MERVA ESA Components

MERVA Connection/NT

Version 4 Release 1

SH12-6339-02

MERVA ESA Components

MERVA Connection/NT

Version 4 Release 1

SH12-6339-02

Note!
Before using this information and the product it supports, be sure to read the general information under "Appendix C. Notices" on page 53.

Third Edition, May, 2001

This edition applies to

Version 4 Release 1 of IBM MERVA ESA Components (5648-B30)

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

Changes to this edition are marked with a vertical bar.

Contents

About This Book

Read this book to find out how to work with MERVA Connection/NT. You learn how to install and customize MERVA Connection/NT, and how to write programs with the Remote MERVA Application Program Interface.

Who Should Read This Book

This book is intended for application programmers and system administrators who want to access Message Entry and Routing with Interfaces to Various Applications USE & Branch for Windows NT (MERVA USE & Branch for Windows NT) from an application program that runs under Windows NT.

This book also helps you install and customize MERVA Connection/NT.

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

- Windows NT server or workstation
- Systems Network Architecture (SNA)
- Application Programming Interface (API) of MERVA
- Transaction Control Protocol/Internet Protocol (TCP/IP)

How This Book is Organized

The first chapter of this book provides general information about MERVA Connection/NT by giving an overview of the product. Chapter 2 describes how to install and customize the Remote MERVA API Client. Chapter 3 tells you how to install and customize the Remote MERVA API Server. Chapter 4 tells you how to work with the Remote MERVA API and how to build API programs. It also helps you understand resynchronization. Chapter 5 covers aspects of security, such as encryption, authentication, and user exits. The appendixes contain diagnosis information and sample definitions.

Conventions and Terminology Used in This Book

In this book, the following naming conventions apply:

- MERVA is used when the description applies to MERVA USE & Branch for Windows NT.
- You can use Communications Server or Personal Communications to perform specific tasks described in this book. Both terms are used as synonyms.

Chapter 1. General Introduction to MERVA Connection/NT

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

Objectives

There is a wide range of banking applications available for the Windows NT platform. While many of these applications create and process S.W.I.F.T. messages, they do not provide a connection to public networks.

With SWIFT Link, MERVA provides connections to the S.W.I.F.T. network. With MERVA Link, MERVA provides connections to other MERVA systems.

To use Windows NT applications as banking applications, you must transfer messages that are created on your operating system to a MERVA system. Messages that you receive from one of these networks must be transferred from a MERVA system to your operating system.

You can achieve this by saving messages to files and transferring the files. At the same time, however, this solution requires operator intervention and can cause message integrity problems. To avoid these problems, you can implement a direct connection from the application on your operating system to the MERVA system. The MERVA system then works as if it were a component of the application.

MERVA Connection/NT is your direct connection. It is, however, not a ready-to-use S.W.I.F.T. interface on your operating system. It does not have a user interface.

MERVA Connection/NT offers you the Remote API for applications on Windows NT. With the Remote API, you can create an application on Windows NT to send messages to MERVA and receive messages from MERVA with a minimum of effort.

Functions

MERVA Connection/NT has the same functions as the MERVA API on your operating system. Additionally, it offers you the following functions:

- Calls that help you establish an intersystem connection.
- Calls that let you use MERVA alarms.
- A real-time interface that allows you to connect to MERVA.
- A flexible user exit interface with which you can handle security aspects.
- A resynchronization mechanism ensures that the Remote MERVA API program provides the same level of message integrity as a local API program.

Components

The following figure shows you the components and programming concepts of MERVA Connection/NT:

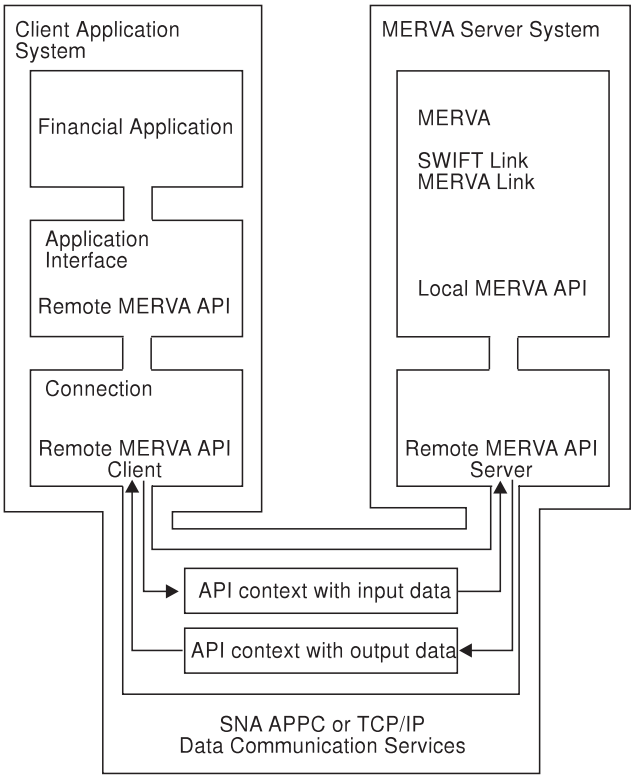

Figure 1. Concept of MERVA Connection/NT

MERVA Connection/NT has the following components:

- The Remote MERVA API Client is installed and runs in the Client Application System. The Client Application System is your operating system Windows NT. MERVA is not installed in the Client Application System.
- The Remote MERVA API Server is installed and runs in the MERVA Server System. The Remote MERVA API Server is part of the MERVA system that is installed in the MERVA Server System.

The Remote MERVA API Client has an interface with which you can call a financial application that uses MERVA services. It sends 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 runs locally.

Chapter 2. Installing and Customizing the Remote MERVA API Client

This chapter describes how to install and customize the Remote MERVA API Client in your operating system.

Installing the Remote MERVA API Client

The following sections describe how to install the Remote MERVA API Client of MERVA Connection/NT.

Machine Requirements

The following requirements are necessary to install the Remote MERVA API Client:

- You can install the Remote MERVA API Client on any Windows NT system with one megabyte or more free space on its hard disk.
- You must connect the MERVA Connection/NT Client Application System with the MERVA Server System by a Data Communication Link. The Data Communication Service (SNA APPC or TCP/IP) defines the type of intersystem link that you can use, such as Token Ring, SDLC, or Twinax. You must also install the corresponding data link adapter in your operating system.

For more detailed information refer to the corresponding documentation about installation and customization listed in "Bibliography" on page 61.

