

MERVA Family



# MERVA Connection/2



MERVA Family



# MERVA Connection/2

**Note!**

Before using this information and the product it supports, be sure to read the general information under “Appendix D. Notices” on page 105.

**First 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.

© **Copyright International Business Machines Corporation 1993, 1997. All rights reserved.**

Note to U.S. Government Users — Documentation related to restricted rights — Use, duplication or disclosure is subject to restrictions set forth in GSA ADP Schedule Contract with IBM Corp.

---

# Contents

<b>About This Book</b> . . . . .	vii
<b>Chapter 1. Introduction to MERVA Connection/2</b> . . . . .	1
Objectives of MERVA Connection/2 . . . . .	1
Functions Provided by MERVA Connection/2 . . . . .	1
Language Support . . . . .	1
Security . . . . .	1
Message Integrity . . . . .	2
Components of MERVA Connection/2 . . . . .	2
<b>Chapter 2. MERVA Connection/2 Client Setup</b> . . . . .	5
MERVA Connection/2 Requirements. . . . .	5
Machine Requirements . . . . .	5
Programming Requirements. . . . .	5
Installing MERVA Connection/2 . . . . .	5
Customizing Communications Server . . . . .	6
Basic SNA Customization . . . . .	7
SNA Customization for MERVA Connection/2 . . . . .	7
Customizing TCP/IP . . . . .	8
Basic TCP/IP Customization . . . . .	8
TCP/IP Customization for MERVA Connection/2 . . . . .	8
Customizing MERVA Connection/2 . . . . .	8
Variable Format Application Profile . . . . .	9
Variable Format Application Profile Parameters. . . . .	10
Fix Format Application Profile . . . . .	12
Selecting the Communication Type . . . . .	13
<b>Chapter 3. RAPI Server Setup on AIX</b> . . . . .	15
Remote API Server Requirements . . . . .	15
Machine Requirements . . . . .	15
Programming Requirements. . . . .	15
Installing the RAPI Server . . . . .	15
Customizing SNA Services . . . . .	15
Basic SNA Customization . . . . .	16
SNA Customization for the RAPI Server . . . . .	16
Customizing TCP/IP Services . . . . .	18
Basic TCP/IP Customization . . . . .	18
TCP/IP Customization for the RAPI Server . . . . .	18
<b>Chapter 4. Remote API Server Setup on OS/2</b> . . . . .	21
Remote API Server Requirements . . . . .	21
Machine Requirements . . . . .	21
Programming Requirements. . . . .	21
Installing Remote API Server . . . . .	21
Installing the Remote MERVA API Server Program . . . . .	21
Installing the Sample Communications Server Configuration Files . . . . .	21
Customizing SNA Services . . . . .	22
Basic SNA Customization . . . . .	22
SNA Customization for the Remote API Server. . . . .	23
Customizing the Trace File for SNA . . . . .	23
Customizing TCP/IP Services . . . . .	24
Customizing Client Network Services . . . . .	24
Customizing Super Daemon Services . . . . .	24

Customizing the Trace File for TCP/IP . . . . .	24
<b>Chapter 5. Verifying Correct Installation and Customization . . . . .</b>	<b>27</b>
<b>Chapter 6. The Application Programming Interface . . . . .</b>	<b>29</b>
Structure of the MERVA API Program on the Client Side . . . . .	29
C Language Data Types . . . . .	29
Additional Functions . . . . .	30
Starting and Ending the Conversation . . . . .	30
ENMSetProfile - Select a Profile . . . . .	30
ENMStartRAPI - Establish Connection to MERVA OS/2 or MERVA AIX . . . . .	31
ENMRestartRAPI - Reconnect Remote API Program to MERVA OS/2 or MERVA AIX . . . . .	32
ENMEndRAPI - Disconnect from MERVA OS/2 or MERVA AIX . . . . .	33
ENMSetSecurity - Set Conversation Security Information . . . . .	33
ENMSetTestEnv - Set Test Environment . . . . .	34
Functions Enabling the API Program to be Triggered . . . . .	35
ENMWaitSemList - Wait for a List of Semaphores. . . . .	35
ENMCloseSem - Close a Semaphore . . . . .	36
ENMSetSem - Set a Semaphore . . . . .	37
ENMClearSem - Clear a Semaphore . . . . .	38
ENMCreateSem - Create a Semaphore . . . . .	39
ENMOpenSem - Open a Semaphore . . . . .	41
Handling Errors . . . . .	42
ENMGetReason - Get Reason Code for Internal Error . . . . .	42
 <b>Chapter 7. Resynchronization . . . . .</b>	 <b>45</b>
How Resynchronization Is Implemented . . . . .	46
Using the Resynchronization Mechanism . . . . .	46
Hints and Tips. . . . .	47
Recovering after a Failed Call . . . . .	47
Not Using Resynchronization . . . . .	47
 <b>Chapter 8. Security . . . . .</b>	 <b>49</b>
Encryption of Transferred Information . . . . .	49
Authentication of Transferred Information . . . . .	49
User Exit Interfaces. . . . .	49
Introduction . . . . .	49
User Exit Points . . . . .	50
User Exit Interfaces in C Language . . . . .	51
User Exit for Encryption . . . . .	52
User Exit for Decryption . . . . .	52
User Exit for MAC Generation . . . . .	53
User Exit for MAC Verification . . . . .	53
 <b>Chapter 9. Building API Programs. . . . .</b>	 <b>55</b>
Compiling Your Own Program . . . . .	55
Compiling the Sample Programs . . . . .	55
List of Sample Files. . . . .	55
 <b>Chapter 10. Replacing Security User Exits . . . . .</b>	 <b>57</b>
Security User Exits . . . . .	57
Generating and Activating Security User Exits on the Client Application System . . . . .	57
Generating and Activating Security User Exits on the MERVA Server System for MERVA OS/2 . . . . .	58

Generating and Activating Security User Exits on the MERVA Server System for MERVA AIX. . . . .	60
<b>Chapter 11. Diagnosis Information</b> . . . . .	61
Log Files on the Client Application Side . . . . .	61
Diagnosis Log . . . . .	61
Programmer's Log . . . . .	61
Log Files on the MERVA OS/2 Side . . . . .	62
Log Files on the MERVA AIX Side . . . . .	62
<b>Appendix A. Sample SNA Definitions</b> . . . . .	63
Customizing an APPN End Node (OS/2) . . . . .	64
SNA Local Node Characteristics . . . . .	64
SNA Connections . . . . .	64
Defining Additional Resources . . . . .	65
Customizing an APPN Network Node (AIX) . . . . .	67
Initial Node Setup . . . . .	67
Check and Modify the Initial Node Setup . . . . .	67
Defining Additional Resources . . . . .	68
Customizing an APPN Network Node (OS/2) . . . . .	69
Customizing an APPC Peer-to-Peer Connection (OS/2) . . . . .	70
Complete Peer-to-Peer Configuration Tables (OS/2) . . . . .	70
<b>Appendix B. Sample Security User Exits</b> . . . . .	73
Module ENM4SNIL - Empty Functions . . . . .	73
Module ENM4SSEC - Sample Functions . . . . .	74
<b>Appendix C. Sample Programs</b> . . . . .	79
Program SMPLO1 . . . . .	80
Program SMPLO2 . . . . .	87
Program SMPLO2S . . . . .	93
Program SMPLO3 . . . . .	95
<b>Appendix D. Notices</b> . . . . .	105
Trademarks. . . . .	106
<b>Glossary of Terms and Abbreviations</b> . . . . .	109
<b>Bibliography</b> . . . . .	111
IBM Publications . . . . .	111
MERVA Family Books . . . . .	111
MERVA OS/2 Books . . . . .	111
MERVA AIX Books . . . . .	111
MERVA ESA Books . . . . .	111
Further IBM Publications . . . . .	111
S.W.I.F.T. Publications . . . . .	112
<b>Index</b> . . . . .	113
<b>Readers' Comments — We'd Like to Hear from You.</b> . . . . .	115





---

## 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 OS/2 Version 3 (abbreviated to MERVA OS/2 V3 in this book) from an application program executing in an OS/2 system.

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:

- Operating System/2 (OS/2)
- Advanced Interactive Executive (AIX) operating system
- Systems Network Architecture (SNA)
- Application Programming Interface (API) of MERVA OS/2
- Application Programming Interface of MERVA AIX
- Transaction Control Protocol/Internet Protocol (TCP/IP)



---

## Chapter 1. Introduction to MERVA Connection/2

This chapter introduces MERVA Connection/2 and briefly describes the facilities supported by MERVA Connection/2.

---

### Objectives of MERVA Connection/2

MERVA Connection/2 provides an interface for application programs on the OS/2 system. It is called the Remote MERVA API (RAPI). Using the Remote MERVA API, you can create an application on OS/2; to send messages to MERVA and receive messages from MERVA with a minimum effort.

MERVA Connection/2 enables you to adapt your OS/2 application to any MERVA system (MERVA OS/2 and MERVA AIX). For example, MERVA Connection/2 provides the functionality to attach a telex provider software to MERVA.

To use OS/2; applications as banking applications, messages created on the OS/2 system must be transferred to a MERVA system. Messages received from any networks must be transferred from a MERVA system to the OS/2 system.

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 OS/2 system to MERVA OS/2 or MERVA AIX, as if MERVA OS/2 or MERVA AIX was a component of the application.

MERVA Connection/2 is a tool that makes it easier for you to implement such a solution. MERVA Connection/2 is not a ready-to-use SWIFT interface on the OS/2 system. It does not have a user interface.

---

### Functions Provided by MERVA Connection/2

MERVA Connection/2 provides the complete functionality of the MERVA AIX or MERVA OS/2 API on the OS/2 system. Additional calls are available for establishing an intersystem connection and making use of MERVA alarms. MERVA Connection/2 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 49).

## Message Integrity

A resynchronization mechanism ensures that the remote API program can provide the same level of message integrity as a local API program (see Figure 10 on page 45).

---

## Components of MERVA Connection/2

Figure 1 names the components of MERVA Connection/2 and illustrates the programming concepts of MERVA Connection/2.

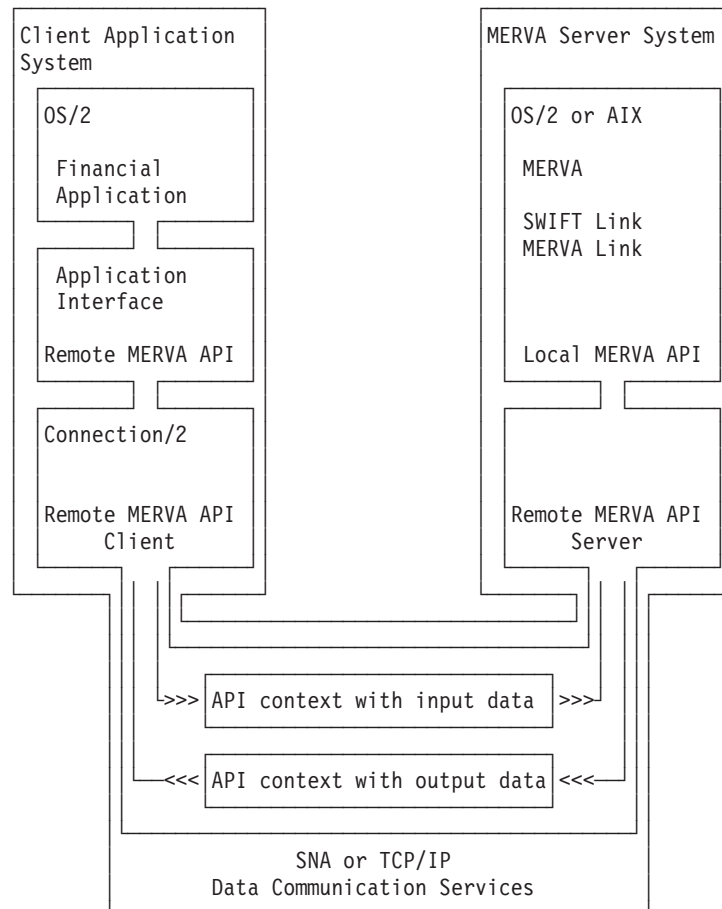


Figure 1. MERVA Connection/2 Concept

MERVA Connection/2 has two main components:

- The **Remote MERVA API Client** is installed and executed on OS/2, 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 a OS/2 system or AIX system, the MERVA Server System. The Remote MERVA API Server is a part of the MERVA AIX or MERVA OS/2 system installed on 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/2 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/2.

---

### MERVA Connection/2 Requirements

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

#### Machine Requirements

The Remote MERVA API Client can be installed on any OS/2 system with approximately one megabyte free space on its hard disk.

The MERVA Connection/2 Client Application System and the MERVA Server System must be interconnected by a Data Communication Link. As specified by the used Data Communication Service (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 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 111.

#### Programming Requirements

The following software must be installed on OS/2 :

- IBM OS/2 Warp Version 3, or a subsequent release
- Personal Communications 4.0 or Communication Server 4.1 *or* TCP/IP for OS/2 Version 3.0, or a subsequent release
- The C Compiler Visual Age for C++ Version 3.0

Personal Communications or Communication Server are required only if an SNA APPC connection is used for the communication between the Remote MERVA API Client and Server. These are not required if a TCP/IP connection is used for that purpose; in this case TCP/IP for OS/2 must be installed on the OS/2 system.

---

### Installing MERVA Connection/2

The directory **CONNECT2** on the MERVA OS/2 V3.3 CD contains all files necessary to install MERVA Connection/2:

```
\CONNECT2\  
  ENMORAPI.DLL  
  ENMORAPI.LIB  
  ENMOSXIT.DLL  
  ENMRATP .DLL  
  ENMRADT .H  
  ENMRAPD .H  
  ENMRAPI .H  
  SAMPLE .PRF  
  SMPL04 .EXE
```

```

\SAMPLES\
  SMPL01 .C
  SMPL01 .DEF
  SMPL01 .MAK
  SMPL02 .C
  SMPL02 .DEF
  SMPL02 .MAK
  SMPL02S .C
  SMPL02S .DEF
  SMPL03 .C
  SMPL03 .DEF
  SMPL03 .MAK
  SMPL04 .C
  SMPL04 .DEF
  SMPL04 .MAK

```

```

\USEREXIT\
  ENM4SNIL.C
  ENM4SSEC.C
  ENMOSXIT.DEF
  ENMOSSEC.MAK

```

**\CONNECT2\** Contains all dynamic link libraries, include files and a sample program for a first installation check (see “Chapter 5. Verifying Correct Installation and Customization” on page 27). The libraries ENMORAPI.DLL, ENMOSXIT.DLL and ENMRATP.DLL must be moved to a directory existing in the LIBPATH environment variable.

- The library ENMORAPI.DLL holds the API entry points and entry points for encryption and authentication routines. This library must be linked by the programs using the Remote MERVA API.
- The library ENMOSXIT.DLL contains entry points for encryption and authentication routines. See “User Exit Interfaces” on page 49 for more information.
- The library ENMRATP.DLL contains the code for the Remote MERVA API Client SNA APPC routines. These routines are loaded at runtime if necessary.

The include files ENMRADT.H, ENMRAPI.H and ENMRAPD.H should be moved to your working directory or into any directory which is contained in the INCLUDE environment variable.

#### **\CONNECT2\SAMPLES\**

This directory contains source files, module definition files (for Visual Age for C++) and makefiles for sample API programs. To compile these samples move the appropriate files to your working directory and process the makefiles.

#### **\CONNECT2\USEREXIT\**

This directory contains sources, a module definition file (for Visual Age for C++) and a makefile for the generation of your own security user exits. See “User Exit Interfaces” on page 49 for more details on this issue.

---

## Customizing Communications Server

MERVA Connection/2 can use SNA APPC services for the communication between the Remote MERVA API Client and Server. As an alternative, MERVA Connection/2 can use TCP/IP services for that purpose.



