LUA]
  Local LU name                                      [LUA]
  Local LU alias                                     [LUA]
  Local LU is dependent?                              no                      +
     If yes,
  ....
  ....
  Conversation Security Access List Profile name     []
  Recovery resource manager (RRM) enabled?            no                      +



F1=Help          F2=Refresh         F3=Cancel            F4=List
F5=Reset         F6=Command         F7=Edit              F8=Image
F9=Shell         F10=Exit           Enter=Do
```

Figure 22. LU 6.2 Local LU Profile in Node NNA

Mode Profile

The sample LU 6.2 Mode profile for application sessions (APPCLU62) is shown in Figure 18 on page 67. It is the same in nodes EN1 and NNA.

TP Name Profile

The Remote MERVA API Server TP must be defined in an LU 6.2 TPN profile. The sample TP name is ENMRAS. A sample LU 6.2 TPN profile for ENMRAS is shown in Figure 4 on page 15.

Customizing an APPN Network Node (OS/2)

An APPN Network Node can only be customized with Communications Manager/2 or Communication Server. Personal Communications is capable of APPN functions, but only as an End Node.

Start the **Communication Server Setup** program, select “Setup...” and a configuration file. Then double-cklick on “CPI Comunications” and in the following profile list choose “SNA local node characteristics”.

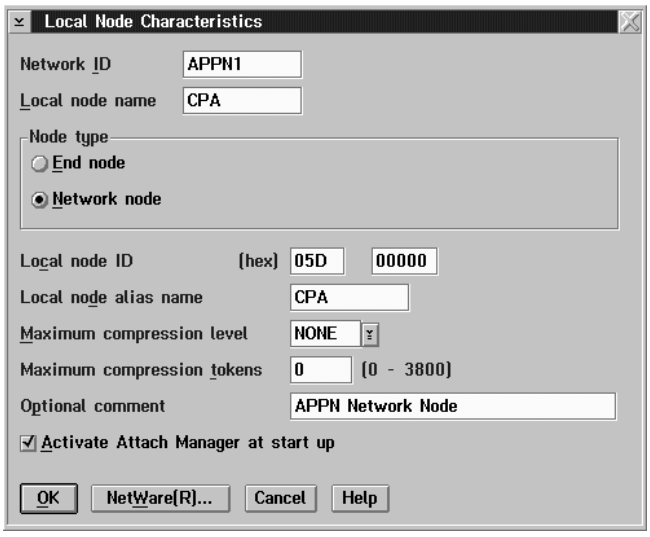

Figure 23. SNA Local Node Characteristics for a Network Node

Select **Network Node** as node type. The field **Local node ID** may be left blank.

This is all you have to do if you want to customize an APPN Network Node with Communication Server.

Appendix B. Sample Security User Exits

This appendix contains listings of sample security user exits that you can use.

Module ENMSNIL - Empty Functions

This program is integrated into MERVA Connection/6000 in the supplied version. No actions are taken in the functions. This means that data transferred between the MERVA system and the remote application system is not encrypted and no authentication key is built or transferred. You can use this program as a skeleton for your code.

```
/*--------------------------------------------------------------------*\
| ENMSNIL.C                                                            |
\*--------------------------------------------------------------------*/
#if defined(OS2)
#define  INCL_BASE
#include <OS2.H>
#endif

#include "enmsxit.h"

#ifndef __32BIT__
   #define APIENTRY16 APIENTRY
   #define PUCHAR16   PUCHAR
#endif

USHORT APIENTRY16 ENM4ExitMacGen (PUCHAR16 pucApplId,
                                  PUCHAR16 pucBuffer,
                                  USHORT   usBufferLen,
                                  PUCHAR16 pucMacBuffer)
{
   return(0);
}

USHORT APIENTRY16 ENM4ExitMacVerify (PUCHAR16 pucApplId,
                                     PUCHAR16 pucBuffer,
                                     USHORT   usBufferLen,
                                     PUCHAR16 pucMacBuffer)
{
   return(0);
}

USHORT APIENTRY16 ENM4ExitEncrypt (PUCHAR16 pucApplId,
                                   PUCHAR16 pucBuffer,
                                   USHORT   usBufferLen )
{
   return(0);
}

USHORT APIENTRY16 ENM4ExitDecrypt (PUCHAR16 pucApplId,
                                   PUCHAR16 pucBuffer,
                                   USHORT   usBufferLen )
{
   return(0);
}
```

Figure 24. Sample Security User Exit ENMSNIL.C

Module ENMSSEC - Sample Functions

This module is supplied as an example for coding security functions. Simple encryption and authentication routines are included. However, they do not provide genuine security.