Program Requirements

The following section shows the program requirements for MERVA Connection/NT:

- Windows NT Version 4.0 or a subsequent release
- IBM eNetwork Personal Communications for Windows NT Version 4.2 or IBM eNetwork Communications Server for Windows NT Version 6.0 or subsequent releases, or Microsoft TCP/IP
- The C Compiler VisualAge® for C++ Version 3.5.4

Personal Communications or Communications Server for Windows NT is only required if you use an SNA APPC connection for the communication between the Remote MERVA API Client and Server. If you use a TCP/IP connection, Personal Communications is not required. In this case, TCP/IP must be installed on the Windows NT system.

Installing MERVA Connection/NT

To install MERVA Connection/NT:

1. Log on with a user ID that has Windows NT administration authorization.
2. Insert the CD labeled **MERVA ESA Components** in the CD ROM drive.
3. Select your CD ROM drive.
4. Select the **MERVA_Features** folder.
5. Select the **ConnectionNT** folder. This folder contains all dynamic link libraries, include files, and a sample program for a first installation check.

6. Copy the libraries **enmnrapi.dll**, **enmnsxit.dll**, and **enmratp.dll** to an existing directory. The directory must be set in the **PATH** environment variable, for example, **C:\WINNT**.
7. Copy the include files **enmradt.h**, **enmtapi.h**, and **enmrapd.h** to an existing directory. The directory must be set in the **INCLUDE** environment variable.
8. Select the **Samples** folder. This folder contains source files, module definition files, and make files for sample API programs.
9. Copy all files to a samples directory of your own.
10. Close the **Samples** folder.
11. Select the **Userexit** folder. This folder contains source files, a module definition file, and a make file for the generation of your own security user exits.
12. Copy all files to a user-exit directory of your own.
13. Close the **Userexit** folder.

Customizing the Communications Server

MERVA Connection/NT can use SNA APPC or TCP/IP services for the communication between the Remote MERVA API Client and Server.

If you use SNA APPC services, you bind APPC sessions between the two partner systems. To do this, install and customize the Communications Server on the Windows NT system.

Basic SNA Customization

For a detailed description of the customization refer to the *Communications Server for Windows NT User's Guide*. For a sample of the Communications Server customization that is independent of MERVA Connection/NT refer to "Appendix B. Sample SNA Definitions for MERVA Connection/NT" on page 47.

SNA Customization for MERVA Connection/NT

You must add an LU 6.2 side information profile to the SNA customization. The Remote MERVA API Client can then access this profile. The side information profile defines a symbolic destination name for the Remote MERVA API Server in the partner system.

The following list shows the parameters and samples of a symbolic destination:

- Symbolic destination name (MERVA)
- Local LU name (LU1)
- Fully qualified partner LU name (APPN1.LUA)
- APPC session mode name (APPCLU62)
- Partner transaction program (TP) name (ENMRAS)

The LU names and the mode name are specified in the basic SNA customization. The partner TP name is specified by the Remote MERVA API Server. For MERVA Connection/NT, the sample Remote MERVA API Server TP name in the MERVA environment is **ENMRAS**.

If the symbolic destination with the correct parameters, except for the TP name, is already defined, you do not have to define a symbolic destination for the Remote MERVA API Server. You can use the existing symbolic destination to identify the partner systems and the APPC session characteristics. If the corresponding TP

name is specified in the application profile of MERVA Connection/NT, the TP name in the side information profile is ignored.

Customizing TCP/IP Services

For the communication between the Remote MERVA API Client and Server, MERVA Connection/NT can use TCP/IP services if the server supports the TCP/IP communication protocol. The TCP/IP support for a Remote MERVA API Server is not available by default for MERVA. You must check whether the applicable Remote MERVA API Server supports TCP/IP.

Basic TCP/IP Customization

You must customize your operating system as a host in an IP network that is a network of connected hosts. It uses TCP/IP communication protocols. Specific MERVA Connection/NT requirements for the basic TCP/IP customization are not necessary.

TCP/IP Customization for MERVA Connection/NT

TCP/IP customization for MERVA Connection/NT is not applicable. Data that is related to the TCP/IP connection to the Remote MERVA API Server is provided in the corresponding application profile of MERVA Connection/NT.

The TCP/IP service can obtain the partner host name if it is defined in the corresponding host file of your operating system.

The TCP/IP service can also understand the partner host name if it is known by a name server in the TCP/IP network.

Customizing MERVA Connection/NT

You must customize any financial application that uses the Remote MERVA API in the Client Application System. The most important customization information is the identification of the applicable Remote MERVA API Server.

To customize a MERVA Connection/NT application, you store the relevant data in a MERVA Connection/NT application profile. This is a flat ASCII file that you can create and update with any text editor.

You can use the following formats for the application profile:

- The fix format profile supports only an SNA connection between the Remote MERVA API Client and Server.
- The variable format supports additional functions such as TCP/IP interconnection, conversation security, and test environment.

Fix Format Application Profile

A fix format application profile contains six parameters. The parameters can be specified in one of the following ways:

- In one line in which the parameters are separated by at least one blank
- In six separate lines
- As parameter groups from two up to five lines

The sequence of parameters is set in the following way:

1. Log level (1 to 4)

2. Name of the programmer's log
3. Name of the diagnosis log
4. SNA symbolic destination name of the Remote MERVA API Server
5. Name of the message integrity control file
6. System type of the Client Application System

A parameter file that starts with the log level is always interpreted as a fix format application profile. If a parameter file does not start with the log level, it is interpreted as a variable format application profile.

The following example shows a fix format application profile for MERVA Connection/NT.

```
1
plog.log
dlog.log
MERVA
mip.ctl
PS/2
```

An application profile in fix format is only compatible with older versions of the Remote MERVA API feature. New functions, such as conversation security and TCP/IP support do not use an application profile in fix format. These functions use an application profile in variable format to access all functions of the Remote MERVA API feature.

Variable Format Application Profile

With the variable format application profile, you can specify environment parameters for a remote MERVA application. The following variable formats are valid:

Parameter Keywords
: An application parameter is specified in a separate line in the format **parameter_keyword = parameter_value**. Any number of blanks can precede the parameter keyword. Any number of blanks can precede and succeed the equal sign. The equal sign is mandatory.

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 is valid.

Comments
: Comments can be part of an application profile. Any line that does not start with a valid parameter keyword is a comment line. An empty line is also a comment line. The first line of a profile must, however, not start with a digit. According to other conventions in the configuration profiles, you should 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.

The following example shows a variable format application profile of the Remote MERVA API Client for an SNA connection. A comment line shows the application profile parameters that are not supported by a fix format application profile. To activate the parameter, remove the hash character at the beginning of a comment line.