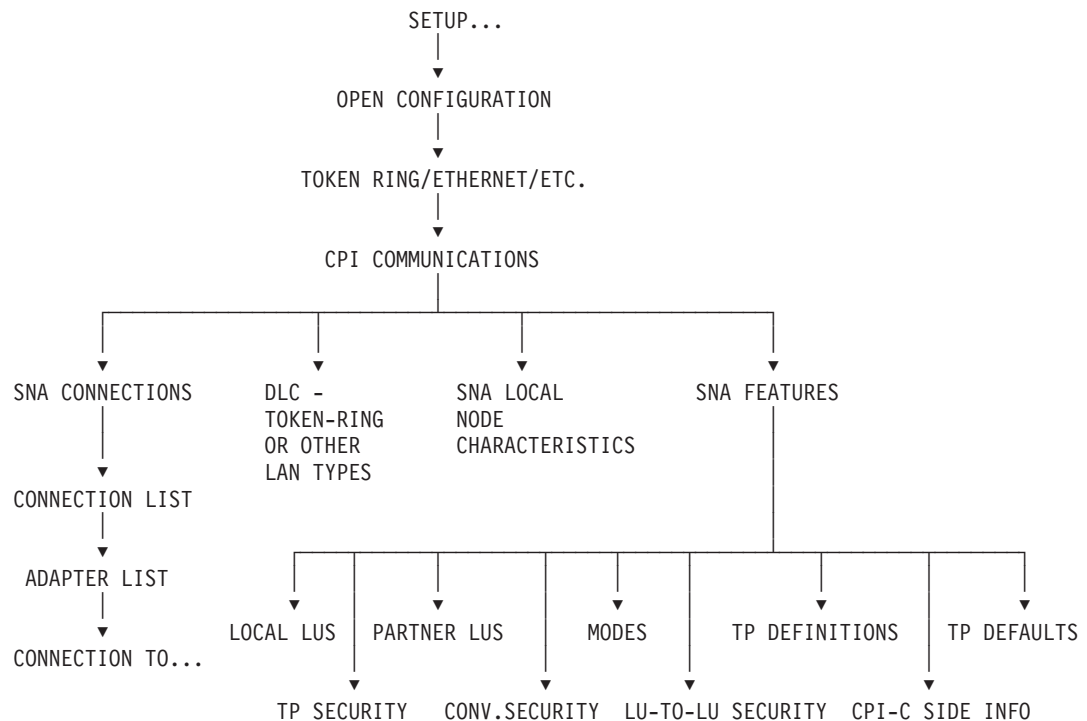
If SNA APPC services are used, Communications Server or Personal Communications must be installed on the OS/2 system and be customized for binding APPC sessions between the two partner systems.

## Basic SNA Customization

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

A sample for the Communications Server customization that is independent of MERVA Connection/2 is provided in “Appendix A. Sample SNA Definitions” on page 63. For beginners in Communications Server customization it is recommended to read through this example.

The following list shows the structure of the Communications Server setup program and tells you how to reach the different panels:



## SNA Customization for MERVA Connection/2

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 in Communications Server are:

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

Sample configuration files can be found in the `\SAMPLES\CONNECT\PC\` directory on the MERVA OS/2 V3.3 CD. See also "Appendix A. Sample SNA Definitions" on page 63.

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 and MERVA OS/2 environment is **ENMRAS**.

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/2 Application Profile. The TP name in the Side Information Profile is disregarded in this case.

---

## Customizing TCP/IP

MERVA Connection/2 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. If TCP/IP is used, TCP/IP for OS/2 must be installed on the OS/2 system. 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 OS/2 system must be customized as a host in an internet, a network of interconnected hosts using TCP/IP communication protocols. No specific MERVA Connection/2 requirements apply for the basic TCP/IP customization (refer to the *Online Reference Guide to TCP/IP* of the TCP/IP for OS/2 package about TCP/IP installation and configuration).

### TCP/IP Customization for MERVA Connection/2

TCP/IP customization for MERVA Connection/2 is not applicable. All information related to the TCP/IP connection to the Remote API Server is provided in the MERVA Connection/2 Application Profile.

You must, however, ensure that the partner host name specified in the MERVA Connection/2 Application Profile can be interpreted by the TCP/IP service. A partner host name can be interpreted if it is defined in the OS/2 hosts file (`C:\MPTN\ETC\HOSTS`), or if it is known by a Name Server in the TCP/IP network.

---

## Customizing MERVA Connection/2

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/2 application is customized by providing information in a MERVA Connection/2 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 MERVA API Client and Server and is only supported due to backward compatibility. The variable format supports additional functions such as TCP/IP interconnection, conversation security, and test environment.

## 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 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 character '#'.

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 is shown in Figure 2. It provides the same function as the sample profile shown in Figure 3 on page 13.

```
#-----
# MERVA Connection/2: Client Application Profile
#-----

log_level           = 1           minimum logging level
#log_mode           = append      append new log entries
system_type         = PS2         type of local/remote system

programmer_log      = plog.log     name of programmer's log file
diagnosis_log       = dlog.log     name of diagnosis log file
control_file        = mip.ctl     name of MIP control file

symbolic_destination = MERVA      SNA APPC SI profile for RAPI Server
#partner_tp_name    = ENMRAS      SNA APPC RAPI Server TP name

#partner_host_name  = merva2      TCP/IP partner host name
#rapi_port_number   = 7118        TCP/IP port number of RAPI Server
#tcp_nodelay        = on          TCP_NODELAY flag will be set

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

#test_environment   = on          RAPI Client test environment active
```

Figure 2. Variable Format Application Profile Sample

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.

### Logging Level

The parameter keyword for the logging level parameter is **log\_level**. The parameter value is a single digit, 1, 2, 3, or 4. Logging level 1 is the lowest level, only error messages are written to the logfile. Logging level 4 is the highest, providing the most 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 plog and dlog 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 plog and dlog entries are written to an empty file.

### System Type

The parameter keyword for the system type parameter is **system\_type**. The parameter value is **PS2**. It identifies the type of the client application system.

### Name of Programmer's Log

The parameter keyword for the programmer's log is **programmer\_log**. The parameter value is the name of an OS/2 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 OS/2 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 OS/2 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 defined in Communications Server. 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 **partner\_host\_name**. The parameter value is the name of the AIX or OS/2 host that houses the RAPI server, or its dotted decimal TCP/IP address. The maximum length of a partner host name is 16 characters.

## TCP/IP Port Number

The parameter keyword for the TCP/IP port number of the Remote API Server is **rapi\_port\_number**. The parameter value is the number assigned to the Remote MERVA API Server, an inetd subserver, in the applicable partner host system. The maximum value of a TCP/IP port number is 64 KB - 1 (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**, which can speed up the TCP/IP communication noticeable.

## 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.

## 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. Logging level (1..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. System type of the client application system (PS2)

A parameter file that starts with a digit (the logging 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 3 on page 13.

```
1
plog.log
dlog.log
MERVA
mip.ct1
PS2
```

*Figure 3. 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.

## 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.

Both, 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 both, SNA APPC and TCP/IP partner information, is available from the application profile.





---

## Chapter 3. RAPI Server Setup on AIX

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

---

### Remote API Server Requirements

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

### Machine Requirements

The MERVA Connection/2 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 AIX system.

For a list of the alternatives available and the hardware required, refer to the appropriate books listed in the "Bibliography" on page 111.

### 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 RAPI Server

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

---

### Customizing SNA Services

MERVA Connection/2 can use SNA APPC services for the communication between the Remote MERVA API Client and Server. As an alternative, MERVA Connection/2 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 AIX system. and be customized for binding APPC sessions between the two partner systems.

## Basic SNA Customization

Various methods apply for the interconnection of an OS/2 and an AIX system. 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/2 is provided in “Appendix A. Sample SNA Definitions” on page 63.

## SNA Customization for the RAPI 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 17.

```

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/2 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 OS/2 system must be customized as a host in an internet, a network of interconnected hosts using TCP/IP communication protocols. No specific MERVA Connection/2 requirements apply for the basic TCP/IP customization.

### TCP/IP Customization for the RAPI 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/2 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 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 111.

#### 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 in 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 22 or "Customizing TCP/IP Services" on page 24 on how to install MERVA Connection/2 with SNA or TCP/IP.

Refer to "Chapter 10. Replacing Security User Exits" on page 57 on how to replace the Security User Exits provided with MERVA Connection/2.

#### 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/2).

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

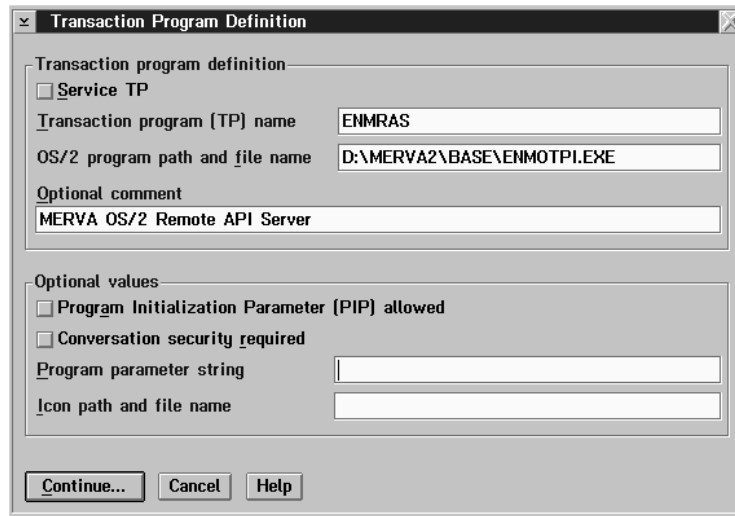


Figure 8. Transaction Program Definition in Communications Server Setup

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:\MPTNETC\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:\MPTNETC\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 chosen. 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. Particularly, 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/2 was successful, run the sample program SMPLO4.EXE. This program is an already compiled version of the sample source SMPLO4.C and is also included in the file enmorsrc.zip on the MERVA Connection/2 disk. SMPLO4.EXE expects to find the profile SMPLO4.PRF in the directory C:\ENMRAPI\, see "Installing MERVA Connection/2" on page 5 on how to obtain the profile.

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 (see *MERVA OS/2 V3 Application Programming* for a description of the prerequisites for the sample 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 Remote API programming on the OS/2 system and the MERVA OS/2 or MERVA AIX API programming.

---

### Structure of the MERVA API Program on the Client Side

One major task of the MERVA API program on the OS/2 system is that it must call functions that deal with connecting and disconnecting to and from MERVA OS/2 system or the MERVA AIX system.

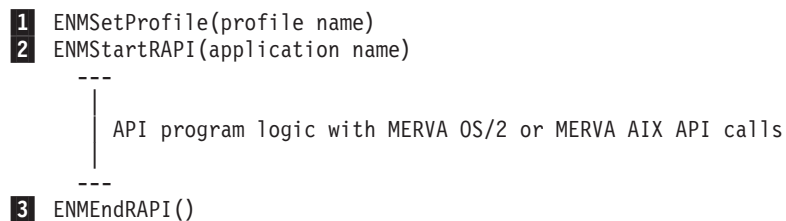


Figure 9. Remote MERVA API Program Structure

- 1** Before the API functions can be called, the Remote MERVA API Client on OS/2 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/2” on page 8.
- 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 Client OS/2 system, 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 `enmraپی.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/2 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/2” on page 8.

### *C Language Example:*

```
#include "enmrapi.h"

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

## ENMStartRAPI - Establish Connection to MERVA OS/2 or MERVA AIX

### C Definition

```
USHORT ENMStartRAPI ( PCHAR pucApplicationName );
```

### Parameter Description

The following parameters are required:

- **retCode**(USHORT) - output

<b>retCode</b>	<b>Meaning</b>
<b>0</b>	Function completed successfully.
<b>2</b>	An internal error has occurred. The API program receives further information by calling the function ENMGetReason, described in “Handling Errors” on page 42. The reason code is also written to the Remote API diagnosis log on the OS/2 system (see “Chapter 11. Diagnosis Information” on page 61). If it is an internal error of the MERVA OS/2 or MERVA AIX API, the reason code is zero.

- **pucApplicationName**(PCHAR) - 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, Reason %d\n", ENMGetReason() );
```

## ENMRestartRAPI

# ENMRestartRAPI - Reconnect Remote API Program to MERVA OS/2 or MERVA AIX

### C Definition

```
USHORT ENMRestartRAPI ( PCHAR pucApplicationName );
```

### Parameter Description

The following parameters are required:

- **retCode**(USHORT) - output

<b>retCode</b>	<b>Meaning</b>
----------------	----------------

<b>0</b>	Function completed successfully.
----------	----------------------------------

<b>2</b>	An internal error has occurred. The API program receives further information by calling the function ENMGetReason, described in "Handling Errors" on page 42. The reason code is also written to the Remote API diagnosis log on the OS/2 system (see "Chapter 11. Diagnosis Information" on page 61). If it is an internal error of the MERVA OS/2 or MERVA AIX API, the reason code is zero.
----------	--

- **pucApplicationName**(PCHAR) - 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

If the connection is established with this call instead of ENMStartRAPI, the resynchronization described in Figure 10 on page 45 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 ENMStartRAPI");
```

## ENMEndRAPI - Disconnect from MERVA OS/2 or MERVA AIX

### C Definition

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

### Parameter Description

The following parameter is required:

- **retCode**(USHORT) - output

retCode	Meaning
0	Function completed successfully.
2	An internal error has occurred. The API program receives further information by calling the function ENMGetReason, described in "Handling Errors" on page 42. The reason code is also written to the Remote API diagnosis log on the OS/2 system (see "Chapter 11. Diagnosis Information" on page 61). 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 ( PCHAR pucUserID,
                    PCHAR pucPassword );
```

### Parameter Description

The following parameters are required:

- **pucUserID**(PCHAR) - input  
A pointer to a null-terminated string of up to 8 characters containing the client user ID.
- **pucPassword**(PCHAR) - 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

**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 ("SAMPLE", "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)**; toggles 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,
                     ...;1
                     (ULONG) 0);1
```

#### Parameter Description

The following parameters are required:

- **retCode**(USHORT) - output

<b>retCode</b>	<b>Meaning</b>
<b>0</b>	Function completed successfully.
<b>2</b>	An internal error has occurred. The API program receives further information by calling the function ENMGetReason, described in "Handling Errors" on page 42. The reason code is also written to the Remote API diagnosis log on the OS/2 system (see "Chapter 11. Diagnosis Information" on page 61). If it is an internal error of the MERVA OS/2 or MERVA AIX API, the reason code is zero.
<b>6</b>	(ERR_OUT_OF_MEMORY) The system does not have enough memory to complete the function.
<b>121</b>	No semaphore is cleared. The timeout was reached.
<b>255</b>	An internal semaphore error occurred, which is dependent on the server system (OS/2 or AIX). ENMGetReason() (see "ENMGetReason - Get Reason Code for Internal Error" on page 42) will provide a reason code of the form 3xxx, where 'xxx' is the error number of the server system.

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).

## ENMWaitSemList

- **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.