```
/*--------------------------------------------------------------------*\
| ENMSSEC.C                                                          |
\*--------------------------------------------------------------------*/
#if defined(OS2)
#define  INCL_BASE
#include <OS2.H>
#endif
#include<string.h>
#include "enmsxit.h"
/* defines that this module can be compiled with Cset/2 and IBM C/2 */
#ifndef __32BIT__
   #define APIENTRY16 APIENTRY
   #define PUCHAR16   PUCHAR
#endif
unsigned char Enm36Table[36]= {'\x00', '\x01', '\x02', '\x03',
                               '\x04', '\x05', '\x06', '\x07',
                               '\x08', '\x09', '\x0A', '\x0B',
                               '\x0C', '\x1D', '\x1E', '\x1F',
                               '\x10', '\x11', '\x12', '\x13',
                               '\x14', '\x15', '\x16', '\x17',
                               '\x18', '\x19', '\x1A', '\x1B',
                               '\x1C', '\x1D', '\x1E', '\x1F',
                               '\x20', '\x21', '\x22', '\x23' };
#define ENM_MAX_BASE              36
#define ENM_FILL_CHAR             0

unsigned short EnmBasestr(unsigned short base,
                          unsigned long  num,
                          unsigned char* basestring,
                          unsigned short max_len)
{
   unsigned long  count=0,reminder=0;
   short          position;
   unsigned long  number;
   number = num;
   position = max_len-1;
   memset (basestring, ENM_FILL_CHAR, max_len);
   basestring[position]=0;

   if (base > ENM_MAX_BASE) return(1);
   do {
      if (--position < 0) return(1);
      reminder = number % (unsigned long)base;
      count = number / (unsigned long)base;
      if (!count) {
          basestring[position++]=Enm36Table[reminder];
          break;
      }
      basestring[position]=Enm36Table[reminder];
      number = count;
   } while (1);
   return(0);
}
USHORT APIENTRY16 ENM4ExitMacGen (PUCHAR16 pucApplId,
                                  PUCHAR16 pucBuffer,
                                  USHORT   usBufferLen,
                                  PUCHAR16 pucMacBuffer)
```

Figure 25. Sample Security User Exit ENMSSEC.C (Part 1 of 3)

```
{
   register       i;
   unsigned long  ulAddedByteValues=0;
   unsigned short rc = 0;

   if (!strcmp(pucApplId,"APPLAUTH") || !strcmp(pucApplId,"APPLSECR")) {
      for (i=0;i<usBufferLen;i++) {
         ulAddedByteValues += (unsigned long) pucBuffer[i];
      }
      rc = EnmBasestr(2,
                      ulAddedByteValues,
                      pucMacBuffer,
                      32);
   }
   return(rc);
}

USHORT APIENTRY16 ENM4ExitMacVerify (PUCHAR16 pucApplId,
                                     PUCHAR16 pucBuffer,
                                     USHORT   usBufferLen,
                                     PUCHAR16 pucMacBuffer)
{
   register       i;
   unsigned long  ulAddedByteValues=0;
   unsigned char  ucaCalcMacBuffer[32];
   unsigned short rc = 0;

   if (!strcmp(pucApplId,"APPLAUTH") || !strcmp(pucApplId,"APPLSECR")) {
      for (i=0;i<usBufferLen;i++) {
         ulAddedByteValues += (unsigned long) pucBuffer[i];
      }
      memset (ucaCalcMacBuffer,0,32);

      rc = EnmBasestr(2,
                      ulAddedByteValues,
                      ucaCalcMacBuffer,
                      32);
      if (!rc) rc = memcmp(ucaCalcMacBuffer,pucMacBuffer,32);
   }
   return(rc);
}
USHORT APIENTRY16 ENM4ExitEncrypt (PUCHAR16 pucApplId,
                                   PUCHAR16 pucBuffer,
                                   USHORT   usBufferLen )
{
   register i;

   if (!strcmp(pucApplId,"APPLENCR") || !strcmp(pucApplId,"APPLSECR")) {
      for (i=0;i<usBufferLen;i++) {
         pucBuffer[i] = pucBuffer[i] [ 255; /* negation */
      }
   }
   return(0);
```

Figure 25. Sample Security User Exit ENMSSEC.C (Part 2 of 3)