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

 log_level            = 1          minimum logging level
#log_mode             = append     append new log entries
 system_type          = PS/2       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
```

Parameters of the Variable Format Application Profile

The following section shows the parameters that apply for the Remote MERVA API Client and the corresponding parameter keywords.

log_level

Specifies the log level parameter. The parameter value is 1, 2, 3, or 4. Log level 4 provides the most detailed information.

log_mode

Specifies the log file mode for the programmer's log and the diagnosis log. The following parameter values apply:

- **append**

  The programmer's and diagnosis log entries are appended to existing log files. If this parameter is not specified in the application profile, this is the default log file mode.

- **new**

  Existing log files are deleted and the programmer's and diagnosis log entries are written to an empty file.

You can also enter only the initial character **a** or **n**. A log file mode parameter that does not start with **a** or **n** is ignored.

system_type

Identifies the type of the Client Application System. The parameter value for MERVA Connection/NT is PS/2.

programmer_log

Specifies the name of the programmer's log. The parameter value for MERVA Connection/NT is a Windows NT file. The programmer's log is only created if this parameter is specified.

diagnosis_log

Specifies the name of the diagnosis log. The parameter value for MERVA Connection/NT is a Windows NT file. The diagnosis log is only created if this parameter is specified.

control_file

Specifies the name of the message integrity (MIP) control file. The

parameter value for MERVA Connection/NT is a Windows NT file. The MIP control file is mandatory. The Remote MERVA API Client can only be started if this parameter is specified.

symbolic_destination
Specifies the name of an SNA APPC side information profile. This profile contains SNA APPC related control information about the APPC partner process in the Remote MERVA API Server. The parameter can be up to 8 characters long.

partner_tp_name
Specifies the name for the SNA APPC partner TP. The parameter value is the TP name of the Remote MERVA API Server that is defined in the partner system. The TP name that is specified in this parameter takes precedence over the TP name that is specified in the side information profile. If this parameter is not specified, the TP name of the side information profile applies. The parameter can be up to 8 characters long.

partner_host_name or partner_host
Specifies the name or the IP address of the host to which the Remote MERVA API Server belongs. You must specify the IP address in dotted-decimal representation. The parameter value for MERVA Connection/NT can be up to 16 characters long.

rapi_port_number or port_number
Specifies the TCP/IP port number for the Internet subserver that represents the Remote MERVA API Server in the corresponding partner host system. The maximum value of a TCP/IP port number is 65.535.

tcp_nodelay
Specifies whether the **tcp_nodelay** parameter must be set. Valid parameter values are **off**, **on**, **0**, and **1**. If the **tcp_nodelay** parameter must be set, the parameter value is **1** or **on**. The default parameter value is **1**. This parameter can affect the performance of the data communication.

client_user_id
Specifies the client user ID for the conversation security. The parameter value is the conversation security user ID that is used 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 MERVA API transaction program. The maximum length of this parameter is 8 characters.

The client user ID that is specified in this parameter applies only if the user application program does not provide a user ID before the application profile is read. You can, however, delete the user ID of the application program with **client_user_id =** ″″.

client_user_password
Specifies the client user password for the conversation security. The parameter value is the conversation security user password that is used for the specified client user. If a user ID is not specified in the application profile or by the application program, the Remote MERVA API Client disregards the password. The maximum length of this parameter is 8 characters.

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

You can, however, delete the user ID of the application program with **client_password =** ″″.

test_environment

Specifies whether the test environment is activated when the client process starts. The parameter value is **on** or **1**. The test environment is not activated if this parameter is not specified or if any other parameter value is specified.

You can use the Remote MERVA API Client function **ENMSetTestEnv()** to set or reset the client process test environment in a Remote MERVA API Client user program for specific phases of the client process.

A Remote MERVA API Client process in a test environment writes a processing trace to the standard output device, usually to the user terminal. You can use this trace for error analysis. The programmer's log and the diagnosis log also contain information for error analysis.

Selecting the Communication Type

The Remote MERVA API Client can establish a conversation with a Remote MERVA API Server by using SNA APPC or TCP/IP services. The application profile must contain the corresponding address information of the partner system. The appropriate customization must apply to the applicable data communication services.

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

TCP/IP partner information establishes a TCP/IP connection to the Remote MERVA API Server if the application profile does not contain an SNA symbolic destination name.

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

Chapter 3. Customizing the Remote MERVA API Server

The Remote MERVA API Server is automatically installed when you install MERVA USE & Branch for Windows NT. You must, however, configure the Remote MERVA API Server program. To use the Remote MERVA API Server, the following requirements are necessary:

Machine Requirements

A Data Communication Link must connect the MERVA Connection/NT Client Application System and the MERVA Server System. As specified by the Data Communication Service that is used, such as SNA APPC or TCP/IP, you can use Token Ring, SDLC, Twinax, or other types of intersystem links. You must also install a corresponding data link adapter in the Windows NT system.

For more detailed information refer to the *MERVA USE & Branch for Windows NT Installation and Customization Guide*.

Program Requirements

The following program requirements are necessary:

- Microsoft Windows NT (refer to the *MERVA USE & Branch for Windows NT Installation and Customization Guide*)
- IBM eNetwork Personal Communications for Windows NT Version 4.2 or IBM eNetwork Communications Server for Windows NT Version 6.0 or subsequent releases, or Microsoft TCP/IP

Personal Communications or Communications Server is only required if you use an SNA APPC connection for the communication between the Remote MERVA API Client and Server. If you use a TCP/IP connection, SNA services are not required.

Customizing the Communications Server

MERVA Connection/NT can use SNA APPC or TCP/IP services for the communication between the Remote MERVA API Client and Server.

If you use TCP/IP services, you must install TCP/IP for Windows NT on your system.

If you use SNA APPC services, you must install and customize the SNA Server for Windows NT or the Communications Server for Windows NT in the MERVA USE & Branch for Windows NT system to bind APPC sessions between the two partner systems.

Basic SNA Customization

You can connect any MERVA Connection/NT system to MERVA USE & Branch for Windows NT.

For a description of the respective customization refer to the corresponding documentation about SNA Server for Windows NT or Communications Server for Windows NT listed in “Bibliography” on page 61.

For a sample of the SNA customization that is independent of MERVA Connection/NT refer to "Appendix B. Sample SNA Definitions for MERVA Connection/NT" on page 47.

SNA Customization for the Remote MERVA API Server

You must add an LU 6.2 TP name profile to the SNA customization. This profile defines the parameters of an inbound APPC transaction program. The parameters are:

- TP name (**ENMRAS**)
- Full path name of the executable (**enmcrtpi.exe**)
- Command line parameters (**tp_name instance_name [TS=trace_level TP=trace_path]**)
- TP access security

If you use **Conversation security**, you must add an appropriate entry for the user ID and the password.

Customizing the Trace File for SNA

You must set the trace switch (**TS**) and the path of the trace file (**TP**) with the command line parameters provided by the transaction program, for example:

```
TS=3 TP=C:\MERVA\TRACE
```

If the trace switch is set to 0, a trace file is not created. The path name shown is the path of the directory to which all trace files are written. Replace **C:\MERVA\TRACE** with an appropriate path name. The name of the trace file is **TPI<timestamp>.LOG**, for example, **TPI104530.LOG**.

Customizing TCP/IP Services

Before MERVA Connection/NT can use TCP/IP services for the communication between the Remote MERVA API Client and Server, you must install TCP/IP for Windows NT on the Remote MERVA API Server and all Remote MERVA API Clients. On the Remote MERVA API Server, the MERVA Inetd service must be installed and configured.

The Remote MERVA API Server must be defined as a client network service (internet service) on the remote host. This definition maps a TCP/IP port number to a designated service program.

The Remote MERVA API Server works as a subservice of the MERVA Inetd service. It must be defined as a MERVA Inetd subservice on the remote host. This definition connects the internet service name and the service program path name for the Remote MERVA API Server.

For more information about the MERVA Inetd service, refer to the *MERVA USE & Branch for Windows NT Installation and Customization Guide*.

Customizing Client Network Services

The file **%SystemRoot%\system32\drivers\etc\services** contains all TCP/IP services that are available on the Remote MERVA API Server. This file maps a service to a specific port and a transport protocol. To customize the Client Network Services, do the following:

1. Change to the directory **%SystemRoot%\system32\drivers\etc**.

2. Edit the file **services** and add a text line such as:

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

Where:

enmras
Is a symbolic subservice name for the Remote MERVA API connection.

7118/tcp
Defines the IP port number 7118 for the connection and specifies that the TCP protocol is required.

Note that the last line in the **services** file must be empty.

Customizing the MERVA Inetd Service

The file **%SystemRoot%\system32\drivers\etc\enminetd.cfg** contains all services that are started with the MERVA Inetd service. To customize the MERVA Inetd service, do the following:

1. Change to the directory **%SystemRoot%\system32\drivers\etc**.
2. Edit the configuration file **enminetd.cfg** and add a text line such as:

```
enmras stream tcp nowait root c:\merva\bin\enmcrtci.exe merva1 [trace-options]
```

Where:

enmras
Is a symbolic subservice name for the Remote MERVA API connection. This name must be identical to the name defined in the **%SystemRoot%\system32\drivers\etc\services** file.

c:\merva\bin\enmcrtci.exe
Is the Remote MERVA API Server program.

merva1
Is the MERVA instance name.

trace-options
Can be set as described in "Customizing the Trace File for TCP/IP".

Customizing the Trace File for TCP/IP

You must set the trace switch (**TS**) and the path of the trace file (**TP**) with the command line parameters provided by the program **enmcrtci.exe** defined in the configuration file **enminetd.cfg**. For example:

```
TS=3 TP=C:\MERVA\TRACE
```

If the trace switch is set to **0**, a trace file is not created. The path name shown is the path of the directory to which all trace files are written. Replace **C:\MERVA\TRACE** with an appropriate path name. The name of the trace file is **TCI<timestamp>.LOG**.

Verifying the Installation

To verify that the installation and customization of MERVA Connection/NT is correct, run the corresponding sample program. Before you can run the sample program, the user ID **SAMPLE** with the password **SAMPLE1** has to be defined in your MERVA system. The user ID has to be approved for application programs. The program checks that the queues **API_IN** and **API_OUT** are customized.

To verify the installation, run the sample program **SMPLN4.EXE**.

This program is a compiled version of the sample source **SMPLN4.C**. It is included in the directory **\SAMPLES** on the MERVA Connection/NT disk. **SMPLN4.EXE** needs the profile **SAMPLE.PRF** in the current directory. For a description of how to get this profile, refer to "Installing MERVA Connection/NT" on page 3.

Chapter 4. The Remote MERVA Application Program Interface

The description of the Remote MERVA Application Program Interface in this chapter is based on the description in the corresponding *MERVA Application Programming Guide*.

The Structure of the Remote MERVA API on the Client Side

The Remote MERVA API program on your operating system calls functions that connect and disconnect to and from the MERVA system. The following figure shows you the structure of the Remote MERVA API:

```
1 ENMSetProfile(profile name)
2 ENMStartRAPI(application name)
    ---
     |
     | API program logic with MERVA API calls
     |
    ---
3 ENMEndRAPI()
```

Figure 2. Remote MERVA API Program Structure

1 Before you can call the API functions, you must initialize the Remote MERVA API Client on your operating system by calling the function **ENMSetProfile**. This function tells the Remote MERVA API Client the name of the profile. For details refer to "Customizing MERVA Connection/NT" on page 5.

2 After you set the profile name, you can connect to the Remote MERVA API Server on the MERVA side. To do this, call the function **ENMStartRAPI**. After 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 API functions can be called as if the program runs locally on a MERVA machine.

3 Before you stop the program, you must release the connection to the Remote MERVA API Server by calling the function **ENMEndRAPI**. You must call this function even if an error occurs in the API program. Otherwise, the Remote MERVA API Server does not know that you exit the program and is not ready to receive the next connection request when the API program is restarted.

C Language Data Types

When you compile a Remote MERVA API program locally on a MERVA machine, the file **enmoapi.h** is automatically included.

When you compile a Remote MERVA API program on Windows NT, you must include the file **enmrapi.h** instead of **enmoapi.h**. The file **enmrapi.h** contains the data types and prototypes of MERVA API functions.

For the description of the API calls in this book, the following data types defined in the included file **enmrapi.h** are used:

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

API Calls of MERVA Connection/NT

MERVA Connection/NT offers you more API calls than the MERVA API. The calls are divided into the following categories:

- Calls to start and end the conversation
- Calls to enable the API program to be triggered by MERVA alarms
- Calls to handle errors

The following sections describe these calls in detail.

Starting and Ending the Conversation

To start and end the conversation between the Remote MERVA API Client and the Remote MERVA API Server with the API program, use the following calls:

ENMSetProfile To select a profile.

ENMStartRAPI To establish a connection to MERVA.

ENMRestartRAPI To reconnect the Remote MERVA API program to MERVA.

ENMEndRAPI To disconnect from MERVA.

ENMSetSecurity To set conversation security information.

ENMSetTestEnv To set the test environment.

Each function is described in detail in the following sections.

Note: The return code that you get depends on your MERVA Connection/NT system.

ENMSetProfile - Select a Profile

C Definition:

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

Parameter Description: The following parameter is required:

- **pucProfileName**(PUCHAR)

  Specifies the pointer to a null-terminated string of up to 80 characters. This is the full path name of the profile.

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

Remarks: Specifies the name of the profile that you want to use. For a description of the format and contents of the profile, refer to "Customizing MERVA Connection/NT" on page 5.

C Language Example:

```
#include "enmrapi.h"

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

ENMStartRAPI - Establish Connection to MERVA

C Definition:

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

Parameter Description: The following parameters are required:

- **retCode**(USHORT) - output

Code	Meaning
0	The function completed correctly.
2	An internal error has occurred. The API program receives further information by calling the function **ENMGetReason**. For details refer to "Handling Errors" on page 29. The reason code is also written to the diagnosis log of your operating system. For details refer to "Appendix A. Diagnosis Information" on page 45. If it is an internal error of the MERVA API, the reason code is **0**.