### C Language Example

```
/*
The following are OS/2 semaphore names
with prefix, which were used in present
code:

#define TRIGGER "\\SEM\\SAMPLE2"
#define STOP    "\\SEM\\STOP.SMP"
*/
/*
We recommend to use this new form of
a semaphore name:
*/
#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

retCode	Meaning
0	Function completed successfully.
2	An internal error has occurred. The API program receives further information by calling the function ENMGetReason, described in "Handling Errors" on page 42. The reason code is also written to the Remote API diagnosis log on the OS/2 system (see "Chapter 11. Diagnosis Information" on page 61). If it is an internal error of the MERVA OS/2 or MERVA AIX API, the reason code is zero.
6	The system does not have enough memory to complete the function.
255	An internal semaphore error occurred, which is dependent on the server system (OS/2 or AIX). ENMGetReason() (see "ENMGetReason - Get Reason Code for Internal Error" on page 42) will provide a reason code of the form 3xxx, where 'xxx' is the error number of the server system.

- **SemHandle**(ULONG) - input

Generated by ENMCreateSem or ENMOpenSem.

## C Language Example

```

/*
  The following is a OS/2 semaphore name
  with prefix, which was used in present
  code:

  #define TRIGGER "\\SEM\\SAMPLE2"
*/
/*
  We recommend to use this new form of
  a semaphore name:
*/
#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);
```

## ENMSetSem

### Parameter Description

The following parameters are required:

- **retCode**(USHORT) - output

<b>retCode</b>	<b>Meaning</b>
<b>0</b>	Function completed successfully.
<b>2</b>	An internal error has occurred. The API program receives further information by calling the function ENMGetReason, described in "Handling Errors" on page 42. The reason code is also written to the Remote API diagnosis log on the OS/2 system (see "Chapter 11. Diagnosis Information" on page 61). If it is an internal error of the MERVA OS/2 or MERVA AIX API, the reason code is zero.
<b>6</b>	The system does not have enough memory to complete the function.
<b>100</b>	The limit of open semaphores in the system is exceeded.
<b>255</b>	An internal semaphore error occurred, which is dependent on the server system (OS/2 or AIX). ENMGetReason() (see "ENMGetReason - Get Reason Code for Internal Error" on page 42) will provide a reason code of the form 3xxx, where 'xxx' is the error number of the server system.

- **SemHandle**(ULONG) - input

Generated by ENMCreateSem or ENMOpenSem.

### C Language Example

```
/*
  The following is a OS/2 semaphore name
  with prefix, which was used in present
  code:

  #define TRIGGER "\\SEM\\SAMPLE2"
*/
/*
  We recommend to use this new form of
  a semaphore name:
*/
#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

retCode	Meaning
0	Function completed successfully.
2	An internal error has occurred. The API program receives further information by calling the function ENMGetReason, described in "Handling Errors" on page 42. The reason code is also written to the Remote API diagnosis log on the OS/2 system (see "Chapter 11. Diagnosis Information" on page 61). If it is an internal error of the MERV A OS/2 or MERV A AIX API, the reason code is zero.
6	The system does not have enough memory to complete the function.
255	An internal semaphore error occurred, which is dependent on the server system (OS/2 or AIX). ENMGetReason() (see "ENMGetReason - Get Reason Code for Internal Error" on page 42) will provide a reason code of the form 3xxx, where 'xxx' is the error number of the server system.

- **SemHandle**(ULONG) - input

Generated by ENMCreateSem or ENMOpenSem.

## C Language Example

```

/*
The following is a OS/2 semaphore name
with prefix, which was used in present
code:

#define TRIGGER "\\SEM\\SAMPLE2"
*/
/*
We recommend to use this new form of
a semaphore name:
*/
#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 MERV A alarms.

### C Definition

```

USHORT ENMCreateSem (PULONG SemHandle,
                    PCHAR SemName);

```

## ENMCreateSem

### Parameter Description

The following parameters are required:

- **retCode**(USHORT) - output

<b>retCode</b>	<b>Meaning</b>
<b>0</b>	Function completed successfully.
<b>2</b>	An internal error has occurred. The API program receives further information by calling the function ENMGetReason, described in "Handling Errors" on page 42. The reason code is also written to the Remote API diagnosis log on the OS/2 system (see "Chapter 11. Diagnosis Information" on page 61). If it is an internal error of the MERVA OS/2 or MERVA AIX API, the reason code is zero.
<b>6</b>	The system does not have enough memory to complete the function.
<b>100</b>	The limit of open semaphores in the system is exceeded.
<b>123</b>	The name of the semaphore is not valid.
<b>183</b>	The semaphore already exists.
<b>255</b>	An internal semaphore error occurred, which is dependent on the server system (OS/2 or AIX). ENMGetReason() (see "ENMGetReason - Get Reason Code for Internal Error" on page 42) will provide a reason code of the form 3xxx, where 'xxx' is the error number of the server system.

- **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. The semaphore name is a logical name without path details. The \SEM\ prefix used for OS/2 semaphores is not necessary, although you may still use semaphore names with this prefix (if no prefix is provided, MERVA Connection/2 will add this prefix automatically on the OS/2 side). For portability reasons, we recommend to use the new semaphore names without \SEM\.

### C Language Example

```
/*
   The following is a OS/2 semaphore name
   with prefix, which was used in present
   code:

   #define TRIGGER "\\SEM\\SAMPLE2"
*/
/*
   We recommend to use this new form of
   a semaphore name:
*/
#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,
                  PCHAR SemName);
```

### Parameter Description

The following parameters are required:

- **retCode**(USHORT) - output

<b>retCode</b>	<b>Meaning</b>
<b>0</b>	Function completed successfully.
<b>2</b>	An internal error has occurred. The API program receives further information by calling the function ENMGetReason, described in "Handling Errors" on page 42. The reason code is also written to the Remote API diagnosis log on the OS/2 system (see "Chapter 11. Diagnosis Information" on page 61). If it is an internal error of the MERVA OS/2 or MERVA AIX API, the reason code is zero.
<b>6</b>	The system does not have enough memory to complete the function.
<b>100</b>	Limit of open semaphores in the system is exceeded.
<b>123</b>	The name for the semaphore is not valid.
<b>187</b>	The semaphore does not exist.
<b>255</b>	An internal semaphore error occurred, which is dependent on the server system (OS/2 or AIX). ENMGetReason() (see "ENMGetReason - Get Reason Code for Internal Error" on page 42) will provide a reason code of the form 3xxx, where 'xxx' is the error number of the server system.

- **SemHandle**(PULONG) - output  
Address of the handle of the opened semaphore.
- **SemName**(PCHAR) - input  
Pointer to a null-terminated string containing the name of the semaphore to be opened.

### C Language Example

```
/*
  The following is a OS/2 semaphore name
  with prefix, which was used in present
  code:

  #define TRIGGER "\\SEM\\SAMPLE2"
*/
/*
  We recommend to use this new form of
  a semaphore name:
*/
```

## ENMOpenSem

```
#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/2.

If an internal error occurs either in MERVA Connection/2 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/2, 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
<b>2xxx</b>	All reason codes between 2000 and 2999 indicate communication problems.
<b>2110</b>	The APPC conversation cannot be established or is canceled.
<b>2120</b>	The Communications Side Information object is not found.
<b>2130</b>	Connection to Remote MERVA API Server program failed.
<b>2140</b>	Deallocation failed because the conversation has already been terminated.
<b>2150</b>	Conversation is interrupted while trying to receive data.
<b>2200</b>	An empty data buffer was received.
<b>29xx</b>	xx is a return code of the CPI-C call.
<b>2999</b>	A general communication problem occurred (see diagnosis log).
<b>3xxx</b>	An internal semaphore error occurred. 'xxx' is the error number provided by OS/2 or AIX. Error codes of both systems are not identical!
<b>7006</b>	The Remote MERVA API Server failed while allocating memory.
<b>7012</b>	The Remote MERVA API Server does not accept further API calls due to a previous error.

<b>7013</b>	The Remote MERV API Server received a negative return code from user exit ENMExitDecrypt.
<b>7014</b>	The Remote MERV API Server received a negative return code from user exit ENMExitEncrypt.
<b>7015</b>	The Remote MERV API Server received a negative return code from user exit ENMExitMacVerify or ENMExitMacGen.
<b>7016</b>	The Remote MERV API Server received an incorrect API request.
<b>7018</b>	The Remote MERV API Server received an error when converting ASCII to EBCDIC. See the diagnosis log of MERV OS/2 or MERV AIX.
<b>7019</b>	The Remote MERV API Server received an error while accessing the message integrity control file.
<b>7030</b>	Internal message space has not been created.
<b>8002</b>	The Remote MERV API Client cannot open the programmer's log file specified in the profile.
<b>8003</b>	The Remote MERV API Client cannot close the programmer's log file specified in the profile.
<b>8004</b>	The Remote MERV API Client cannot open the diagnosis log file specified in the profile.
<b>8005</b>	The Remote MERV API Client cannot close the diagnosis log file specified in the profile.
<b>8006</b>	The Remote MERV API Client could not allocate memory.
<b>8007</b>	The Remote MERV API Client cannot write to the diagnosis log file specified in the profile.
<b>8008</b>	The Remote MERV API Client cannot write to the programmer's log file specified in the profile.
<b>8010</b>	The RRemote MERV API Client failed because the profile name in ENMSetProfile was incorrect or was not specified.
<b>8011</b>	The Remote MERV API Client failed because the profile specified in ENMSetProfile does not exist.
<b>8013</b>	The Remote MERV API Client received a negative return code from user exit ENMExitDecrypt.
<b>8014</b>	The Remote MERV API Client received a negative return code from user exit ENMExitEncrypt.
<b>8015</b>	The Remote MERV API Client received a negative return code from user exit ENMExitMacVerify.
<b>8016</b>	The Remote MERV API Client received a negative return code from user exit ENMExitMacGen.
<b>8017</b>	Conversation has not been started with ENMStartAPPC.
<b>8019</b>	The Remote MERV API Client could not access the message integrity control file.
<b>8020</b>	The Remote MERV API Client could not load ENMRATP.DLL.
<b>8021</b>	Partner system info is missing in the profile.

## ENMGetReason

### C Language example

```
#include "enmrapi.h"

USHORT rc = 0;
USHORT reason = 0;

rc = ENMFree();
if (rc) {
    reason = ENMGetReason();
    if (reason) {
        printf ("Internal error in MERVA Connection/2 occurred, reason code %d",
            reason);
    }
}
```

## 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.

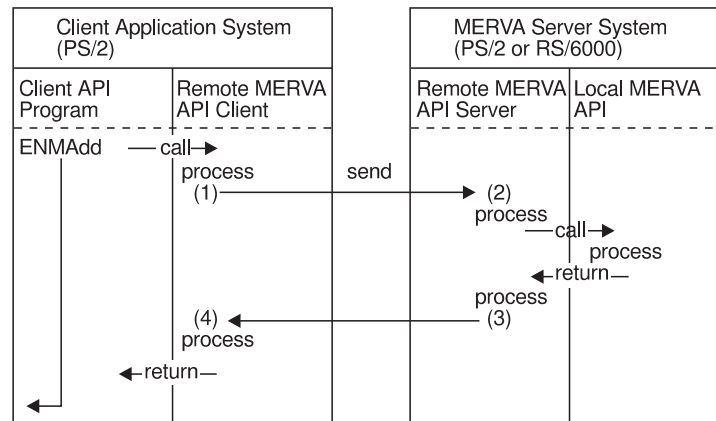


Figure 10. Resynchronization Support

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 (see *MERVA OS/2 V3 Application Programming*). MERVA Connection/2 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/2 the API program reestablishes the connection in the next run using `ENMRestartAPI`. 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/2 profile as described in “Customizing MERVA Connection/2” on page 8. 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 structure of 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 OS/2 V3 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/2 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/2 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/2.

---

### Encryption of Transferred Information

Using MERVA Connection/2 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/2 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/2.

#### 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/2.

- A user exit is a routine written by you and called by MERVA Connection/2. 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 on page 51 shows what happens when an API function is called by an API program on the OS/2 system. 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:

<b>ENCRYP</b>	ENM4ExitEncrypt
<b>DECRYP</b>	ENM4ExitDecrypt
<b>MACGEN</b>	ENM4ExitMacGen
<b>MACVFY</b>	ENM4ExitMacVerify

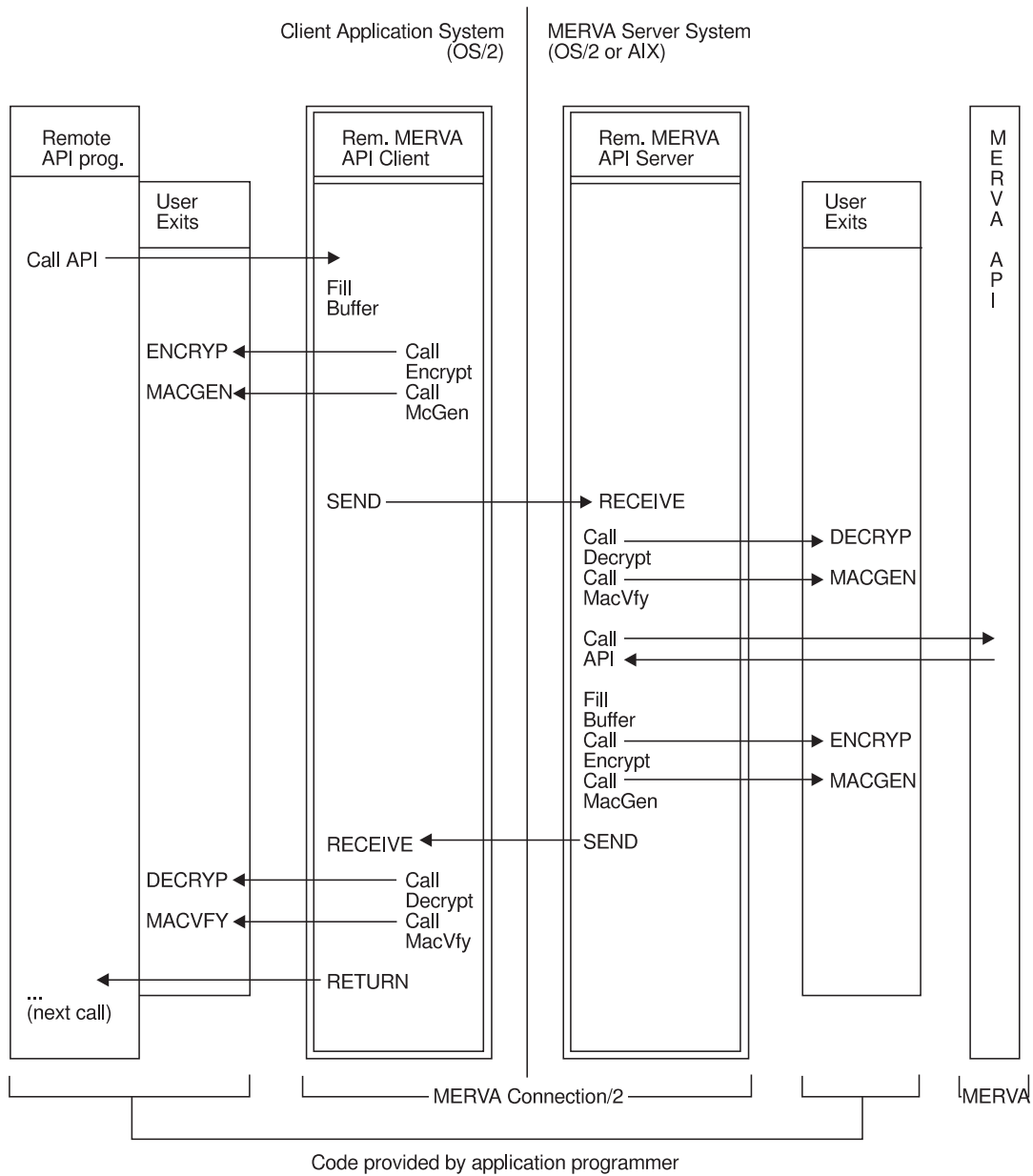


Figure 11. User Exit Points

## 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.

## ENM4ExitEncrypt

### 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* pucAppId,
                               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:

- **pucAppId**(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* pucAppId,
                                   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:

- **pucAppId**(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 MAC and for which the passed MAC has been generated on the partner side.

## ENM4ExitMacVerify

- **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.



---

## Chapter 9. Building API Programs

This chapter describes how to compile MERVA Connection/2 programs in the C programming language.

---

### Compiling Your Own Program

To generate your API program on the OS/2 system, issue the following commands:

- `icc /C /DOS2 /Gd+ /Gm+ /Gs- /Gt+ <name>.c`
- `ilink /NOE <name>.obj ENMORAPI.LIB <name>.def`

It may be necessary to link additional libraries to your program.