```
}

USHORT APIENTRY16 ENM4ExitDecrypt (PUCHAR16 pucApplId,
                                   PUCHAR16 pucBuffer,
                                   USHORT   usBufferLen )
{
   register i;

   if (!strcmp(pucApplId,"APPLENCR") || !strcmp(pucApplId,"APPLSECR")) {
      for (i=0;i<usBufferLen;i++) {
         pucBuffer[i] = pucBuffer[i] [ 255; /* negation */
      }
   }
   return(0);
}
```

Figure 25. Sample Security User Exit ENMSSEC.C (Part 3 of 3)

Appendix C. Notices

This information was developed for products and services offered in the U.S.A. IBM may not offer the products, services, or features discussed in this document in other countries. Consult your local IBM representative for information on the products and services currently available in your area. Any reference to an IBM product, program, or service is not intended to state or imply that only that IBM product, program, or service may be used. Any functionally equivalent product, program, or service that does not infringe any IBM intellectual property right may be used instead. However, it is the user's responsibility to evaluate and verify the operation of any non-IBM product, program, or service.

IBM may have patents or pending patent applications covering subject matter described 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:

IBM Director of Licensing
IBM Corporation
North Castle Drive
Armonk, NY 10504-1785
U.S.A.

For license inquiries regarding double-byte (DBCS) information, contact the IBM Intellectual Property Department in your country or send inquiries, in writing, to:

IBM World Trade Asia Corporation
Licensing
2-31 Roppongi 3-chome, Minato-ku
Tokyo 106, Japan

The following paragraph does not apply to the United Kingdom or any other country where such provisions are inconsistent with local law:
INTERNATIONAL BUSINESS MACHINES CORPORATION PROVIDES THIS PUBLICATION "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF NON-INFRINGEMENT, MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE. Some states do not allow disclaimer of express or implied warranties in certain transactions, therefore, this statement may not apply to you.

This information could include technical inaccuracies or typographical errors. Changes are periodically made to the information herein; these changes will be incorporated in new editions of the publication. IBM may make improvements and/or changes in the product(s) and/or the program(s) described in this publication at any time without notice.

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 Deutschland
Informationssysteme GmbH
Department 3982
Pascalstrasse 100
70569 Stuttgart
Germany

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

The licensed program described in this information and all licensed material available for it are provided by IBM under terms of the IBM Customer Agreement or any equivalent agreement between us.

The following paragraph does apply to the US only.

All IBM prices shown are IBM's suggested retail prices, are current and are subject to change without notice. Dealer prices may vary.

This information is for planning purposes only. The information herein is subject to change before the products described become available.

COPYRIGHT LICENSE:

This information contains sample application programs in source language, which illustrates programming techniques on various operating platforms. You may copy, modify, and distribute these sample programs in any form without payment to IBM, for the purposes of developing, using, marketing or distributing application programs conforming to the application programming interface for the operating platform for which the sample programs are written. These examples have not been thoroughly tested under all conditions. IBM, therefore, cannot guarantee or imply reliability, serviceability, or function of these programs. You may copy, modify, and distribute these sample programs in any form without payment to IBM for the purposes of developing, using, marketing, or distributing application programs conforming to IBM's application programming interfaces.

Trademarks

The following terms are trademarks of the IBM Corporation in the United States or other countries, or both:

- ACF/VTAM
- AIX
- AIX/6000
- AS/400
- CICS/ESA
- C/400
- DATABASE 2
- DB2
- IBM
- MERVA
- MVS/ESA
- MVS/SP
- Operating System/2
- OS/2
- OS/400
- Personal System/2
- POWER Architecture
- PS/2

- RACF
- RISC System/6000
- RISC/6000
- RS/6000
- RPG/400
- SAA
- Systems Application Architecture
- VSAM