- **pucApplicationName**(PUCHAR) - input

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

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

Remarks: This function establishes the conversation with MERVA (Remote MERVA API Server) and initializes internal buffers and variables. After this function is called, the program must call **ENMEndRAPI** before it ends.

C Language Example:

```
#include "enmrapi.h"

USHORT rc = 0;

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

ENMRestartRAPI - Reconnect to MERVA

C Definition:

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

Parameter Description: The following parameters are required:

- **retCode**(USHORT) - output

Code	Meaning
0	The function completed correctly.
2	An internal error has occurred. The API program receives further information by calling the function **ENMGetReason**. For details refer to "Handling Errors" on page 29. The reason code is also written to the diagnosis log of your operating system. For details refer to "Appendix A. Diagnosis Information" on page 45. If it is an internal error of the MERVA API, the reason code is **0**.

- **pucApplicationName**(PUCHAR) - input

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

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

Remarks: If the connection is established with this function instead of **ENMStartRAPI**, the resynchronization is provided for the following API calls:

- **ENMAdd**
- **ENMDelete**
- **ENMPut**
- **ENMRouteAdd**
- **ENMRoutePut**

For details refer to "Resynchronization" on page 33.

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

C Language Example:

```
#include "enmrapi.h"

USHORT rc = 0;

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

ENMEndRAPI - Disconnect from MERVA

C Definition:

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

Parameter Description: The following parameter is required:

- **retCode**(USHORT) - output

Code	Meaning
0	The function completed correctly.
2	An internal error has occurred. The API program receives further information by calling the function **ENMGetReason**. For details refer to "Handling Errors" on page 29. The reason code is also written to the diagnosis log of your operating system. For details refer to "Appendix A. Diagnosis Information" on page 45. If it is an internal error of the MERVA API, the reason code is **0**.

Remarks: The Remote MERVA API conversation to MERVA is stopped.

C Language Example:

```
#include "enmrapi.h"

USHORT rc = 0;

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

ENMSetSecurity - Set Conversation Security Information

C Definition:

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

Parameter Description: The following parameters are required:

- **pucUserID**(PUCHAR) - input

  A pointer to a null-terminated string of up to 8 characters that contains the client user ID.
- **pucPassword**(PUCHAR) - input

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

Remarks: A MERVA application program can provide conversation security information that is used for client authorization in the Remote MERVA API Server system. To provide this information, use the function **ENMSetSecurity()**. The parameters of this function are a client user ID and a password. Both parameters can be empty.

Before you start **ENMStartRAPI()** or **ENMRestartRAPI()**, you must call **ENMSetSecurity()**.

You can also provide conversation security information with application profile parameters. Usually, the information provided by **ENMSetSecurity()** takes precedence over profile parameters. You can, however, overwrite the security information set with **ENMSetSecurity()** by using application profile parameters.

C Language Example:

```
#include "enmrapi.h"

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

ENMSetTestEnv - Set Test Environment

C Definition:

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

Parameter Description: The following parameter is required:

- **ucTestEnvIndicator**(UCHAR) - input

  Function parameter **1** activates the test environment. Function parameter **0** deactivates the test environment.

Remarks: A MERVA application program can activate or deactivate the Remote MERVA API Client test environment for specific sections of the application program. To do this, call the function **ENMSetTestEnv()**. You can call this function as often as you want.

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

C Language Example:

```
#include "enmrapi.h"

#define TESTENV_ON  '1'

ENMSetTestEnv (TESTENV_ON);
```

Calls to Trigger the API Program

If you want the API program to be triggered by MERVA alarms the semaphores of which are located on the MERVA system, use the following calls:

ENMWaitSemList	To wait for a list of semaphores.
ENMCloseSem	To close a semaphore.
ENMSetSem	To set a semaphore.
ENMClearSem	To clear a semaphore.
ENMCreateSem	To create a semaphore.
ENMOpen	To open a semaphore.

Each function is described in detail in the following sections.

ENMWaitSemList - Wait for a List of Semaphores

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

C Definition:

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

Parameter Description: The following parameters are required:

- **retCode**(USHORT) - output

Code	Meaning
0	The function completed correctly.
2	An internal error has occurred. The API program receives further information by calling the function **ENMGetReason**. For details refer to "Handling Errors" on page 29. The reason code is also written to the diagnosis log of your operating system. For details refer to "Appendix A. Diagnosis Information" on page 45. If it is an internal error of the MERVA API, the reason code is **0**.
6	The system does not have enough memory to complete the function.
121	No semaphore is cleared. The timeout is reached.
255	An internal semaphore error occurred. For details refer to "Handling Errors" on page 29. The reason code is also written to the diagnosis log of your operating system. For details refer to "Appendix A. Diagnosis Information" on page 45. The reason code is in the form of **3xxx** where **xxx** denotes the error number of the server system.

- **Index**(PUSHORT) - output

  ENMWaitSemList returns an index from 0 to 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

  Handles up to 16 semaphores that are created by the call **ENMCreateSem** or **ENMOpenSem**.

- **(ULONG)0** - input

  Stops the list of semaphores. The parameter value must be 0 and a 4-byte value. If the parameter is missing, **ENMWaitSemList** cannot recognize the end of the semaphore list.

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

C Language Example:

```
#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 that is obtained with an **ENMCreateSem** or **ENMOpenSem** call.

C Definition:

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

Parameter Description: The following parameters are required:

- **retCode**(USHORT) - output

Code	Meaning
0	The function completed correctly.
2	An internal error has occurred. The API program receives further information by calling the function **ENMGetReason**. For details refer to "Handling Errors" on page 29. The reason code is also written to the diagnosis log of your operating system. For details refer to "Appendix A. Diagnosis Information" on page 45. If it is an internal error of the MERVA API, the reason code is **0**.
6	The system does not have enough memory to complete the function.
102	A semaphore is set. Therefore, it cannot be closed.
255	An internal semaphore error occurred. For details refer to "Handling Errors" on page 29. The reason code is also written to the diagnosis log of your operating system. For details refer to "Appendix A. Diagnosis Information" on page 45. The reason code is in the form of **3xxx** where **xxx** denotes the error number of the server system.

- **SemHandle**(ULONG) - input

  Generated by ENMCreateSem or ENMOpenSem.

C Language Example:

```
#define TRIGGER "SAMPLE2"

#include "enmrapi.h"

   USHORT      rc = 0;
   ULONG       SemTrigger;

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

ENMSetSem - Set a Semaphore

ENMSetSem sets a semaphore unconditionally. For MERVA, you can clear the semaphore by raising an alarm.