---

### Compiling the Sample Programs

To generate the executable files for the delivered sample programs, place the contents of the \CONNECT2\SAMPLES\ directory (see the following list) in a working directory of your choice. All sample programs use the profile SAMPLE.PRF which must reside in the same path as the sample program.

#### List of Sample Files

**SMPLO1.MAK**

Makefile for SMPLO1. Use **nmake /f smplo1.mak** to generate the sample API program SMPLO1.

**SMPLO1.C**

Sample program (attaching to MERVA, querying queue information, creating messages, sending messages etc.)

**SMPLO1.DEF** Module definitions file for SMPLO1

**SMPLO2.MAK**

Makefile for SMPLO2. Use **nmake /f smplo2.mak** to generate the sample API programs SMPLO2 and SMPLO2S.

**SMPLO2.C** MERVA triggering sample program

**SMPLO2.DEF** Module definitions file for SMPLO2

**SMPLO2S.C** Program used to stop SMPLO2 from running

**SMPLO2S.DEF**

Module definitions file for SMPLO2S

**SMPLO3.MAK**

Makefile for SMPLO3. Use **nmake /f smplo3.mak** to generate the sample API program SMPLO3.

**SMPLO3.C** Program to load Telex Messages through API queues

**SMPLO3.DEF** Module definitions file for SMPLO3

**SMPLO4.MAK**

Makefile for SMPLO4. Use **nmake /f smplo4.mak** to generate the sample API program SMPLO4.

**SMPLO4.C** Sample to verify Connection/2 installation

**SMPLO4.DEF** Module definitions file for SMPLO4

**SAMPLE.PRF** File containing a sample profile

**SMPLO4.EXE** Compiled version of SMPLO4.C for immediate use

---

## 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 client application system and the MERVA server system (MERVA OS/2 and MERVA AIX side).

---

### Security User Exits

Two sets of sample security user exits are provided (see “User Exit Interfaces” on page 49):

- |                 |  |
|-----------------|--|
| <b>enm4ssec</b> | 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 on the client application system is the shared library <code>enmosxit.dll</code> , on the MERVA OS/2 side it is the dynamic link library <code>enm4ssec.dll</code> , and on the MERVA AIX side it is the shared library <code>libenmssec.a</code> . |
| <b>enm4snil</b> | These routines do not contain any code. Use this file if no encryption or authentication is desired. The provided code on the client application system is the shared library <code>enmosxit.dll</code> , on the MERVA OS/2 side it is the dynamic link library <code>enm4snil.dll</code> , and on the MERVA AIX side it is the shared library <code>libenmsnil.a</code> .   |

On the client application system the shared library containing the user exits is `enmosxit.dll`. If you want to add your own user exits exchange this library with your own. You may use `ENMOSXIT.MAK` to achieve this.

On the MERVA OS/2 side the dynamic link library containing the user exits must have the name `enmosxit.dll`. The shipped version of `enmosxit.dll` is a copy of the sample library `enm4snil.dll`.

On the MERVA AIX side 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`.

If you want to use the second set (`enm4ssec`) of sample user exit routines, copy (and rename) the following files:

- `enmrssec.dll` to `enmosxit.dll` on the remote application side
- `enm4ssec.dll` to `enmosxit.dll` on the MERVA OS/2 side
- `libenmssec.a` to `libenmsxit.a` on the MERVA AIX side.

### Generating and Activating Security User Exits on the Client Application System

On the remote application side the user exit routines must be placed in shared libraries.

To replace the sample user exits by your own routines, use `enm4ssec.c` as a skeleton. Generate a shared library to replace `enmosxit.dll`. To do this, replace all occurrences of “`enm4snil`” in **`enmosxit.mak`** and **`enmosxit.lrf`** with “`enm4ssec`”. Then start the compilation of the DLL with the following command:

```
nmake all /f enmosxit.mak
```

This will generate a new enmosxit.dll with the user exits contained in enm4ssec.c. Replace the previous version of the DLL with the new one. The files that you need for the generation reside in the compressed file enmorsrc.zip.

## **Generating and Activating Security User Exits on the MERVA Server System for MERVA OS/2**

On the MERVA OS/2 side the user exit routines must be placed in DLLs.

If you want to replace the sample user exits with your own routines, use enm4ssec.c as a skeleton. The following file generates a new enmosxit.dll from enm4ssec.c:

```

#-----
# MAKEFILE - MERVA Connection/2 User Exits DLL
#-----
# Compile-Options:
# /C      Compile, do not link
# /Gd-    Static linking of the runtime library
# /O-     No Optimization
# /Ti     Generate debugger information
# /N2     End make after 2 errors
# /Gm+    Link with multithreaded library
# /Gs-    Don't remove stack probes
# /Gt+    Enable variables for passing to 16 bit functions
# /W2     Compiler messages severity level 2
# /Ge-    Compile for DLL (not for EXE)
#
# /DOS2   Defines the variable OS2 which indicates the use of OS/2
#
# Linker-Options:
# /NOE    Don't use extended dictionary to search libraries
# /STACK: Set the Stack size
#
# Usage:
#         nmake all /f enmossec.mak
#-----

#-----
# Options
#-----
C_OPT = /C /DOS2 /O- /Ti /N2 /Gd- /Gm+ /Gs- /Gt+ /W2 /Ge-
L_OPT = /NOE /STACK:65536

#-----
# Objects which are to be generated in this makefile
#-----
ALL:      ENMOSXIT.DLL \
          ENMOSXIT.LIB

#-----
# Link ENMOSXIT.DLL
#-----
ENMOSXIT.DLL: ENM4SSEC.OBJ
              ILINK $(L_OPT) /DLL ENM4SSEC.OBJ \
              /OUT:ENMOSXIT.DLL \
              /MAP:ENMOSXIT.MAP \
              OS2386.LIB CPPOM30.LIB \
              ENMOSXIT.DEF

#-----
# Compile ENM4SSEC
#-----
ENM4SSEC.OBJ: ENM4SSEC.C \
              ENM4SXIT.H
              $(CC) $(C_OPT) ENM4SSEC.C

#-----
# Generate LIB file for User Exits DLL
#-----
ENMOSXIT.LIB: ENMOSXIT.DEF
              IMPLIB ENMOSXIT.LIB ENMOSXIT.DEF

```

Figure 12. Make File *enmossec.mak* to Generate a DLL

The makefile *enmossec.mak* is also provided in the compressed file *enmorsrc.zip* (see “Installing MERVA Connection/2” on page 5 on how to unpack this file).

Use the following command to create the new DLL:

```
nmake /f enmossec.mak
```

Replace the old enmosxit.dll with the newly generated enmosxit.dll.

If your source file name is different from enm4ssec.c, replace every occurrence of enm4ssec within the make file enmossec.mak with your source file name.

## **Generating and Activating Security User Exits on the MERVA Server System for MERVA AIX**

The sample security user exits can be accessed by the Remote MERVA API Server on the MERVA AIX side 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 enm4ssec.c as a skeleton. The file can be retrieved from directory */usr/lpp/merva/samples*. The file enm4ssec.mak generates a new library libenmssec.a from the source file enm4ssec.c.

Use the following command:

```
make -f enm4ssec.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 client application side, the MERVA OS/2 side, and the MERVA AIX side.

---

### Log Files on the Client Application Side

Two logs are written. You can choose their names and directories by setting them in the MERVA Connection/2 profile (see “Customizing MERVA Connection/2” on page 8).

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

```
* 19930402192358ENM4RAPI ENMRestartRAPI 00000 00000
ENM9153: API function ENMRestartRAPI called.
      Parameters:
      App: SAMPLE3

* 19930402192357ENM4RUTL APIInit 00000 00000
ENM9108: Error in CPIC Call  CMALLC  RC = 19.

* 19930402192413ENM4RAPI ENMRestartRAPI
ENM9109: Error in RAPI Initialization.

* 19930402192413ENM4RAPI ENMRestartRAPI
ENM9152: API function returned with reason code 2130.
```

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 OS/2 V3 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.

<b>Date</b>	The date is in the form YYYYMMDD, where YYYY is the year, MM is the month, and DD is the day.
<b>Time</b>	The time is in the form HHMMSS, where HH is the hour, MM are the minutes, and SS are the seconds.
<b>Module name</b>	The module name is an 8-character code identifying the module the message originated from.
<b>Function name</b>	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 OS/2 V3 Messages and Codes* for the meaning of the messages.

**Note:** Logging entries are appended to the existing files. If you want that MERVA Connection/2 creates new log files, delete the old log files.

---

## Log Files on the MERVA OS/2 Side

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.

---

## Log Files on the MERVA AIX Side

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*.



---

## Appendix A. Sample SNA Definitions

MERVA Connection/2 can use 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 two examples for these many ways. The first 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. This example uses APPN and can only be used if a Network Node is available in the LAN (see "Customizing an APPN Network Node (AIX)" on page 67 and "Customizing an APPN Network Node (OS/2)" on page 69). The MERVA server system resides on the network node in this example. The second example uses an APPC peer-to-peer connection for communication. In this case, no Network Node is necessary but the maintenance of such a configuration is more costly in terms of time.

The naming conventions for the SNA resources in the sample Network Node (also MERVA Connection/2 server) in the first example are:

**NNA** The control point (CP) name or local node (LN) name of the MERVA server (either AIX or OS/2). A second server in this APPN network would be named NNB.

**ENMRAS**

The name of the Transaction Program (MERVA Connection/2 server) on the network node.

**LUA** The name of an independent LU 6.2 in NNA.

The sample token ring address of NNA is **100000000000**. This is just an exemplary address, you have to replace it with your actual address of the network node.

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

**EN1** The end node name (client side).

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

**LU1** The name of an independent LU 6.2 in EN1.

**MERVA**

The symbolic destination name used in EN1.

In the peer-to-peer example, a second end node is used for the server side:

**EN2** The end node name (server side).

**LUB** The name of an independent LU 6.2 in EN2.

**ENMRAS**

The name of the Transaction Program (MERVA Connection/2 server) in EN2.

**Note:** For the APPN example at least one workstation (the network node) must have software installed, which provides the network node services (for

example Communications Server or SNA Server for AIX). For the peer-to-peer example, Personal Communications is fully sufficient.

---

## Customizing an APPN End Node (OS/2)

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

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.

The MERVA Connection/2 SNA customization samples define 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.

## SNA Local Node Characteristics

The local node setup can be performed by entering the applicable parameters in the OS/2 Communications Manager Setup's "SNA local node characteristics" panel shown in Figure 14. Start **Communications Server Setup**, select "Setup..." and a configuration file. Then double-click on "CPI Communications" and in the following profile list choose "SNA local node characteristics".

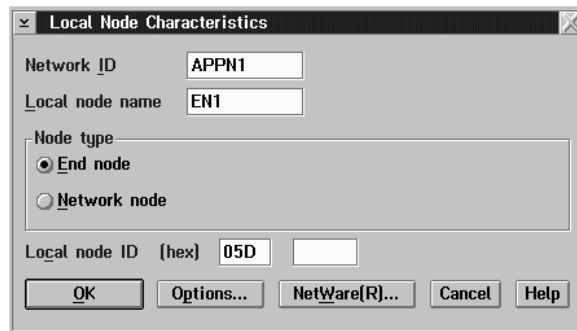


Figure 14. SNA Local Node Characteristics

The field **Local node ID** may be left blank. Use the **Options...** button, to get to an additional panel where you can enter the End node alias.

## SNA Connections

To create a link station for the End node, select "SNA connections" in the profile list. Choose **To network node** and click on the **Create...** button. Select the appropriate adapter type and push **Continue....** The following panel is shown in Figure 15 on page 65.

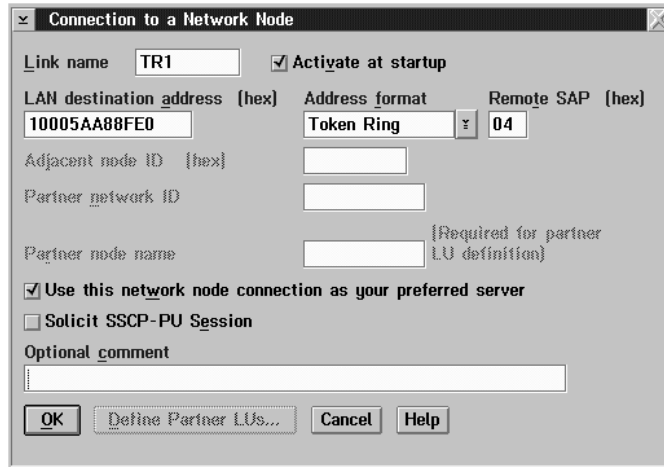


Figure 15. Connection to a Network Node

Fill in the LAN destination address and allow “Activate at startup”. After clicking on the OK button, your link station is defined.

## Defining Additional Resources

A Local LU, a Partner LU, an APPC Session Mode, and a Side Information Profile are the additional resources required for the communication with the MERVA Server System. Select “SNA features” in the profile list of Communications Server Setup.

### Local LU and Partner LU

The sample LU 6.2 Local LU profile for node EN1 is shown in Figure 16.

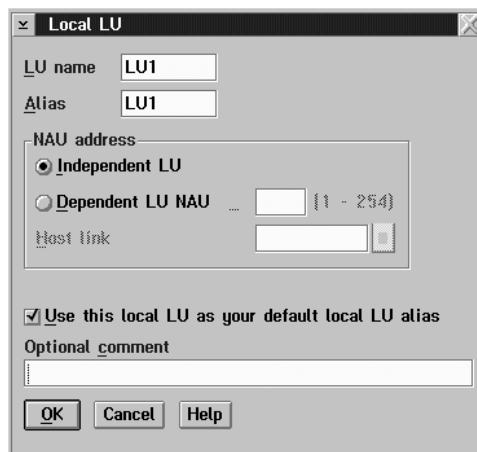
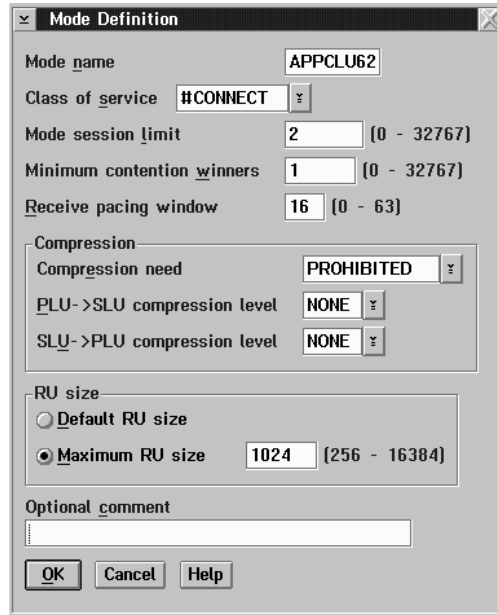


Figure 16. LU 6.2 Local LU Profile in Node EN1

To define the Partner LU select “Partner LU” in the features list and fill in “APPN1.LUA” as the **LU name** and “LUA” as the **LU alias**. Turn on **Conversation security**.

## Mode

The sample LU 6.2 Mode for application sessions (APPCLU62) is shown in Figure 17. In the “Features List” select **Modes** to get to this panel.



The screenshot shows a dialog box titled "Mode Definition" with the following fields and options:

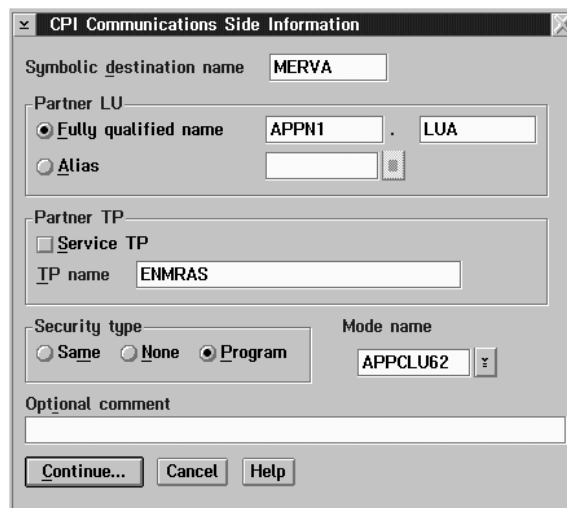
- Mode name: APPCLU62
- Class of service: #CONNECT
- Mode session limit: 2 (range 0 - 32767)
- Minimum contention winners: 1 (range 0 - 32767)
- Receive pacing window: 16 (range 0 - 63)
- Compression section:
  - Compression need: PROHIBITED
  - PLU->SLU compression level: NONE
  - SLU->PLU compression level: NONE
- RU size section:
  - Default RU size:
  - Maximum RU size: 1024 (range 256 - 16384)
- Optional comment: (empty text box)
- Buttons: OK, Cancel, Help

Figure 17. LU 6.2 Mode Profile in Node EN1

The sample Mode in Figure 17 defines an SNA logon mode that can be used for APPC sessions to all kinds of partner systems.

## CPI Communications Side Information

The sample CPI Communications (CPIC) Side Information for the Remote MERVA API Server is shown in Figure 18.



The screenshot shows a dialog box titled "CPI Communications Side Information" with the following fields and options:

- Symbolic destination name: MERVA
- Partner LU section:
  - Fully qualified name: APPN1 . LUA
  - Alias:
- Partner TP section:
  - Service TP:
  - TP name: ENMRAS
- Security type section:
  - Same:
  - None:
  - Program:
- Mode name: APPCLU62
- Optional comment: (empty text box)
- Buttons: Continue..., Cancel, Help

Figure 18. LU 6.2 Side Information Profile ENMRAS

Click on the **Continue...** button to insert the User ID and the password for conversation security.

---

## 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

The sample SNA Control Point profile for node NNA is shown in Figure 19.