- VTAM

Workstation (AWS) and Directory Services Application (DSA) are trademarks of S.W.I.F.T., La Hulpe in Belgium.

Pentium is a trademark of Intel Corporation.

UNIX is a registered trademark in the United States and other countries licensed exclusively through X/Open Company Limited.

Microsoft, Windows, and the Windows 95 logo are trademarks or registered trademarks of Microsoft Corporation.

Other company, product, and service names may be trademarks or service marks of others.

Glossary of Terms and Abbreviations

This glossary defines terms and abbreviations as they are used in the MERVA AIX books. If you do not find the terms your are looking for, refer to *Dictionary of Computing*, New York: McGraw-Hill, 1994, the *S.W.I.F.T. User Handbook*, or the *S.W.I.F.T. USE Planning Guide*.

A

AMPDU. Application Message Protocol Data Unit defined in the MERVA Link P1 protocol. It consists of an envelope and the content, which is an ASP-supplied information.

answerback. In telex, the response from the dialed correspondent to the “WHO R U” signal.

AP. Application.

APAR. Authorized Program Analysis Report.

APDU. Application protocol data unit.

API. Application Programming Interface.

APPC. Advanced Program-to-Program Communication based on LU 6.2 protocols.

Application Support (AS). Application Support is the name of the upper sublayer functionality of MERVA Link.

Application Support Layer (ASL). ASL contains the Application Support functionality.

Application Support Process (ASP). The part of MERVA Link that implements the Application Support Layer.

ASCII. American Standard Code for Information Interchange.

ASP. Application Support Process.

ASPDU. Application Support Protocol Data Unit defined in the MERVA Link P2 protocol.

association time-out. The period of time allowed for the establishment of a MERVA Link session with the remote partner before giving up.

authentication. The SWIFT security check used to ensure that a message is not changed during transmission and that it was sent by an authorized sender.

authenticator key. A set of alphanumeric characters used to check the authentication of a message sent via the SWIFT network.

authenticator-key file. This file contains the keys used for authenticating messages. The file contains a record for each correspondent bank.

B

Bank Identifier Code. The SWIFT address of a bank as assigned by SWIFT. See also *SWIFT address*.

BCR. Basic Card Reader.

BIC. Bank Identifier Code. See *SWIFT address*.

bi-directional. A type of bilateral key where the same key is used to authenticate messages sent to and received from a correspondent.

bilateral key. A key, generated inside an SCR, that is used to authenticate financial messages passing between a pair of correspondents. A bilateral key can be *bi-directional* or *uni-directional*.

bilateral key exchange (BKE) service. The SWIFT USE service in which authenticator keys are generated in an SCR and exchanged by means of the SWIFT network, as an alternative to exchanging keys by mail.

BK. Bilateral Key.

BKE. Bilateral Key Exchange.

BK-ID. Bilateral key identifier. The BK-ID has the following format:

- The first character is either B (Bilateral) or M (Manual)
- The second character is the BK type, as defined by SWIFT
- Characters 3 to 8 inclusive are the date
- Characters 9 to 16 inclusive are the key check value.

blacklist. A list of USE items, such as SCRs or CVs, that are no longer valid. For example, a stolen SCR is blacklisted to prevent future use.

branch code. The last 3 digits of the BIC, used to identify a bank.

C

CBT. SWIFT Computer Based Terminal.

certificate. A guarantee issued by SWIFT that the holder of a public key is genuine. You must obtain a certificate for each public key you generate before starting bilateral key exchange.

CID. Central Institution Destination.

control database. Contains all MERVA AIX-specific configuration data, such as routing table information, system configuration data, and user-specific information, such as the user file containing details of MERVA AIX users and their access rights to functions and queues.

correspondent. An institution to which your institution sends messages and from which messages are received.

correspondents database. A database containing the SWIFT address, nickname, and descriptive name and address of each bank with which your bank corresponds. The file is used to store the descriptive names and addresses used in the address expansion process.