C Definition:

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

Parameter Description: The following parameters are required:

- **retCode**(USHORT) - output

Code	Meaning
0	The function completed correctly.
2	An internal error has occurred. The API program receives further information by calling the function **ENMGetReason**. For details refer to "Handling Errors" on page 29. The reason code is also written to the diagnosis log of your operating system. For details refer to "Appendix A. Diagnosis Information" on page 45. If it is an internal error of the MERVA API, the reason code is **0**.
6	The system does not have enough memory to complete the function.
100	The limit of open semaphores in the system is exceeded.
103	There are too many semaphore requests on the system.
255	An internal semaphore error occurred. For details refer to "Handling Errors" on page 29. The reason code is also written to the diagnosis log of your operating system. For details refer to "Appendix A. Diagnosis Information" on page 45. The reason code is in the form of **3xxx** where **xxx** denotes the error number of the server system.

- **SemHandle**(ULONG) - input

  Generated by ENMCreateSem or ENMOpenSem.

C Language Example:

```
#define TRIGGER "SAMPLE2"

#include "enmrapi.h"

   USHORT      rc = 0;
   ULONG       SemTrigger;

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

ENMClearSem - Clear a Semaphore

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

C Definition:

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

Parameter Description: The following parameters are required:

- **retCode**(USHORT) - output

Code	Meaning
0	The function completed correctly.
2	An internal error has occurred. The API program receives further information by calling the function **ENMGetReason**. For details refer to "Handling Errors" on page 29. The reason code is also written to the diagnosis log of your operating system. For details refer to "Appendix A. Diagnosis Information" on page 45. If it is an internal error of the MERVA API, the reason code is **0**.
6	The system does not have enough memory to complete the function.
101	A semaphore cannot be cleared because another process owns it.
255	An internal semaphore error occurred. For details refer to "Handling Errors" on page 29. The reason code is also written to the diagnosis log of your operating system. For details refer to "Appendix A. Diagnosis Information" on page 45. The reason code is in the form of **3xxx** where **xxx** denotes the error number of the server system.

- **SemHandle**(ULONG) - input

  Generated by ENMCreateSem or ENMOpenSem.

C Language Example:

```
#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 a semaphore on the Remote MERVA API Server.

C Definition:

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

Parameter Description: The following parameters are required:

- **retCode**(USHORT) - output

Code	Meaning
0	The function completed correctly.
2	An internal error has occurred. The API program receives further information by calling the function **ENMGetReason**. For details refer to "Handling Errors" on page 29. The reason code is also written to the diagnosis log of your operating system. For details refer to "Appendix A. Diagnosis Information" on page 45. If it is an internal error of the MERVA API, the reason code is **0**.
6	The system does not have enough memory to complete the function.
87	One of the parameters is not valid.
100	The limit of open semaphores in the system is exceeded.
123	The name of the semaphore is not valid.
183	The semaphore already exists.
255	An internal semaphore error occurred. For details refer to "Handling Errors" on page 29. The reason code is also written to the diagnosis log of your operating system. For details refer to "Appendix A. Diagnosis Information" on page 45. The reason code is in the form of **3xxx** where **xxx** denotes the error number of the server system.

- **SemHandle**(PULONG) - output

  Address of the semaphore handle.

- **SemName**(PUCHAR) - input

  Pointer to a null-terminated string that contains the name of the semaphore to be created. The semaphore name is a logical name without path details.

C Language Example:

```
#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**.

C Definition:

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

Parameter Description: The following parameters are required:

- **retCode**(USHORT) - output

Code	Meaning
0	The function completed correctly.
2	An internal error has occurred. The API program receives further information by calling the function **ENMGetReason**. For details refer to "Handling Errors". The reason code is also written to the diagnosis log of your operating system. For details refer to "Appendix A. Diagnosis Information" on page 45. If it is an internal error of the MERVA API, the reason code is **0**.
100	The limit of open semaphores in the system is exceeded.
123	The name for the semaphore is not valid.
187	The semaphore does not exist.
255	An internal semaphore error occurred. For details refer to "Handling Errors". The reason code is also written to the diagnosis log of your operating system. For details refer to "Appendix A. Diagnosis Information" on page 45. The reason code is in the form of **3xxx** where **xxx** denotes the error number of the server system.

- **SemHandle**(PULONG) - output

  Address of the handle of the opened semaphore.

- **SemName**(PUCHAR) - input

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

C Language Example:

```
#define TRIGGER "SAMPLE2"

#include "enmrapi.h"

   USHORT      rc = 0;
   ULONG       SemTrigger;

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

Handling Errors

If you want the API call to return reason codes, use the function **ENMGetReason**. This function gets the reason for the internal error. It is described in the following section.

ENMGetReason - Get Reason Code for Internal Error

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

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

C Definition:

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

Parameter Description: The following parameters are required:

- **retCode**(USHORT) - output

Code	Meaning
2xxx	Reason codes from **2000** to **2999** indicate communication problems.
2110	The APPC conversation cannot be established or is cancelled.
2120	The Communications Side Information object is not found.
2130	The connection to the Remote MERVA API Server program failed.
2140	Deallocation failed because the conversation has already been stopped.
2150	The conversation was interrupted while the program tried to receive data.
2200	An empty data buffer was received.
28xx	xx is a return code of the TCP/IP service programs.
29xx	xx is a return code of the CPI-C call.
2999	A general communication problem occurred. For details refer to the diagnosis log.
3xxx	An internal semaphore error occurred. **xxx** is the error number provided by Windows NT.
7006	The Remote MERVA API Server failed while the program tried to allocate memory.
7012	The Remote MERVA API Server does not accept further API calls due to a previous error.
7013	The Remote MERVA API Server received a negative return code from user exit **ENM4ExitDecrypt**.
7014	The Remote MERVA API Server received a negative return code from user exit **ENM4ExitEncrypt**.
7015	The Remote MERVA API Server received a negative return code from user exit **ENM4ExitMacVerify** or **ENM4ExitMacGen**.
7016	The Remote MERVA API Server received an incorrect API request.
7018	The Remote MERVA API Server received an error while the program converted ASCII to EBCDIC. For details refer to the diagnosis log of MERVA.
7019	The Remote MERVA API Server received an error while the program accessed the message integrity control file.
7030	Internal message space was not created.