```
Change/Show Control Point Profile

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

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

Figure 19. SNA Control Point in Node NNA

The Control Point profile modified by the initial node setup is not modified by the MERVA Connection/2 configuration sample.

## Token Ring SNA DLC Profile

The sample Token Ring SNA DLC profile TR1 for node EN1 is shown in Figure 20.

```

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 20. Token Ring SNA DLC Profile TR1 in Node EN1

The Token Ring SNA DLC profile generated by the initial node setup is not modified by the MERVA Connection/2 configuration sample.

## Defining Additional Resources

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

### Local LU Profile

The sample LU 6.2 Local LU profile for node NNA is shown in Figure 21 on page 69.

## 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 21. 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 17 on page 66 for OS/2. It is the same in nodes EN1 and NNA on both OS/2 and AIX.

### 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 17.

---

## Customizing an APPN Network Node (OS/2)

An APPN Network Node can only be customized with 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-click on "CPI Communications" and in the following profile list choose "SNA local node characteristics".

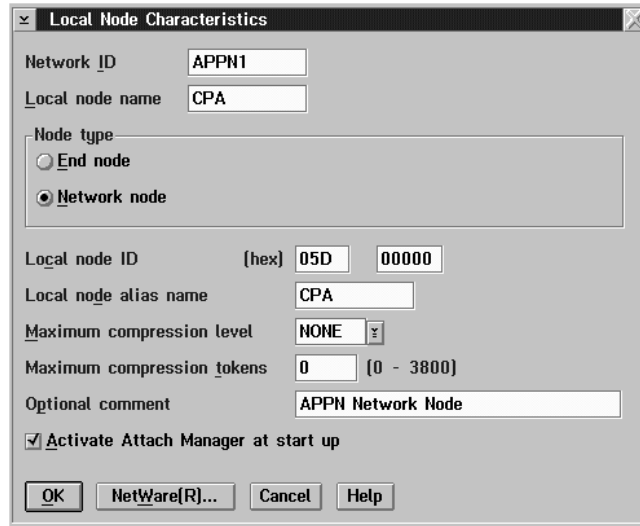


Figure 22. SNA Local Node Characteristics for Network Node

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.

## Customizing an APPC Peer-to-Peer Connection (OS/2)

The following tables describe a simple peer-to-peer connection between the client and the server side of MERVA Connection/2. Both stations run OS/2 (refer to the accompanying documentation for other operating systems, for example, AIX). You have to adapt the network name **NETNAME** used in this example (replace it with your network name) and the LAN destination address of the server to your needs.

Start Communications Manager/2 Setup, choose "Setup..." and a configuration name of your choice. Then, select "APPC APIs over Token-ring" in the list box. Now, the Communications Manager Profile list appears. Go through this list by configuring all profiles in this list. The following tables provide all entries you should make in these profiles. Take care to distinguish between the values for the server (on the left) and the client (on the right).

## Complete Peer-to-Peer Configuration Tables (OS/2)

Table 1. DLC Token-Ring or Other LAN Types

Parameter	Server side	Client side
C&SM LAN ID	NETNAME (change!)	NETNAME (change!)
All others	no changes	no changes

Table 2. SNA Local Node Characteristics

Parameter	Server side	Client side
Network ID	NETNAME (change!)	NETNAME (change!)
Local node name	EN2	EN1
Node type	End node	End node
Local node ID	(maybe blank)	(maybe blank)



Table 2. SNA Local Node Characteristics (continued)

Parameter	Server side	Client side
Local node alias name	EN2	EN1
Activate Attach Manager at startup	(yes)	(yes)
All others	no changes	no changes

Do not make any changes in the “NetWare(R)...” panel.

Table 3. SNA Connections

Parameter	Server side	Client side
Partner Type	-	To end node (click on “Create...”)
Adapter Type	-	Token-ring or other LAN types (“Continue...”)
Link name	-	TR1
Activate at startup	-	(yes)
LAN destination address	-	(insert server network address Entry. <sup>2</sup> , then “OK”)
All others	-	no changes

Table 4. SNA dependent LU Server Definitions

Parameter	Server side	Client side
Whole profile	no changes	no changes

Table 5. SNA Features

Parameter	Server side	Client side
<b>Local LUs</b>		
Lu name	LUB	LUA
Alias	LUB	LUA
NAU address	Independent LU	Independent LU
<b>Partner LUs</b>		
Fully qualified LU name	NETNAME.LUA (change!)	NETNAME.LUB (change!)
Alias	LUA	LUB
Conversation security verification	(yes)	(yes)
<b>Modes</b>		
Mode name	APPCLU62	APPCLU62
Class of service	#CONNECT	#CONNECT
Mode session limit	4	4
Minimum contention winners	2	2
Maximum RU size	1024	1024
<b>Transaction program definitions</b>		
Service TP	(no)	-
Transaction program (TP) name	ENMTPI	

Table 5. SNA Features (continued)

Parameter	Server side	Client side
OS/2 program path and file name	X:\EXAMPLE\ENMOTPI.EXE	-
Programm Initialization parameter (PIP) allowed	(no)	-
Conversation security required	(yes) (click on "Continue...")	-
Presentation type	Background	-
Operation type	Non-queued, Attach Manager started	-
<b>Transaction program defaults</b>		
<i>All parameters</i>	<i>no changes</i>	<i>no changes</i>
<b>Transaction program security</b>		
<i>All parameters</i>	<i>no changes</i>	<i>no changes</i>
<b>Conversation security</b>		
UserID	SAMPLE (or your choice)	-
Password	SAMPLE1 (also retype)	-
Utilize User Profile Management	(yes) (click on "Add")	-
<b>LU-to-LU security</b>		
<i>All parameters</i>	<i>no changes</i>	<i>no changes</i>
<b>CPI Communications side information</b>		
Symbolic destination name	-	MERVA
Partner LU (Fully qualified name)	-	NETNAME.LUB (change!)
Service TP	-	(no)
TP name	-	ENMTPI
Security type	-	Program
Mode name	-	APPCLU62 (click on "OK")
User ID	-	SAMPLE (or your choice)
Password (and retype)	-	SAMPLE1
<b>Generally applies:</b>		
<i>All others</i>	<i>no changes</i>	<i>no changes</i>

---

## Appendix B. Sample Security User Exits

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

---

### Module ENM4SNIL - Empty Functions

This program is integrated into MERVA Connection/2 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.

```
/*-----*\
| ENM4SNIL.C                                     |
\*-----*/
#if defined(WIN32)
    #include <windows.h>
#elif defined(OS2)
    #define INCL_BASE
    #include <os2.h>
#endif

#include <stdlib.h>
#include <stdio.h>
#include <stddef.h>
#include <string.h>
#include <time.h>

#include "enm4sxit.h"

#ifndef __32BIT__
    #define APIENTRY16 APIENTRY
    #define PCHAR16 PCHAR
#endif

USHORT APIENTRY16 ENM4ExitMacGen (PCHAR16 pucAppId,
                                PCHAR16 pucBuffer,
                                USHORT usBufferLen,
                                PCHAR16 pucMacBuffer)
{
    return(0);
}

USHORT APIENTRY16 ENM4ExitMacVerify (PCHAR16 pucAppId,
                                    PCHAR16 pucBuffer,
                                    USHORT usBufferLen,
                                    PCHAR16 pucMacBuffer)
{
    return(0);
}

USHORT APIENTRY16 ENM4ExitEncrypt ( PCHAR16 pucAppId,
                                    PCHAR16 pucBuffer,
                                    USHORT usBufferLen )
```

Figure 23. Sample Security User Exit ENM4SNIL.C (Part 1 of 2)

```

    {
        return(0);
    }

USHORT APIENTRY16 ENM4ExitDecrypt ( PCHAR16 pucAppId,
                                    PCHAR16 pucBuffer,
                                    USHORT   usBufferLen )
{
    return(0);
}

```

*Figure 23. Sample Security User Exit ENM4SNIL.C (Part 2 of 2)*

---

## Module ENM4SSEC - 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.

```

/*-----*\
| ENM4SSEC.C                                     |
\*-----*/
#if defined(OS2)
    #define INCL_BASE
    #include <OS2.H>
#endif
#ifdef WIN32
    #include <windows.h>
#endif
#include<stdlib.h>
#include<stdio.h>
#include<stddef.h>
#include<string.h>
#include<time.h>
#include "enm4sxit.h"
/* defines that this module can be compiled with Cset/2 and IBM C/2 */
#ifdef __32BIT__
    #define APIENTRY16 APIENTRY
    #define PCHAR16 PCHAR
#endif
unsigned char Enm36Table[36]= {'\x00', '\x01', '\x02', '\x03',
                               '\x04', '\x05', '\x06', '\x07',
                               '\x08', '\x09', '\x0A', '\x0B',
                               '\x0C', '\x0D', '\x0E', '\x0F',
                               '\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)
{

```

Figure 24. Sample Security User Exit ENM4SSEC.C (Part 1 of 3)

```

unsigned long count=0,remainder=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);
    remainder = number % (unsigned long)base;
    count = number / (unsigned long)base;
    if (!count) {
        basestring[position++]=Enm36Table[remainder];
        break;
    }
    basestring[position]=Enm36Table[remainder];
    number = count;
} while (1);
return(0);
}
USHORT APIENTRY16 ENM4ExitMacGen (PUCHAR16 pucApp1Id,
                                PCHAR16 pucBuffer,
                                USHORT usBufferLen,
                                PCHAR16 pucMacBuffer){
    register i;
    unsigned long ulAddedByteValues=0;
    unsigned short rc = 0;

    if (!strcmp(pucApp1Id,"APPLAUTH") || !strcmp(pucApp1Id,"APPLSECR")) {
        for (i=0;i<usBufferLen;i++) {
            ulAddedByteValues += (unsigned long) pucBuffer[i];
        }
        rc = EnmBasestr(2,
                       ulAddedByteValues,
                       pucMacBuffer,
                       32);
    }
    return(rc);
}

USHORT APIENTRY16 ENM4ExitMacVerify (PUCHAR16 pucApp1Id,
                                     PCHAR16 pucBuffer,
                                     USHORT usBufferLen,
                                     PCHAR16 pucMacBuffer)
{
    register i;
    unsigned long ulAddedByteValues=0;
    unsigned char ucaCalcMacBuffer[32];
    unsigned short rc = 0;

    if (!strcmp(pucApp1Id,"APPLAUTH") || !strcmp(pucApp1Id,"APPLSECR")) {
        for (i=0;i<usBufferLen;i++) {
            ulAddedByteValues += (unsigned long) pucBuffer[i];
        }
        memset (ucaCalcMacBuffer,0,32);
    }
}

```

Figure 24. Sample Security User Exit ENM4SSEC.C (Part 2 of 3)

```

        rc = EnmBasestr(2,
                        u1AddedByteValues,
                        ucaCalcMacBuffer,
                        32);
        if (!rc) rc = memcmp(ucaCalcMacBuffer,pucMacBuffer,32);
    }
    return(rc);
}
USHORT APIENTRY16 ENM4ExitEncrypt ( PCHAR16 pucAppId,
                                    PCHAR16 pucBuffer,
                                    USHORT  usBufferLen )
{
    register i;

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

USHORT APIENTRY16 ENM4ExitDecrypt ( PCHAR16 pucAppId,
                                    PCHAR16 pucBuffer,
                                    USHORT  usBufferLen )
{
    register i;

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

```

Figure 24. Sample Security User Exit ENM4SSEC.C (Part 3 of 3)





---

## Appendix C. Sample Programs

This appendix includes listings of sample MERVA Connection/2 programs written in C.

---

## Program SMPLO1

```
/* ***** Sample program for MERV Connection/2 ***** */
/*
/* PROGRAM NAME: SMPLO1
/* -----
/* Sample API application for loading / unloading messages to/from API queues*/
/*
/* COPYRIGHT:
/* -----
/* (C) Copyright International Business Machines Corporation 1993
/*
/* REVISION LEVEL: 1.0
/* -----
/*
/*
/* WHAT THIS PROGRAM DOES:
/* -----
/* This program demonstrates the use of API calls of MERV Connection/NT/2
/* to either load messages to API queues or unload them from queues to a
/* file member. The function to execute is specified on the commandline as
/* first parameter followed by the name of the member to be used.
/*
/* Load Messages: Messages are loaded from the member to the API_IN queue
/* and are routed as defined by the routing that had been
/* set up for this queue.
/*
/* Unload Messages: Messages in the API_OUT queue are exported to the member.*
/* Note: This function does NOT check if new messages
/* enter the queue while the messages are unloaded.
/* So there can be messages left in the queue if
/* they were routed/added to the queue after the
/* unload started.
/*
/*
/* FILE MEMBERS NEEDED TO COMPILE:
/* -----
/* SMPLO1.C - This file
/* ENMRAPI.H - The API include file
/*
/* PROGRAMS TO BE LINKED:
/* -----
/* ENMORAPI.LIB - Remote MERV API library
/*
/*
/* EXPECTED INPUT:
/* -----
/* The user is expected to supply the function to perform (l or u) and the
/* name of the member where the messages are to be stored or loaded from.
/*
/* EXPECTED OUTPUT:
/* -----
/* If "unload" was selected:
/* all messages that were in the queue API_OUT when the program was
/* started, are written to the member.
/*
/*
```

Figure 25. Sample Program SMPLO1.C (Part 1 of 7)

```

/* If "load" was selected: */
/* all messages from the member are added to the API_IN queue and */
/* routed as defined by the routing table of MERVA OS/2 V3. */
/* */
/* */
/* MERVA OS/2 V3 CALLS USED: */
/* ----- */
/* ENMAttach - Attach to MERVA OS/2 */
/* ENMDetach - Detach from MERVA OS/2 */
/* ENMCreate - Create a new message */
/* ENMWriteField - Set specific fields (telexheader) in a message */
/* ENMRouteAdd - Route a message from a source to a destination queue */
/* ENMQueryQueue - Query queue information */
/* ENMNextEntry - Get next message from queue */
/* ENMDelete - Delete message from queue */
/* */
/* ADDITIONAL MERVA Connection/2 CALLS USED: */
/* ----- */
/* ENMSetProfile - Select the profile to be used */
/* ENMStartAPPC - Establish connection to MERVA OS/2 V3 */
/* ENMEndAPPC - Disconnect from MERVA OS/2 V3 */
/* ENMGetReason - Get reason code for internal error */
/* */
/*****
#include <stdlib.h>
#include <string.h>
#include <stdio.h>
#include <stddef.h>
#include <stdarg.h>
#include <time.h>
#include "enmrapi.h"