country code. A 2-character code that is part of the BIC, used to identify countries.

CRC. Cyclic Redundancy Check.

CS. When CS is used, you can use Communications Server or Personal Communications.

CUG. Closed User Group.

CV. See *certificate*.

CV-ID. Certificate identity. A unique identifier of a certificate made up of the destination, expiration date, and number of the certificate.

D

destination. For SWIFT, the first 8 characters of the SWIFT address, consisting of the bank, country, and location codes.

domain. For MERVA AIX, a set of workstations that share a MERVA installation. The MERVA domain is a part of the MERVA Message Reference Number (MRN).

E

emitting destination. The SWIFT destination that appears on messages sent to SWIFT. You must specify the emitting destination, for example, when sending a message to SWIFT requesting the blacklisting of a card reader.

F

FIN. Financial Application (SWIFT).

four-eyes principle. A banking security concept, whereby changes and approval of changes must always be carried out by two different people.

I

IAM. Interapplication messaging is a MERVA Link message exchange protocol.

ICC. Integrated Circuit Card.

IM-ASPDU. Interapplication messaging application support PDU. It contains an application message and consists of a header and a body.

initiator. The correspondent that begins bilateral key exchanges. See also *responder*.

ISC. Intersystem communication.

ISN. Input sequence number.

ISO. International Organization for Standardization.

K

kernel. A secret value stored on a USER ICC for each LT, used to define access rights to SWIFT applications and to generate session keys. There are eight kernels per USER ICC.

kernel version. A pointer to the kernel currently in use.

key check value. (1) Part of the *BK-ID*. If you encounter problems communicating with your correspondent, check the key check value, which should be identical to your correspondent's. (2) Part of the *secure transmission key (STK)*, used as a check that you have entered the remainder of the STK correctly.

KMA. Key Management Authority.

L

LAK. Login acknowledgment message. This message informs you that you have successfully logged in to the SWIFT network.

LNK. SWIFT login negative acknowledgment message. This message informs you that the login to the SWIFTnetwork has failed.

local LU name. The logical unit name, or workstation identifier, of the local machine.

logging database. Contains all MERVA AIX audit logging data.

logical unit. In SNA, a port through which the user accesses the SNA network.

log in. To start the connection to the SWIFT network.

LSN. Login Sequence Number.

LT. Logical Terminal. The S.W.I.F.T. II equivalent of the TID (Terminal Identifier).

LU name. Logical Unit name.

M

MAC. Message Authentication Code.

master logical terminal. The 9-character code assigned by SWIFT to uniquely identify each terminal attached to the S.W.I.F.T. II network.

MERVA AIX. Message Entry and Routing with Interfaces to Various Applications for AIX.

MERVA domain. See *domain*.

MERVA Link. The component that can be used to interconnect MERVA systems.

message. A string of fields in a predefined form used to provide or request information. See also SWIFT *message*.

message buffer. The part of the queue buffer that holds messages in network format.

message database. Contains all messages created by the user or received by the MERVA AIX system.

message field. A predefined part of a message, identified either by a known offset from the start of a message, or by a delimiter known as a scan pattern.

message header. The leading part of a message that contains the sender and receiver of the message, the message priority, and the type of message.

message integrity. A facility provided by MERVA Link that ensures that, in the case of an interruption during message exchange, duplicates of messages are not sent, and that no messages are lost.

message integrity protocol. A facility used by MERVA Link to assist the provision of message integrity.

message queue. A queue used to store messages on a first-in, first-out basis.

message reference number. A unique 16-digit identifier assigned by MERVA AIX to each message for identification purposes. The message reference number consists of an 8-character domain identifier followed by an 8-digit sequence number.

message separator. A predefined series of characters used to separate message fields. For example, :32A is the separator of the SWIFT currency field. Also known as a scan pattern.

message sequence number (MSN). MERVA Link protocol element. Sequence number for messages transferred by MERVA Link.

message transfer. Message Transfer is the name of the lower sublayer functionality of MERVA Link.

Message Transfer Process or Program (MTP). The MERVA Link Message Transfer Program supports a specific remote partner MTP. It exchanges messages and reports with this partner. The conversation protocol used by these programs must be bilaterally agreed between two programs.

message type (MT). A number, up to 7 digits long, that identifies a message. SWIFT messages are identified by a 3-digit number; for example, SWIFT message type MT S100.

MPDU. Message Protocol Data Unit defined in the MERVA Link P1 protocol.

MRN. Message reference number.

Msg.ID. Message identifier.

MSN. Message Sequence Number.

MTN. Message Transfer Node. The unique identifier of a MERVA Link system. Exchanged as part of the address information when establishing a connection with a remote MERVA Link system.

MTP. Message Transfer Process or Program (MTP). The lower layer of the MERVA Link communications protocol. The MERVA Link Message Transfer Program supports a specific remote partner MTP. It exchanges messages and reports with this partner. The conversation protocol used by these programs must be bilaterally agreed between two programs.

N

nested message. A message that contains another message. For example, SWIFT MT 195 could be used to request information about a SWIFT MT 100 (customer transfer). The SWIFT MT 100 (only mandatory fields) is then nested in SWIFT MT 195.

network identifier. A single character stored together with the message in the MERVA AIX message database that shows which network is to be used to send the message. For example S for SWIFT.

NCP. Network Control Program.

nickname. An abbreviation or synonym of the Bank Identifier Code (BIC) of a financial institution with which you frequently correspond.

NSDU. Network Service Data Unit. A logical unit of data used at the network layer of the SWIFT Link communications protocol.

O

OSN. Output sequence number.

P

Partner Table (PT). In MERVA ESA, the Partner Table defines message processing in MERVA Link. It consists of a header and different entries, such as entries to define the message-processing parameters of an ASP or MTP.

PDE. Possible Duplicate Emission.

PDU. Protocol Data Unit.

Personal Identification Number (PIN). A 6-digit confidential code number used to restrict the use of ICCs to authorized card holders only.

personalize. To customize the information stored about a card set. This includes unblocking the cards, setting the PIN parameters, and for USER cards, setting the LT access rights.

PIN. Personal Identification Number.

pre-agreement. An agreement between an institution and its correspondent that governs the exchange of bilateral keys.

protocol data unit (PDU). In MERVA Link a PDU consists of a structured sequence of implicit and explicit data elements:

- Implicit data elements contain other data elements.
- Explicit data elements cannot contain any other data elements.

PSN. Public switched network (connection).

PT. MERVA Link Partner Table (for MERVA ESA).

PTF. Program Temporary Fix.

PTT. National Post and Telecommunication Authority (post, telegraph, telephone).

PU. Physical unit.

public key. A key held by each institution that is used to encipher a bilateral key received from a correspondent. See also *secret key*.

purpose group. A logical grouping of queues associated with a function. The function is responsible for processing the messages in all queues belonging to the purpose group.

P1. In MERVA Link, a peer-to-peer protocol between cooperating ASPs in remote systems.

P2. In MERVA Link, a peer-to-peer protocol between cooperating MTPs in remote systems.

Q

queue. See *message queue*.

queue buffer. The internal representation of a MERVA AIX message when held in a queue.

queue management. A MERVA AIX process that handles the storing and retrieval of messages in the message database.

R

repeatable sequence. A field or group of fields that can be entered or displayed more than once in a message.

responder. The correspondent that does not initiate a bilateral key exchange. See also *initiator*.

routing. The passing of messages from one of the processing stages in a predefined processing path to the next stage.

routing condition. A logical test to determine the target queues that messages are sent to. Routing conditions are defined for source queues, that is, the queue that messages are taken from for further routing. You can test for:

- The presence of a field within a message
- The presence of data within a message field
- The value of the contents of a message field.

RSA. Asymmetric cryptographic algorithm designed by Rivest, Shamir, and Adleman.

S

scan pattern. A character string placed between message fields to identify where a field begins. Also known as a tag.

SCR. Secure Card Reader.

SDLC. Synchronous Data Link Control.

secret key. The part of an RSA key used to encipher bilateral keys that remains stored inside the SCR. See also *public key*.

secure login and select (SLS) service. ICC-based alternative to the use of paper LOGIN/SELECT tables.

secure transmission key (STK). The key that is generated by the SCR and used to protect the transfer of bilateral keys over the link between the SCR and the workstation. The STK is also used in the workstation to store the bilateral keys securely.

security management center (SMC). The SWIFT facility responsible for security administration and the issue of ICCs to users. The SMC also acts as the certification authority for Public RSA keys.

session key (SK). A number required for each LOGIN and SELECT request.

sign on. To start a MERVA AIX session.

SK. Session Key.

SK number. A parameter stored on an ICC that specifies the number of session keys that can be generated with a USER card before the user must enter the PIN again.

SLS. Secure Login and Select.

SMC. Security Management Center.

SNA. Systems network architecture.

source queue. In a routing condition, the queue from which messages are routed to the next defined message queue.

SSN. Select Sequence Number.

STK. Secure Transmission Key.

subfield. A subdivision of a field with a specific meaning. For example, the SWIFT field 32 has the subfields date, currency, and amount. A field can have several subfield layouts depending on the way the field is used in a particular message.

SWIFT. Society for Worldwide Interbank Financial Telecommunication, s.c.

SWIFT. Refers to the SWIFT II network of the Society for Worldwide Interbank Financial Telecommunication, s.c. (SWIFT).

SWIFT address. A code used to identify a bank within the SWIFT network. The code is also called a bank identifier code (BIC) or a terminal identifier and is assigned by SWIFT.

SWIFT correspondents database. The database containing the SWIFT address or BIC, together with the name, postal address, and zip code of each financial institution in the BIC Directory.

SWIFT destination address. The first 8 characters of the SWIFT address consisting of the bank, country, and location codes.

SWIFT financial message. A message in one of the SWIFT categories 1 to 9 that you can send or receive via the SWIFT network. See *SWIFT input message* and *SWIFT output message.*

SWIFT header. The leading part of a message that contains the sender and receiver of the message, the message priority, and the type of message.

SWIFT input message. A SWIFT message prepared by a user to be sent to the SWIFT network.

SWIFT Link. The MERVA AIX component that provides you with a link to the S.W.I.F.T. II network, enabling you to send messages to and receive messages from the SWIFT network.

SWIFT message. A message in one of the SWIFT categories as defined in the *S.W.I.F.T. User Handbook* that can be sent or received via the SWIFT network. See also SWIFT *input message* and SWIFT *output message*.

SWIFT output message. A SWIFT message coming from the SWIFT network.

SWIFT system message. A message in SWIFT category 0.

systems network architecture (SNA). The description of the logical structure, formats, protocols, and operating sequences for transmitting information units through, and controlling the configuration and operation of, networks.

T

tag. A field identifier, consisting of a 2- or 3-digit number, or a 2-digit number followed by a letter.

target queue. In a routing condition, the message queue to which messages are next routed.

TCT. Terminal control table.

technology flag. A parameter, which is controlled by the USOF, that indicates to SWIFT which access technology, ICCs or paper tables, is being used by the LTs of a particular destination.

TPDU. Transport Protocol Data Unit. A logical unit of data used at the Transport layer of the SWIFT Link communications protocol.

TRN. Transaction Reference Number.

U

UKMO. User Key Management Officer.

uni-directional. A type of bilateral key where different separate keys are used to authenticate messages send to and received from a correspondent.