#define MSGSIZE 28000

/* Function prototypes for the used functions. */
USHORT load_msgs ( FILE *, QNAME );

USHORT unload_msgs ( QNAME , FILE * );

USHORT read_msg ( char *, USHORT *, FILE * );

FILE *OpenReadFile ( UCHAR * );

FILE *OpenWriteFile ( UCHAR *, USHORT );

USHORT ReadRecord ( FILE *, UCHAR *, USHORT, USHORT * );

USHORT WriteRecord ( FILE *, UCHAR *, USHORT );

/* The main entry of the program. */
int main(int argc, char *argv[])
{
    FILE *pFile; /* handle for the file to be used */
    USHORT rc = 0; /* returncode of API calls */
    SHORT rs = 0; /* addit. return-, reasoncode */

```

Figure 25. Sample Program SMPLO1.C (Part 2 of 7)

```

printf("\n ----- Merva OS/2 V3 API Sample ----- \n" );
printf("\n --- Loading/unloading messages through API queues of ----- \n" );
printf("\n ----- Merva OS/2 V3 ----- \n\n" );

if ( argc != 3 ) {
    printf("\n Usage: SMPLO1 {l|u} member\n");
    printf("\n l | u:\n");
    printf( " either one or the other option has to be specified:\n");
    printf( " l: load messages from a member to the API_IN queue\n");
    printf( " u: unload messages from the API_OUT queue to a member\n");
    printf("\n member:\n");
    printf( " name of file member to read from or write to messages.\n");
    return( -1 );
}

switch ( argv[1][i0] )
{
    case 'l':
    case 'L':
        /* If option is L => load messages from file member, */
        /* open member for read access. */
        pFile = OpenReadFile( argv[2] );
        if ( pFile == NULL ) {
            printf( "\nCould not open the file '%s'\n",argv[2]);
            return( -2 );
        }
        break;
    case 'u':
    case 'U':
        /* If option is U => unload messages to file member, */
        /* open member for write access. */
        pFile = OpenWriteFile( argv[2], 1024 );
        if ( pFile == NULL ) {
            printf( "\nCould not open the file '%s'\n",argv[2]);
            return( -2 );
        }
        break;
    default:
        /* Invalid parameter specified. */
        printf( "\nParameter 1, invalid value: '%s'\n", argv[1] );
        return( -3 );
}

ENMSetProfile ("SAMPLE.PRF");

if ((rc = ENMStartRAPI ( "SMPLO1" )) == 0 ) {
    printf("APPC or TCP/IP Conversation is up\n");

    /* Now Attach to MERVA OS/2 using the UserID SAMPLE */
    /* and call the appropriate function depending on option selected. */
    rc = ENMAttach( "SAMPLE", "SAMPLE1", "API" );
    if ( rc == NO_ERROR ) {
        printf( "\nProgram attached to Merva...\n" );
    }
}

```

Figure 25. Sample Program SMPLO1.C (Part 3 of 7)

```

switch( argv[1] )
{
    case 'l':
    case 'L':
        rc = load_msgs( pFile, "API_IN" );
        break;
    case 'u':
    case 'U':
        rc = unload_msgs( "API_OUT", pFile );
        break;
}
/* Close the file member. */
fclose( pFile );

/* Now do the detach from MERVA OS/2. */
rc = ENMDetach();
if ( rc == NO_ERROR )
    printf( "\n... Program detached\n" );
else
    printf( "\nCould not detach from MERVA, rc = %d\n", rc );
}
else
    printf( "\nCould not attach to MERVA, rc = %d\n", rc );

if ((rc = ENMEndAPPC ()) != 0)
    printf( "\nCould not stop APPC, rc = %d\n", rc );

}
else {
    rs = ENMGetReason();
    printf("Error in ENMStartRAPI, rc = %d, rs = %d\n", rc,rs );
}

return( rc );
}

/*****
/* Function      : load_msgs
/* What it does: This function loads the messages from the given file
/*               and does a minimal check if the {l: is included in the
/*               message (Basic header). It then adds and routes the
/*               message to API_IN queue and prints a dot on the screen
/*               to show that the message was added.
*****/
USHORT load_msgs( FILE *pFile, QNAME qnQueue )
{
    USHORT   rc = 0;                /* returncode of API calls
    MMSG     Message;              /* Actual storage of the message
    CHAR     msgTxt[MSG_SIZE];     /* storage for 1 message (max length)
    FIELD    fldAssociated;        /* field for network identifier
    USHORT   msgcnt=0;            /* number of messages loaded so far
    USHORT   msgLngth;            /* actual length of message

    fldAssociated.msgnet = NET_SWIFT;
    strcpy(fldAssociated.msgmac, "MAC!");

```

Figure 25. Sample Program SMPLO1.C (Part 4 of 7)

```

printf("\nLoading messages \n");

while ((rc = ReadRecord( pFile, msgTxt, 1024, &msgLngth)) == 0) {
    if ( memcmp( msgTxt, "{1:", 3 ) != 0 ) {
        printf( "\nMessage %d header is not SWIFT II format\n", msgcnt+1 );
        printf( "\n%i messages loaded.\n", msgcnt );
        return( rc );
    }
    /* Create a new message. */
    rc = ENMCreate( &Message );
    if ( rc != NO_ERROR ) {
        printf( "\nCould not create a message, rc = %d\n", rc );
        printf( "\n%i messages loaded.\n", msgcnt );
        return( rc );
    }
    /* Set the destination network to 'SWIFT network'. */
    rc = ENMWriteField( FLD_MSGNET, &fldAssociated );
    if ( rc != NO_ERROR ) {
        printf( "\nCould not set destination network, rc = %d\n", rc );
        printf( "\n%i messages loaded.\n", msgcnt );
        return( rc );
    }
    /* Set MAC field to test-message */
    rc = ENMWriteField( FLD_MSGMAC, &fldAssociated );
    if ( rc != NO_ERROR ) {
        printf( "\nCould not set MAC field, rc = %d\n", rc );
        printf( "\n%i messages loaded.\n", msgcnt );
        return( rc );
    }
    /* Copy the message read to the created message area */
    /* and call ENMRouteAdd to add it to the system. */
    memcpy( Message, msgTxt, msgLngth );
    rc = ENMRouteAdd( qnQueue );
    if ( rc != NO_ERROR ) {
        printf( "\nCould not add message to queue, rc = %d\n", rc );
        printf( "\n%i messages loaded.\n", msgcnt );
        return( rc );
    } else {
        putchar('.');
        msgcnt++;
    }
}
printf( "\n%i messages loaded.\n", msgcnt );
return( NO_ERROR );
}

/*****
/* Function      : unload_msgs
/* What it does: This function determines the number of messages in the
/*               queue and unloads them to the file member. After
/*               unloading, the messages are deleted from the queue.
*****/
USHORT unload_msgs( QNAME  qnQueue, FILE  *pFile )
{

```

Figure 25. Sample Program SMPLO1.C (Part 5 of 7)

```

USHORT rc = 0;                /* returncode of API calls      */
USHORT usMsg_count = 0;      /* number of messages in the queue */
USHORT usMsg_length = 0;    /* individual length of message */
USHORT usCount = 0;        /* Count of unloaded messages so far */
MMSG Message;              /* Actual storage of the message */

printf( "\nUnloading messages \n" );

/* Query the number of messages in the queue. */
rc = ENMQueryQueue( qnQueue, &usMsg_count );
if ( rc != NO_ERROR ) {
    printf( "\nCould not query number of messages in queue %s, rc = %d\n",
            qnQueue, rc );
    return( rc );
}
/* Read messages from the queue with ENMNextEntry. */
for (usCount = 0; usCount < usMsg_count; usCount++)
{
    rc = ENMNextEntry( qnQueue, ON, &Message, &usMsg_length );
    if ( rc != NO_ERROR ) {
        printf( "\nCould not retrieve message #%i out of %i total.\n",
                usCount+1, usMsg_count );
        return( rc );
    }
    if ((rc = WriteRecord( pFile, Message, 1024)) != 0) {
        printf( "\nCould not write message #%i to file member.\n", usCount+1 );
        if ((rc = ENMFree()) != 0)
            printf( "\nCould not free message #%i.\n", usCount+1 );
        return( -5 );
    }
    /* Message successfully written to file member , so send dot */
    /* to the screen and delete the message from the queue. */
    putchar('.');
    rc = ENMDelete();
    if ( rc != NO_ERROR ) {
        printf( "\nCould not delete message #%i from queue, rc = %d\n",
                usCount, rc );
        return( rc );
    }
}
printf( "\n%i messages unloaded.\n", usCount );
return( NO_ERROR );
}

/*****
/* Function      : ReadRecord
/* What it does: Read one record from the file member.
/* Arguments   : pFile (FILE*)          - file member handle
/*              pucBuffer (unsigned char*) - data buffer
/*              usRecLength (unsigned short) - expected record length
/*              pusLength (unsigned short*) - ret. string length of buffer */
*****/
USHORT ReadRecord( FILE *pFile,    UCHAR *pucBuffer,
                  USHORT usRecLength, USHORT *pusLength )
{

```

Figure 25. Sample Program SMPLO1.C (Part 6 of 7)

```

        if (fread( pucBuffer, usRecLength, 1, pFile ) == 1) {
            *pusLength = strlen(pucBuffer);
            return( 0 );
        } else {
            return( 12 );
        }
    }
}

/*****
/* Function      : WriteRecord                               */
/* What it does: Read one record from the file member.      */
/* Arguments    : pFile (FILE*)                             - file member handle */
/*              : pucBuffer (unsigned char*)                - data buffer         */
/*              : usRecLength (unsigned short)              - record length to write */
*****/
USHORT WriteRecord( FILE *pFile, UCHAR *pucBuffer, USHORT usRecLength)
{
    if ( fwrite( pucBuffer, usRecLength, 1, pFile ) == 1) {
        return( 0 );
    } else {
        return( 12 );
    }
}

/*****
/* Function      : OpenReadFile                               */
/* What it does: Opens file member for read access.         */
/* Arguments    : pFile (FILE*)                             - file member handle */
*****/
FILE *OpenReadFile( UCHAR *pFile)
{
    return(fopen( pFile, "rb" ));
}

/*****
/* Function      : OpenWriteFile                               */
/* What it does: Opens file member for write access.        */
/* Arguments    : pFile (FILE*)                             - file member handle */
/*              : usRecLength (unsigned short)              - record length to write */
*****/
FILE *OpenWriteFile( UCHAR *pFile, USHORT usRecLength)
{
    UCHAR ucaOpenStringi40;

    sprintf( ucaOpenString, "wb, lrecl=%d, recfm=f, type=record", usRecLength);
    return(fopen( pFile, ucaOpenString ));
}

```

Figure 25. Sample Program SMPLO1.C (Part 7 of 7)



---

## Program SMPLO2

```
/****** Trigger sample program for MERVA Connection/2 *****/
/*
/* PROGRAM NAME: SMPLO2
/* -----
/* Sample API application with triggering by MERVA OS/2
/*
/* COPYRIGHT:
/* -----
/* (C) Copyright International Business Machines Corporation 1994
/*
/* REVISION LEVEL: 1.0
/* -----
/*
/* WHAT THIS PROGRAM DOES:
/* -----
/* This program demonstrates the triggering concept of MERVA OS/2. It will
/* monitor the Queue API_OUT for messages coming into the queue and will
/* write them to the file specified on the command line. It does the
/* monitoring by using semaphores that are cleared by MERVA OS/2 instead of
/* continuously polling if there are messages in the queue.
/* The program checks first if there are messages in the queue and will
/* unload them to the file. Then it waits on a semaphore until new messages
/* enter the queue. When new messages are entered, MERVA OS/2 will clear the
/* semaphore and the program can continue reading messages from the queue
/* until all messages are processed.
/* This processing loop is ended by starting the program SAMPLE2S in another
/* OS/2 session.
/*
/*
/* WHAT YOU NEED TO COMPILE THIS PROGRAM:
/* -----
/*
/* REQUIRED FILES:
/* -----
/* SMPLO2.C - This file
/* ENMRAPI.H - The API include file
/*
/* OS2.H - OS/2 System include File or
/* STDLIB.H - Miscellaneous function declarations
/* STRING.H - String function declarations
/* STDIO.H - Input/Output function declarations
/*
/* REQUIRED LIBRARIES:
/* -----
/* OS2386.LIB - OS/2 library
/* CPPOM30.LIB - Visual Age for C++ Library (or equivalent)
/* ENMRAPI.LIB - MERVA OS/2 Application Programming Interface library
/*
/* REQUIRED PROGRAMS:
/* -----
/* Visual Age for C++ (or equivalent)
/*
/* Compiler & Linker options (Visual Age for C++)
/* -----
/* /C /DOS2 /O+ /N2 /Gd+ /Gm+ /Gs- /Gt+
/* /NOE /STACK:65536
```

Figure 26. Sample Program SMPLO2.C (Part 1 of 6)

```

/* */
/* */
/* EXPECTED INPUT: */
/* ----- */
/* The user is expected to supply a filename where the unloaded message */
/* are stored in. */
/* */
/* EXPECTED OUTPUT: */
/* ----- */
/* All messages that where in the queue API_OUT are written to the file */
/* while the program is running. */
/* */
/* */
/* CALLS USED (Dynamic Link References): */
/* ----- */
/* */
/* ENMNextEntry - MERVA OS/2 API function */
/* */
/* ENMCreateSem \ */
/* ENMSetSem \ */
/* ENMWaitSemList -- Remote API Semaphore functions */
/* ENMCloseSem / */
/* ENMClearSem / */
/* */
/*****

/* Define the following values to include the OS/2 Base services for */
/* Semaphores and error codes. */
#define INCL_BASE
#define INCL_DOSSEMAPHORES
#define INCL_DOSERRORS
#include <os2.h>
#include <stdlib.h>
#include <string.h>
#include <stdio.h>
#include "enmrapi.h"

/* The following define is the semaphore that will be cleared by MERVA OS/2 */
/* when Messages enter the queue. Please be sure to set this name (SAMPLE2) */
/* in the MERVA OS/2 Customizer Alarm definition for the queue API_OUT, */
/* where the messages are unloaded from. */

/* You may either use semaphore names with the \SEM\ prefix or without, */
/* but for portability reasons, we recommend the new form without prefix. */

#define TRIGGER "SAMPLE2"
#define STOP "SAMPLE2"

/* Function prototype for the unload_msgs function. */
SHORT unload_msgs( QNAME qnQueue, FILE *pFile );

/* The main entry of the program. */
int main(int argc, char *argv[])
{
    FILE *pFile; /* handle for the file to be used */

```

Figure 26. Sample Program SMPLO2.C (Part 2 of 6)



```

    }
} else {
    printf("Error in ENMStartRAPI, rc = %d, rs = %d\n", rc, ENMGetReason() );
}

return ( rc );
}

/*****
/* Function      : unload_msgs                               */
/* What it does: It contains the loop to retrieve the messages from the */
/*               queue and write them to the file. If no messages are */
/*               currently in the queue, it will wait to be triggered by */
/*               MERVAs OS/2.                                         */
*****/
SHORT unload_msgs( QNAME  qnQueue, FILE  *pFile )
{
    USHORT    usMsg_length;          /* Msg length returned by NextEntry */
    USHORT    usCount;              /* Number of messages unloaded */
    MMSG      Message;              /* Actual storage of the message */
    CHAR      lengthi4;             /* length field for msgs in the file */
    USHORT    rc;                   /* Returncode of functions */
    ULONG     SemTrigger;           /* Handle to Trigger semaphore */
    ULONG     SemStop;             /* Handle to Stop Semaphore */
    USHORT    usIndex = 0;          /* Index of semaphore cleared */
    BOOL      fStop = FALSE;        /* Boolean variable to control loop */
    FILE