USE. User Security Enhancements.

USER. SWIFT Link operator; the holder of a USER ICC.

user file. The user file has a record for each MERVA AIX user, containing the user's details. The record specifies the functions that a user is allowed access to. The user file can only be accessed by authorized users.

user key management officer (UKMO). The administrator who is the holder of a UKMO ICC. The UKMO is responsible for managing the exchange and use of bilateral keys, and other BKE-related functions.

user security officer (USOF). The administrator who is the holder of a USOF ICC. The USOF is responsible for control and management of ICCs and card readers and their related data, and for liaison with SWIFT for matters relating to ICCs and card readers.

USOF. User Security Officer.

W

whitelist flag. A mechanism for preventing the use of cards that are suspected of being lost, stolen, or otherwise compromised. If a card is lost, the USOF increments the whitelist flag on the remaining cards, thus rendering the whitelist flag on the lost card incorrect.

X

X.25. ISO standard for interface to packet switched communications services.

Bibliography

IBM Publications

With exception of the General Information and the Licensed Program Specifications, all MERVA books are available as softcopy on the

- *MERVA Family C-Kit*, SK2T-0157

MERVA Family Books

- *MERVA OS/2 Client User's Guide*, SH12-6282
- *MERVA Family USE Administration Guide*, SH12-6065

MERVA OS/2 Books

- *MERVA OS/2 V3 and MERVA ESA V3 General Information*, GH12-6018
- *MERVA OS/2 V3 Licensed Program Specifications*, GH12-6057
- *MERVA OS/2 V3 Application Programming*, SH12-6058
- *MERVA OS/2 V3 Diagnosis Guide*, SH12-6059
- *MERVA OS/2 V3 User's Guide*, SH12-6060
- *MERVA OS/2 V3 Installation and Customization Guide*, SH12-6061

MERVA AIX Books

- *MERVA AIX Licensed Program Specifications*, GH12-6180
- *MERVA AIX User's Guide*, SH12-6181
- *MERVA AIX Installation and Customization Guide*, SH12-6182
- *MERVA AIX Application Programming*, SH12-6183
- *MERVA AIX Diagnosis Guide*, SH12-6184

MERVA ESA Books

- *MERVA OS/2 V3 and MERVA ESA V3 General Information*, GH12-6018
- *MERVA ESA V3 Licensed Program Specifications*, GH12-6019
- *MERVA ESA V3 Application Programming Interface Guide*, SH12-6183
- *MERVA ESA V3 Operations Guide*, SH12-6021
- *MERVA ESA V3 User's Guide*, SH12-6022
- *MERVA ESA V3 Macro Reference*, SH12-6023
- *MERVA ESA V3 Installation Guide*, SH12-6025
- *MERVA ESA V3 Messages and Codes*, SH12-6026
- *MERVA ESA V3 Customization Guide*, SH12-6027
- *MERVA ESA V3 Concepts and Components*, SH12-6028
- *MERVA ESA V3 Advanced MERVA Link*, LY12-5081
- *MERVA ESA V3 Workstation Based Functions*, SH12-6069
- *MERVA ESA V3 IFT Connection for MVS*, SH12-6280
- *MERVA ESA V3 Traffic Reconciliation Reference*, SH12-6281

Further IBM Publications

- *IBM AIX and Related Products Documentation Overview*, SC23-2456
- *IBM DATABASE 2 for AIX Planning Guide*, S20H-4758
- *DB2 Application Programming Guide*, S20H-4643
- *DB2 API Reference*, S20H-4984
- *DB2 Problem Determination Guide*, S20H-4779
- *SNA Server for AIX Diagnosis Guide and Messages*, SC31-8215
- *SNA Server for AIX Command Reference*, SC31-8214
- *SNA Server for AIX Configration Reference*, SC31-8213
- *CID Enablement Guidelines*, S10H-9666
- *CICS-RACF Security Guide*, SC33-1185
- *MVS/ESA Planning: APPC Management*, GC28-1110
- *ITSC redbook APPC Security: MVS/ESA, CICS/ESA, and OS/2*, GG24-3960
- *IMS/ESA Version 4 Data Communication Administration Guide*, SC26-3060

S.W.I.F.T. Publications

The following books are published by the Society for Worldwide Interbank Financial Telecommunication, s.c., in La Hulpe, Belgium:

- *S.W.I.F.T. User Handbook (1996)*
- *S.W.I.F.T. Dictionary (1996)*

- *S.W.I.F.T. Directory (1996)*
- *S.W.I.F.T. FIN Security Guide (1996)*
- *S.W.I.F.T. Card Readers User Guide (1996)*

Index

R

S

T

U

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

MERVA Family
MERVA Connection/6000

Publication No. SH12-6097-02

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
SH12-6097-02

Cut or Fold
Along Line

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

PLACE
POSTAGE
STAMP
HERE

IBM Deutschland Entwicklung GmbH
Information Development, Dept. 0446
Postfach 1380
71003 Boeblingen
Germany

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

SH12-6097-02

Cut or Fold
Along Line

Program Number: 5765-449

Printed in Denmark by IBM Danmark A/S

SH12-6097-02