    /* Create the trigger semaphore. This one will be cleared by MERVAs OS/2. */
    /* It has to be PUBLIC so that MERVAs OS/2 can access the semaphore. */
    rc = ENMCreateSem( &SemTrigger, TRIGGER );
    if ( rc ) {
        printf( "\nError %d, reason %d while creating semaphore %s\n",
                rc, ENMGetReason(), TRIGGER );
        return( rc );
    }
    /* Create the 'Stop' semaphore that will be used to stop the program */
    /* from another session. It is also public. */
    rc = ENMCreateSem( &SemStop, STOP );
    if ( rc ) {
        printf( "\nError %d, reason %d while creating semaphore %s.\n",
                rc, ENMGetReason(), STOP );
        return( rc );
    }
    /* Now set the 'Stop' semaphore so that the process can wait till */
    /* it is cleared from the stop program, SAMPLE2S. */
    rc = ENMSetSem( SemStop );
    if ( rc ) {
        printf( "\nError %d, reason %d while setting semaphore %s.\n",
                rc, ENMGetReason(), STOP );
        return( rc );
    }
}
usCount = 0;

```

Figure 26. Sample Program SMPLO2.C (Part 4 of 6)

```

rc = 0;
do {
    /* Try to read the next message in the queue, lock the message */
    /* in order to be able to delete it after writing. */
    rc = ENMNextEntry( qnQueue, 0N, &Message, &usMsg_length );
    if ( rc == NO_ERROR ) {
        /* Print message only the first time, then only fullstops. */
        if ( usCount == 0 ) printf( "Unloading messages, please wait\n" );
        usCount++;
        /* We found one or more message(s) in the queue, so unload them */
        /* to the file. First write the length field (4bytes) and then */
        /* the total message to the file. */
        *(int *)&lengthi0 = usMsg_length + 4;
        *(int *)&lengthi2 = 0;
        if ( fwrite(length,4,1,pFile) != 1 ) {
            printf( "\nWrite error\n" );
            return( -4 );
        }
        if ( fwrite( Message, usMsg_length, 1, pFile) != 1 ) {
            printf( "\nCould not write message #%i to file.\n", usCount );
            return( -5 );
        }
        /* Message succesfully written to file, so send dot */
        /* to the screen and delete the message from the queue. */
        putchar( '.', out );
        fflush( out );
        rc = ENMDelete();
        if ( rc != NO_ERROR ) {
            printf( "\nCould not delete message #%i from queue, rc = %d\n",
                usCount, rc );
            return( rc );
        }
    } else {
        if ( rc != ERR_MSG_NOT_FOUND ) {
            /* If error in NextEntry call, print it here and leave. */
            printf( "Error in ENMNextEntry call, rc = %d\n", rc );
            /* Try to free the locked message before exiting the function */
            (void) ENMFree();
            return( rc );
        } else {
            printf( "\n%i Messages unloaded, Waiting to be triggered.\n"
                "(Start SMPLO2S in another session to stop)\n",
                usCount );
            usCount = 0;
            /* Set the trigger semaphore so that we can wait on it. */
            rc = ENMSetSem( SemTrigger );
            if ( rc ) {
                printf( "\nError %d, reason %d while setting semaphore %s.\n",
                    rc, ENMGetReason(), STOP );
                return( rc );
            }
        }
    }
}
/* Now wait on the two semaphores to be cleared. The wait is an */
/* undefined wait to block processing until one semaphore is cleared. */

```

Figure 26. Sample Program SMPLO2.C (Part 5 of 6)

```

rc = ENMWaitSemList(&usIndex, /* ptr to usIndex, triggering Sem */
                    -1L,      /* wait indefinitely */
                    SemStop,  /* first Semaphore */
                    SemTrigger, /* second Semaphore */
                    0L);      /* 0 signals last parameter */
if ( rc == 0 )
    /* If the first semaphore in the list is cleared, the program */
    /* was asked to shut down, so indicate the end of the loop. */
    if ( usIndex == 0 ) fStop = TRUE;
} while( !fStop );

/* Issue an ending message, clear the used semaphores and close them */
/* If messages were unloaded, but no info messages so far, do one */
/* last message. */
if ( usCount > 0 )
    printf( "\n%i Messages unloaded.\n", usCount );

printf( "Program stopped via SMPLO2S from another session.\n" );
ENMClearSem( SemTrigger );
ENMClearSem( SemStop );
ENMCloseSem( SemTrigger );
ENMCloseSem( SemStop );
return( 0 );
}

```

Figure 26. Sample Program SMPLO2.C (Part 6 of 6)

---

## Program SMPLO2S

```
/****** Trigger sample Stop program for MERVA/2 *****/
/*
/* PROGRAM NAME: SMPLO2S
/* -----
/* Stop program for the MERVA trigger sample program SMPLO2.EXE
/*
/* COPYRIGHT:
/* -----
/* (C) Copyright International Business Machines Corporation 1994
/*
/* REVISION LEVEL: 1.0
/* -----
/*
/* WHAT THIS PROGRAM DOES:
/* -----
/* This program stops the trigger sample program by clearing the stop
/* semaphore. This program has to be started in another session while the
/* trigger sample program is running.
/*
/* WHAT YOU NEED TO COMPILE THIS PROGRAM:
/* -----
/*
/* REQUIRED FILES:
/* -----
/*
/* SMPLO2S.C - This file
/*
/* OS2.H - OS/2 System include File
/* STDLIB.H - Miscellaneous function declarations
/* STRING.H - String function declarations
/* STDIO.H - Input/Output function declartions
/*
/* REQUIRED LIBRARIES:
/* -----
/*
/* OS2386.LIB - OS/2 library
/*
/* REQUIRED PROGRAMS:
/* -----
/*
/* IBM Visual Age for C++
/*
/* EXPECTED INPUT:
/* -----
/* No input is expected.
/*
/* EXPECTED OUTPUT:
/* -----
/* The box indicating that the Stop program was started.
/* Any errors are written to the screen.
/*
/* CALLS USED (Dynamic Link References):
/* -----
/*
/* ENMOpenSem \
/* ENMclearSem -- Remote API Semaphore functions
/* ENMcloseSem /
/******
```

Figure 27. Sample Program SMPLO2S.C (Part 1 of 2)





---

## Program SMPLO3

```
/****** Sample program for MERV Connection/2 or Connection/NT *****/
/*
/* PROGRAM NAME: SMPLO3
/* -----
/* Sample API application for loading telex messages to API_IN queue.
/*
/* COPYRIGHT:
/* -----
/* (C) Copyright International Business Machines Corporation 1992
/*
/* REVISION LEVEL: 1.0
/* -----
/*
/*
/* WHAT THIS PROGRAM DOES:
/* -----
/* This program demonstrates the use of API calls of MERV OS/2 to load
/* telex messages to an API queue (API_IN).
/*
/* Telexes in a special format (created with a simple editor) were loaded
/* from the specified file to the API_IN queue.
/* It's possible to load SWIFT messages in TMP/2 format, messages in Telex
/* Link format and negatively acknowledged telexes in the free format.
/*
/*
/* WHAT YOU NEED TO COMPILE THIS PROGRAM:
/* -----
/*
/* REQUIRED FILES:
/* -----
/* SMPLO3.C - This file
/* ENMRAPI.H - The API include file
/*
/* STDLIB.H - Miscellaneous function declarations
/* STRING.H - String function declarations
/* STDIO.H - Input/Output function declarations
/*
/* REQUIRED LIBRARIES:
/* -----
/* OS2386.LIB - OS/2 library
/* DOSCALLS.LIB - OS/2 library
/* CPPOM30.LIB - Visual Age for C++ standard multithreaded lib
/* ENMRAPI.LIB - MERV/2 Application Programming Interface library
/*
/* REQUIRED PROGRAMS:
/* -----
/* Visual Age for C++
/*
/*
/* EXPECTED INPUT:
/* -----
/* The user is expected to supply the filename where the messages are
/* loaded from. The file should contain the telex headers of one or more
/* messages in a special format: see the example file 'SAMPLE3.DAT'.
/*
/*
```

Figure 28. Sample Program SMPLO3.C (Part 1 of 9)



```

if ( argc != 2 ) {
    printf("\n      Usage: SMPLO3 file\n");
    printf("\n      file: name of file in local directory\n");
    printf( "      containing the telex headers of\n");
    printf( "      one or more messages in special\n");
    printf( "      format (see 'SMPLO3.DAT').      \n");
    return( -1 );
}
pFile = fopen( argv[1], "rb" );
if ( pFile == NULL ) {
    printf( "\nCould not open the file '%s'\n", argv[1] );
    return( -2 );
}

ENMSetProfile ("SAMPLE.PRF");      /* pass profile to enmrapi      */

if ((rc = ENMStartRAPI ( "SAMPLE3" )) == 0 ) {
    /* Attach to MERVA/2 using the USERID SAMPLE. */
    rc = ENMAttach( "sample", "sample1", "API" );
    if ( rc != NO_ERROR ) {
        printf( "\nCould not attach to MERVA, rc = %d\n", rc );
    } else {
        printf( "\nProgram attached to MERVA...\n" );

        strcpy( qnQueue1, "API_IN" );
        strcpy( qnQueue2, "API_OUT" );

        /* Create messages, read the MRN, write the network id 'NET_TELEX',      */
        /* read the tags and data for the telex header, write the telex header */
        /* and then add the message until 'end of file' or an error occurred. */
        while ( rc == NO_ERROR && !eof ) {

            printf( "\nCreating a telex message: " );

            if ( rc == NO_ERROR ) {
                /* Create a new message. */
                rc = ENMCreate( &message );
                if ( rc != NO_ERROR )
                    printf( "Could not create a message, rc = %d", rc );
            }
            if ( rc == NO_ERROR ) {
                /* Read the MRN of the created message. */
                rc = ENMReadField( FLD_MRN, (PPFIELD)&pField );
                if ( rc != NO_ERROR )
                    printf( "Could not read field MRN, rc = %d", rc );
                else
                    strncpy( mrn, pField->mrn, MRNlen+1 );
            }
            if ( rc == NO_ERROR ) {
                /* Set the destination network to 'Telex network'. */
                pField->msgnet = NET_TELEX;
                rc = ENMWriteField( FLD_MSGNET, pField );
                if ( rc != NO_ERROR )
                    printf( "Could not set destination network, rc = %d", rc );
            }
        }
    }
}

```

Figure 28. Sample Program SMPLO3.C (Part 3 of 9)



```

        if ( rc != NO_ERROR )
            printf( "Could not add message to queue, rc = %d", rc );
    }
    if ( rc == NO_ERROR )
        printf( "    MRN = %s", mrn );
    else {
        /* Clear the message area if there was an error. */
        rc = ENMClear();
        if ( rc != NO_ERROR )
            printf( "\nCould not clear the message, rc = %d", rc );
    }
}
/* Close the input file. */
fclose( pFile );

/* Now do the detach from MERVA. */
rc = ENMDetach();
if ( rc == NO_ERROR ) printf( "\n\n... Program detached\n\n" );
else printf( "\n\nCould not detach from MERVA, rc = %d\n\n", rc );
}

rc = ENMEndRAPI();
if (rc)
    printf( "\n\nError ending the conversation, rc = %d\n\n");
}
else {
    rs = ENMGetReason();
    printf("Error in ENMStartRAPI, rc = %d, rs = %d\n", rc,rs );
}

return( rc );
}

/*****
/* Function      : read_tag                               */
/* What it does: Reads the tag from the input file.      */
/*              At first search the next colon, assuming that's the      */
/*              beginning of the tag. If found, read the next characters  */
/*              until the 2nd colon or the max.size of tag (TAGSIZE) is  */
/*              reached. If OK, return NO_ERROR.         */
*****/
SHORT read_tag( char *tag, FILE *pFile )
{
    int ch;
    int i=1;

    do {
        ch = fgetc( pFile );
    } while ( ch != EOF && (char)ch != ':' );

    if ( (char)ch == ':' ) {
        do {
            *(tag++) = (char)ch;
            if ( i == TAGSIZE ) {
                *(tag++) = '\0';
            }
        } while ( (char)ch != ':' );
    }
}

```

Figure 28. Sample Program SMPLO3.C (Part 5 of 9)

```

        return( 1 );
    }
    i++;
    ch = fgetc( pFile );
} while ( ch != EOF && ch != 10 && ch != 13 && (char)ch != ':' );

if ( (char)ch == ':' ) {
    *(tag++) = (char)ch;
    *(tag++) = '\0';
    return( NO_ERROR );
}
}
if ( ch == EOF ) return( EOF );
else return( 2 );
}

/*****
/* Function      : read_data
/* What it does: Reads the data from the input file.
/*              For all tags describing a field of the telex header the
/*              associated data is read from the input file.
/*              The data will be read until the 'end of file', a carriage
/*              return (0x0D), a line feed (0x0A) or a '|' (0x7C) appears.
/*              To read the message data (tag :MESSAGE:), the function
/*              'read_msg' will be called instead of this one.
*****/
SHORT read_data( char *data, FILE *pFile )
{
    int ch;
    int i=1;

    ch = fgetc( pFile );

    /* Read while not 'end of file', not CR LF and not '|' (0x7C) */
    /* ('|' indicates the end of the data for this tag.) */
    while ( ch != EOF && ch != 10 && ch != 13 && (char)ch != '|' )
    {
        if ( i > DATSIZE ) {          /* Max.size of data reached, so stop.*/
            *(data++) = '\0';
            return( 1 );
        }
        *(data++) = (char)ch;         /* Build the data. */
        i++;
        ch = fgetc( pFile );
    }
    *(data++) = '\0';
    return( NO_ERROR );
}

/*****
/* Function      : read_msg
/* What it does: Reads the message from the input file.
/*              For the tag :MESSAGE: the associated data is read from the
/*              input file.
/*              The data will be read until the 'end of file' (0x1A) or
/*              a '|' (0x7C) appears.
*****/

```

Figure 28. Sample Program SMPLO3.C (Part 6 of 9)

```

/*****
SHORT read_msg( char *msg, FILE *pFile )
{
    int ch;
    int i=1;

    ch = fgetc( pFile );

    /* Read while not 'end of file' and not '|' (0x7C) */
    /* ('|' indicates the end of the message.) */
    while ( ch != EOF && ch != 26 && (char)ch != '|' )
    {
        if ( i > MSGSIZE ) {          /* Max.size of msg reached, so stop. */
            *(msg++) = '\0';
            return( 1 );
        }
        *(msg++) = (char)ch;          /* Build the message. */
        i++;
        ch = fgetc( pFile );
    }
    *(msg++) = '\0';
    return( NO_ERROR );
}

/*****
/* Function      : build_txhead
/* What it does: Builds the telex header.
/*              Depending on the tag, the associated field of the telex
/*              header will be filled with the data read from input file.
/*****
SHORT build_txhead( PFIELD pfield, char *tag, char *data )
{
    if ( strcmp( tag, ":KEY_CAL:" ) == 0 ) {
        strncpy( pfield->txhead.testkey_cal, data, 1 );
        return( NO_ERROR );
    }
    if ( strcmp( tag, ":KEY_RC:" ) == 0 ) {
        strncpy( pfield->txhead.testkey_rc, data, 1 );
        return( NO_ERROR );
    }
    if ( strcmp( tag, ":KEY_VAL:" ) == 0 ) {
        strncpy( pfield->txhead.testkey_val, data, TESTKEYlen );
        return( NO_ERROR );
    }
    if ( strcmp( tag, ":KEY_TXT1:" ) == 0 ) {
        strncpy( pfield->txhead.testkey_comment1, data, TEST_COMMLen );
        return( NO_ERROR );
    }
    if ( strcmp( tag, ":KEY_TXT2:" ) == 0 ) {
        strncpy( pfield->txhead.testkey_comment2, data, TEST_COMMLen );
        return( NO_ERROR );
    }
    if ( strcmp( tag, ":S_ADDR0:" ) == 0 ) {
        strncpy( pfield->txhead.sender_addr0, data, ADDRlen );
        return( NO_ERROR );
    }
}

```

Figure 28. Sample Program SMPLO3.C (Part 7 of 9)

```

if ( strcmp( tag, ":S_ADDR1:" ) == 0 ) {
    strncpy( pfield->txhead.sender_addr1, data, ADDR1len );
    return( NO_ERROR );
}
if ( strcmp( tag, ":S_ADDR2:" ) == 0 ) {
    strncpy( pfield->txhead.sender_addr2, data, ADDR1len );
    return( NO_ERROR );
}
if ( strcmp( tag, ":S_ADDR3:" ) == 0 ) {
    strncpy( pfield->txhead.sender_addr3, data, ADDR1len );
    return( NO_ERROR );
}
if ( strcmp( tag, ":DATE:" ) == 0 ) {
    strncpy( pfield->txhead.date, data, DATE1len );
    return( NO_ERROR );
}
if ( strcmp( tag, ":TO_ID:" ) == 0 ) {
    strncpy( pfield->txhead.to_id, data, TO_ID1len );
    return( NO_ERROR );
}
if ( strcmp( tag, ":R_ADDR0:" ) == 0 ) {
    strncpy( pfield->txhead.receiver_addr0, data, ADDR1len );
    return( NO_ERROR );
}
if ( strcmp( tag, ":R_ADDR1:" ) == 0 ) {
    strncpy( pfield->txhead.receiver_addr1, data, ADDR1len );
    return( NO_ERROR );
}
if ( strcmp( tag, ":R_ADDR2:" ) == 0 ) {
    strncpy( pfield->txhead.receiver_addr2, data, ADDR1len );
    return( NO_ERROR );
}
if ( strcmp( tag, ":R_ADDR3:" ) == 0 ) {
    strncpy( pfield->txhead.receiver_addr3, data, ADDR1len );
    return( NO_ERROR );
}
if ( strcmp( tag, ":LINE:" ) == 0 ) {
    strncpy( pfield->txhead.line, data, LINE1len );
    return( NO_ERROR );
}
if ( strcmp( tag, ":DIAL_UP1:" ) == 0 ) {
    strncpy( pfield->txhead.dial_up1, data, DIAL_UP1len );
    return( NO_ERROR );
}
if ( strcmp( tag, ":ANSBACK1:" ) == 0 ) {
    strncpy( pfield->txhead.answ_back1, data, ANSW_BA1len );
    return( NO_ERROR );
}
if ( strcmp( tag, ":DIAL_UP2:" ) == 0 ) {
    strncpy( pfield->txhead.dial_up2, data, DIAL_UP1len );
    return( NO_ERROR );
}
if ( strcmp( tag, ":ANSBACK2:" ) == 0 ) {
    strncpy( pfield->txhead.answ_back2, data, ANSW_BA1len );
    return( NO_ERROR );
}

```

Figure 28. Sample Program SMPLO3.C (Part 8 of 9)



```

}
if ( strcmp( tag, ":TYPE:" ) == 0 ) {
    strncpy( pfield->txhead.type, data, 1 );
    return( NO_ERROR );
}
if ( strcmp( tag, ":T_TIME:" ) == 0 ) {
    strncpy( pfield->txhead.timed_time, data, TIMElen );
    return( NO_ERROR );
}
if ( strcmp( tag, ":T_DATE:" ) == 0 ) {
    strncpy( pfield->txhead.timed_date, data, DATElen );
    return( NO_ERROR );
}
if ( strcmp( tag, ":REF_TEXT:" ) == 0 ) {
    strncpy( pfield->txhead.ref_text, data, REFlen );
    return( NO_ERROR );
}
if ( strcmp( tag, ":NOTE:" ) == 0 ) {
    strncpy( pfield->txhead.note, data, NOTElen );
    return( NO_ERROR );
}
return( 1 );
}

```

*Figure 28. Sample Program SMPLO3.C (Part 9 of 9)*



---

## Appendix D. 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
- Advanced Peer-to-Peer Networking
- AIX
- AIX/6000
- APPN
- AS/400
- C/2
- C/400
- CICS/ESA
- COBOL/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
- RS/6000
- SAA
- Series/1
- Systems Application Architecture
- S/390
- 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 this book. If you do not find the terms you are looking for, refer to *Dictionary of Computing*, New York: McGraw-Hill, 1994, or the *S.W.I.F.T. User Handbook*.

### A

**Advanced Program-to-Program Communication.** A communications architecture that allows transaction programs to exchange information on a peer-to-peer basis. SNA LU 6.2 allows APPC architecture to operate on an SNA network.

**API socket.** MERV Application Programming Interface on the RISC System/6000.

**API executer.** A program of MERV Connection/2 that is installed and runs on OS/2 or AIX. It communicates with the API socket on the RISC System/6000.

**API.** Application program interface.

**application program interface.** (1) A set of run-time routines or system calls that allows an application program to use a particular service provided by either the operating system or another licensed program. (2) The formally defined programming language interface that is between a system control program or a licensed program and the user of the program.

**APPC.** Advanced Program-to-Program Communications.

**APPN.** Advanced Peer-to-Peer Networking. Enhances APPC usability through reduced configuration requirements, dynamic directory searches, route calculation capabilities, and intermediate session routing.

### C

**call.** To activate a program or procedure, usually by specifying the entry conditions and jumping to an entry point.

**Common Programming Interface.** An interface providing languages and services that can be used to develop applications that take advantage of Systems Application Architecture (SAA) consistency.

**Communications Side Information.** An object in CPI Communications containing initialization parameters. These are, for example:

- The name of the partner program (for example, of the API executer) with which a program can establish a conversation

- The name of the logical unit (LU) at the partner program's node, which CPI Communications requires to establish a conversation.

**CPI.** Common Programming Interface.

**CPI Communications.** Provides a consistent application programming interface for applications that require program-to-program communication. The interface makes use of the SNA LU 6.2 protocol to create a rich set of interprogram services.

**CPI-C.** CPI Communications.

**CSI.** Communications Side Information.

**customization.** The process of describing optional changes to defaults of a software program that is already installed on the system and configured so that it can be used.

**customize.** (1) To describe to the system the devices, programs, users, and user defaults for a particular data processing system or network. (2) To describe optional preferences or changes to defaults in a software program that is already installed and configured.

### E

**EPM.** Extended Programming Model.

### H

**handle.** A data structure that is a temporary local identifier for an object. You create a handle by allocating it. You make a handle to identify an object at a specific location by binding it.

### I

**identifier.** (1) A name you use to refer to a data object. An identifier contains some combination of letters, digits, and underscores, but its first character cannot be a digit. (2) In programming languages, a lexical unit that names a language object, such as the name of an array, record, label, or procedure. An identifier usually begins with a letter optionally followed by letters, digits, or other characters. (3) A sequence of bits or characters that identifies a program, device, or system to another program, device, or system.

**include file.** A text file that contains declarations used by a group of functions, programs, or users.

## I

**Input Sequence Number (ISN).** A sequential number that identifies a message sent to the SWIFT network.

## J

**JCL.** Job Control Language.

## L

**logical unit.** (1) A type of network addressable unit that enables end users to communicate with each other and gain access to network resources. (2) In SNA, a port through which an end user accesses the SNA network in order to communicate with another user, and through which the end user accesses the functions provided by system services control points (SSCPs). An LU can support at least two sessions, one with an SSCP and one with another LU, and may be capable of supporting many sessions with other LUs.

**loop.** A sequence of instructions performed repeatedly until an ending condition is reached.

**LU.** Logical unit.

## M

**MAC.** Message Authentication Code.

**Message Authentication Code.** A code of a specific length that is calculated with a particular algorithm from a message buffer. It is sent with the message. The partner recalculates it and compares it with the received code. This allows modifications of the transferred data to be detected.

**Message Reference Number.** A unique 16-digit number assigned by MERVA to each message for identification purposes. The message reference number consists of an 8-digit domain identifier and an 8-digit sequential number.

**MRN.** Message Reference Number.

## N

**node.** An end point of a link, or a junction common to two or more links in a network. Nodes can be processors, controllers, or workstations, and they can vary in routing and other functional capabilities.

## P

**partner.** In data communications, the remote application program or the remote computer.

**peer-to-peer communications.** Pertaining to data communications between two nodes that have equal status in the interchange. Either node can begin the conversation.

## R

**Reduced Instruction Set Computer.** A class of computer designs that uses a relatively small set of frequently used instructions that execute in one cycle.

**RISC.** Reduced Instruction Set Computer.

**RISC System/6000.** A family of workstations and servers based on POWER Architecture. They are primarily designed for running multi-user numerical computing applications that use the UNIX operating system.

## S

**semaphore.** (1) Entity used to control access to system resources. Processes can be locked to a resource with semaphores if the processes follow certain programming conventions. (2) Provides a general method to synchronize two processes.

**SMIT.** System Management Interface Tool.

**SNA.** System Network Architecture.

**System Network Architecture.** (1) An architecture for controlling the transfer of information in a data communications network. (2) 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

**TCP/IP.** Transmission Control Protocol/Internet Protocol. A communications subsystem that allows you to set up local area and wide area network.



---

# 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

Most operating and system software documentation is available online. Check the corresponding product packages (OS/2 Warp, OS/2 Warp Connect, OS/2 Server, Database 2, Database Server, Communication Server, Personal Communications) for available online books. Some of the online books are available in printed form, too. The following books might be useful:

- *IBM DATABASE 2 Information and Concepts Guide for Common Server*, S20H-4664
- *IBM DATABASE 2 Administration Guide for Common Server*, S20H-4580
- *IBM DATABASE 2 OS/2 Command Reference for Common Server*, S20H-4645
- *IBM DATABASE 2 for OS/2 Planning Guide*, S20H-4784
- *IBM DATABASE 2 for OS/2 Installation and Operation Guide*, S20H-4785
- *IBM Communications Server Version 4.1 Up and Running !*, GC31-8189
- *IBM Personal Communications Version 4.1 for OS/2 Up and Running !*, GC31-8258
- *IBM TCP/IP Version 2 for OS/2 Installation and Administration*, SC31-6075

---

## **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

## A

- activating security user exits 57, 58, 60
- API
  - building programs 55
  - Remote MERVA API Client 3
  - Remote MERVA API Server 3
- API functions (C) 33, 34
  - data types 29
  - ENMClearSem 38
  - ENMCloseSem 36
  - ENMCreateSem 39
  - ENMEndRAPI 33
  - ENMGetReason 42
  - ENMOpenSem 41
  - ENMRestartRAPI 32
  - ENMSetProfile 30
  - ENMSetSem 37
  - ENMStartRAPI 31
  - ENMWaitSemList 35
- authentication 49

## C

- Communications Server
  - installing sample configuration files 21
- connection to MERVA AIX
  - disconnecting 33
  - reconnecting remote program 32
  - starting 31
- connection to MERVA OS/2
  - disconnecting 33
  - reconnecting remote program 32
  - starting 31
- conversation to MERVA AIX
  - ending 30
  - starting 30
- conversation to MERVA OS/2
  - ending 30
  - starting 30

## D

- decryption
  - user exit for 52
- diagnosis log
  - on the MERVA AIX side 62
  - on the MERVA OS/2 side 62
  - on the remote application side 61
- disconnecting from MERVA (C) 33

## E

- encryption
  - of transferred information 49
  - user exit for 52
- ENM4ExitDecrypt 52
- ENM4ExitEncrypt 52
- ENM4ExitMacVerify (C) 53

- ENMClearSem 38
- ENMCloseSem 36
- ENMCreateSem 39
- ENMEndRAPI 33
- ENMGetReason 42
- ENMOpenSem 41
- ENMRestartRAPI 32
- ENMSetProfile 30
- ENMSetSecurity 33
- ENMSetSem 37
- ENMSetTestEnv 34
- ENMStartRAPI 31
- ENMWaitSemList 35
- error handling
  - getting the reason code 42

## G

- generating security user exits 57, 58, 60

## L

- language support 1
- log files
  - on the MERVA AIX side 62

## M

- MAC
  - user exit to generate 53
  - user exit to verify 53
- MERVA AIX
  - Display Diagnosis Log function 62
  - logging directory 62
- MERVA Connection/2
  - differences to MERVA AIX API 29
  - differences to MERVA OS/2 API 29
  - functions provided by 1
  - language support 1
  - objectives 1
- MERVA Connection/2 Client
  - Client requirements 5
  - customizing a Client application 8
  - customizing SNA services 6
  - customizing TCP/IP services 8
  - installing the Client 5
- MERVA OS/2
  - additional functions 30
  - diagnosis log 62
  - Display/Print Diagnosis Log (DPD) function 62
  - programmer's log 62
- message authentication code 53

## N

- Notices 105

## P

- profile
  - selecting 30

- programmer's log
  - Display/Print Diagnosis Log (DPD) function 62
  - on the MERVA AIX side 62
  - on the MERVA OS/2 side 62
  - on the remote application side 61

- programs, sample

- smplo1 80
  - smplo2 87
  - smplo2s 93
  - smplo3 95

## R

- RAPI Server AIX

- server requirements 15

- RAPI Server AIX system

- customizing SNA services 15
  - customizing TCP/IP services 18
  - installing the server 15

- RAPI Server OS/2

- server requirements 21

- RAPI Server OS/2 system

- customizing SNA services 22
  - installing the Server 21

- reason code, returning 42

- reconnecting remote program (ENMRestartRAPI) 32

- resynchronization 45

## S

- sample

- programs (C) 80
  - security exits 58
  - security user exits 73
  - SNA definitions 63

- sample programs

- smplo1 80
  - smplo2 87
  - smplo2s 93
  - smplo3 95

- security considerations

- overview 49
  - replacing user exits 57

- security information 33

- security user exits

- activating on remote application side 57
  - activating on the MERVA AIX side 60
  - activating on the MERVA OS/2 side 58
  - generating on remote application side 57
  - generating on the MERVA AIX side 60
  - generating on the MERVA OS/2 side 58
  - sample 58, 73

- semaphore

- clearing 38
  - closing 36
  - creating 39
  - opening 41
  - setting 37

- semaphores

- waiting for a list of 35

- setting conversation security information 33

- setting semaphores 37

- setting test environment 34

- smplo1 80

- smplo2 87

- smplo2s 93

- smplo3 95

## T

- test environment 34

## U

- user exit

- replacing security 57

- user exit points 51

- user exits

- ENM4ExitDecrypt (C) 52

- ENM4ExitEncrypt (C) 52

- ENM4ExitMacGen 53

- ENM4ExitMacVerify 53

- for MAC generation 53

- for MAC verification 53

- generating authentication key with 49

- introduction to interfaces 49

- sample security exit 73

- using to encrypt data 49

---

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

MERVA Family  
MERVA Connection/2

Publication No. SH12-6293-00

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

	Very Satisfied	Satisfied	Neutral	Dissatisfied	Very Dissatisfied
Overall satisfaction	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>

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

	Very Satisfied	Satisfied	Neutral	Dissatisfied	Very Dissatisfied
Accurate	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
Complete	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
Easy to find	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
Easy to understand	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
Well organized	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
Applicable to your tasks	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>

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**  
SH12-6293-00



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

SH12-6293-00

Cut or Fold  
Along Line





Program Number: 5622-122 OS/2 LAN  
5622-127 OS/2 Standalone  
5765-449 AIX

Printed in Denmark by IBM Danmark A/S

SH12-6293-00