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

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

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

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

8006 The Remote MERVA API Client could not allocate memory.

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

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

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

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

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

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

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

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

8017 The conversation has not been started with **ENMStartRAPI** or with **ENMStartAPPC**.

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

8020 The Remote MERVA API Client could not load the file **ENMRATP.DLL**.

8021 The profile does not contain information about the partner system.

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/NT occurred, reason code %d",
            reason);
      }
   }
```

Building API Programs

This section describes how to compile MERVA Connection/NT programs in the C programming language.

Compiling Your Own API Program

To compile your API program on the Windows NT system, enter the following commands:

1. **icc /C /DWIN32 /Gd+ /Gm+ /Gs- /Gt+ <name>.c**
2. **ilink /NOE <name>.obj ENMNRAPI.LIB <name>.def**

   Note that you might have to link additional libraries to your program.

Compiling the Sample Programs

To generate the executable files for the delivered sample programs, copy the files of the directory **\SAMPLES** to a directory of your choice. Note that the sample programs use the profile **SAMPLE.PRF** that must be located in the same path as the sample program. The following list shows you the contents of the **samples** directory:

SMPLN1.MAK — Make file to generate the sample API program **SMPLN1**. To generate this program, enter:

nmake /f smpln1.mak

SMPLN1.C — Sample program that is attached to MERVA, for example, to query queue information, create messages, or send messages.

SMPLN2.MAK — Make file for **SMPLN2**. To generate the sample API programs **SMPLN2** and **SMPLN2S**, enter:

nmake /f smpln2.mak

SMPLN2.C — Sample program to trigger MERVA.

SMPLN2S.C — Sample program to stop **SMPLN2**.

SMPLN3.MAK — Make file for **SMPLN3**. To generate the sample API program **SMPLN3**, enter:

nmake /f smpln3.mak

SMPLN3.C — Sample program to load telex messages through API queues. To run the sample program **SMPLN3**, you need the data file **SAMPLE.DAT**.

SMPLN4.MAK — Make file for **SMPLN4**. To generate the sample API program **SMPLN4**, enter:

nmake /f smpln4.mak

SMPLN4.C — Sample program to verify the MERVA Connection/NT installation.

SAMPLE.PRF — File that contains a sample profile.

SMPLN4.EXE — Compiled version of **SMPLN4.C** for immediate use.

SMPLN4.DAT — Date file that contains three telex messages for **SMPLN3**.

Resynchronization

If a network connection is interrupted, the recovery procedure ensures that the status of a message in MERVA, such as **Add**, **Route**, or **Delete** is changed only once. This affects the programs that use the Remote MERVA API and programs that call the local MERVA API.

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

The following figure shows you an example of the processing steps:

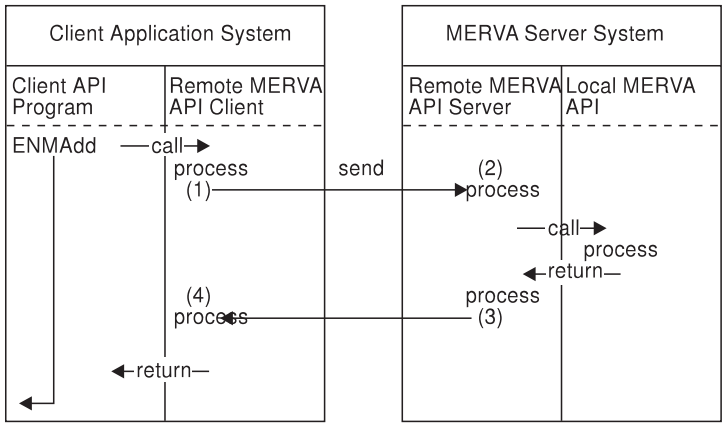

Figure 3. Resynchronization Support

The return code **ERR_SYSTEM** of the API call and the corresponding reason code (**2000** to **2999**) of an additional **ENMGetReason** call indicates whether the network connection is interrupted. MERVA Connection/NT does not know whether the call completed successfully, or whether it is not executed in the MERVA system. In the example of Figure 3, the API program does not know whether the message is added to the MERVA queue.

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

- **ENMAdd**
- **ENMDelete**
- **ENMPut**
- **ENMRouteAdd**
- **ENMRoutePut**

How to Implement Resynchronization

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

If the network connection stops before the return data is sent 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 then sent back as if the call was not interrupted in the previous run.

The necessary control data is saved in files. On the Remote MERVA API Client, you can specify the file name in the MERVA Connection/NT profile as described in "Customizing MERVA Connection/NT" on page 5. 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:

- 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 example shows you the 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 stops. When you restart the API program, the loop is entered as if there was no interruption in the previous run.

To ensure that resynchronization works correctly:

- Use the same profile as in the previous run.
- Call **ENMRestartRAPI** by using the same application name.
- Call **ENMCreate** and **ENMWriteField** by using the same data, such as the message field contents, as in the previous run.
- Call **ENMRouteAdd** by using the same queue name.
- After resynchronization you can continue with the loop as you do in normal processing.

If the program runs in this way, it does not have to check the status of processing at the time when the **ENMRouteAdd** call was interrupted.

Recovering after a Failed Call

If the call of **ENMAdd** or **ENMRouteAdd** fails, you usually call **ENMClear** to clear the message space. For details refer to the corresponding MERVA *Programming Guide*.

If these calls fail after you reestablish the connection because of other reasons than network problems, you might get the following return code (**202**) when you call **ENMClear**:

```
ERR_NO_MSG_CREATED
```

You can ignore this error message because the API call was executed in the first run.

The same applies to an **ENMFree** call that returns the following message (**201**) after the call of **ENMDelete**, **ENMPut**, or **ENMRoutePut** fails:

```
ERR_NO_MSG_LOCKED
```

Using Other Methods

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

MERVA Connection/NT does not save the type or input data of the API call that fails due to the network failure. If one of the above mentioned calls fails, ensure that the same call is repeated after reconnection to the MERVA system. If you do not ensure this, an API call with new data could be wrongly considered as a repeated call from a previous run. This applies only if you use **ENMRestartRAPI**.

MERVA Connection/NT does not recognize an inappropriate API call. If the internal state indicates that the last API call from the previous run is executed, the call is ignored.

Chapter 5. Security

Security is an important requirement of all financial institutions. The security of message transfers is determined by:

- Encryption of transferred data
- Authentication of transferred data

MERVA Connection/NT supports encryption and authentication.

Encryption of Transferred Data

To encrypt data, you activate user exits. User exits allow you to include your own algorithm or products that support encryption and decryption routines.

The following user exits are valid:

- ENM4ExitEncrypt for encryption
- ENM4ExitDecrypt for decryption

For detailed information on how to implement these routines, refer to "User Exit Interfaces".

Authentication of Transferred Data

To generate an authentication key that covers all exchanged data, you activate user exits. User exits allow you to include your own algorithm or products that support authentication routines.

The following user exits are valid:

- ENM4ExitMacGen for MAC generation
- ENM4ExitMacVerify for MAC verification

For detailed information on how to implement these routines, refer to "User Exit Interfaces".

User Exit Interfaces

API calls and user exits are different:

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

User Exit Points

The following figure shows an example of an API function that is called by an API program on your operating system. You can see who calls a user exit at which processing step. In the figure, the following abbreviations are used for the user exits:

ENCRYP ENM4ExitEncrypt

DECRYP ENM4ExitDecrypt

MACGEN ENM4ExitMacGen

MACVFY ENM4ExitMacVerify

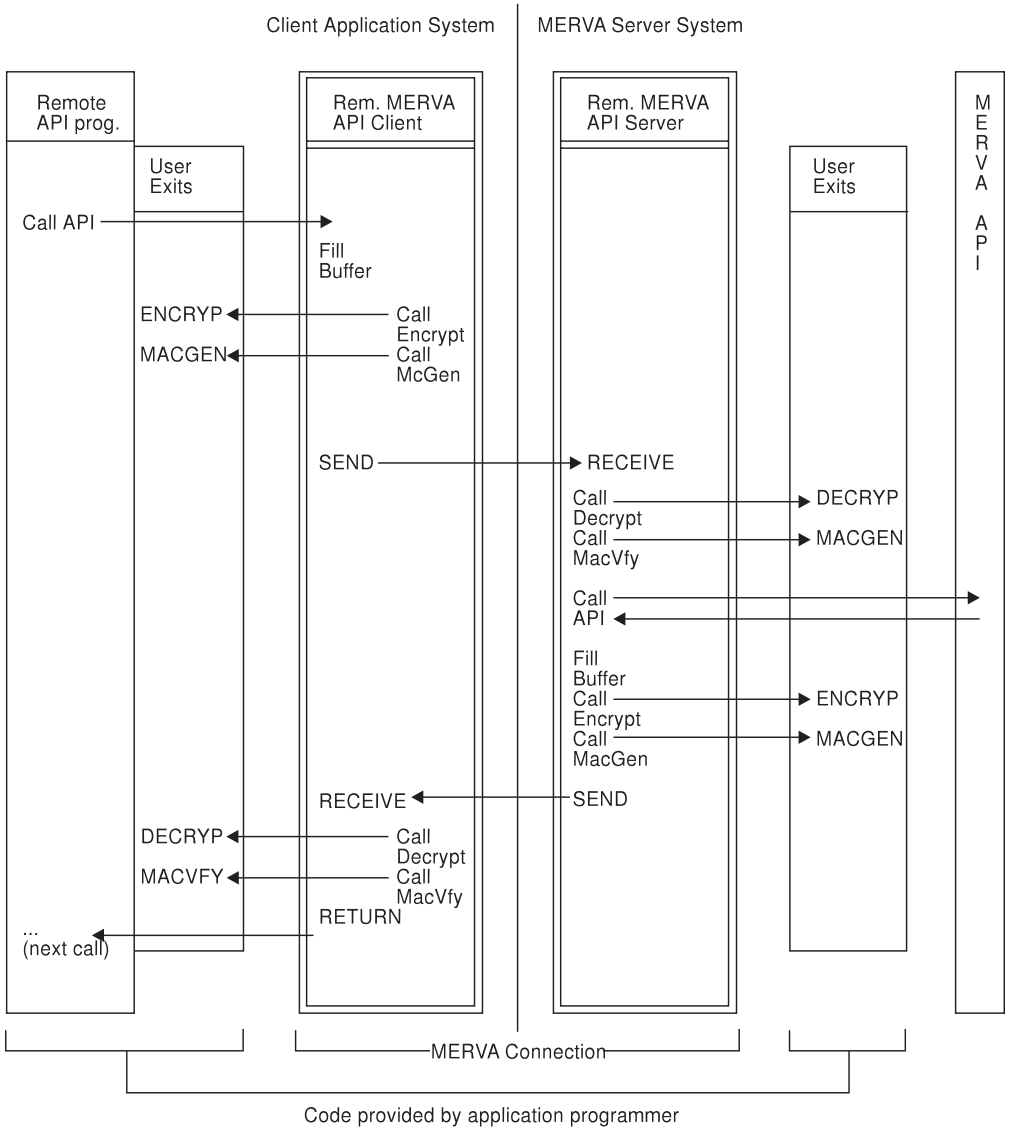

Figure 4. Example of an API Program That Calls an API Function

User Exit Interfaces in C Language

The data types that are used in these routines can be different, depending on the operating system on which they are implemented.

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 up to 8 characters long. The string contains the application identifier that is passed as a parameter of the API call **ENMStartRAPI**. You can use this string to provide different encryption keys for different partner connections. You can also use this string to specify the connections or API programs for which 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 up to 8 characters long. The string contains the application identifier that is passed as a parameter of the API call **ENMStartRAPI**. You can use this string to provide different encryption keys for different partner connections. You can also use this string to specify the connections or API programs for which the information is to be encrypted.
- **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 Message Authentication Code (MAC) Generation

C Definition:

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

Purpose of the User Exit Routine: Generates a MAC for the passed data buffer.

Parameter Description: The following parameters are required:

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

  Address of a null-terminated string up to 8 characters long. The string contains the application identifier that is passed as a parameter of the API call **ENMStartRAPI**. You can use this string to provide different MAC generation algorithms for different partner connections. You can also use this string to specify the connections or API programs for which a MAC is to be generated.
- **pucBuffer**(unsigned char*) - input

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

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

  Address of the area to which the generated MAC is to be copied. The address can be up to 32 bytes long.

User Exit for MAC Verification

C Definition:

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

Purpose of the User Exit Routine: Generates a MAC for the passed data buffer and compares it with the passed MAC. If both MACs match, set the return code to **0**. If the MACs do not match, set the return code to **1**.

Parameter Description: The following parameters are required:

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

  Address of a null-terminated string up to 8 characters long. The string contains the application identifier that is passed as a parameter of the API call **ENMStartRAPI**. You can use this string to provide different MAC verification algorithms for different partner connections. You can also use this string to specify the connections or API programs for which a MAC is to be verified.
- **pucBuffer**(unsigned char*) - input

  Address of the data buffer for which a MAC is to be generated and for which the passed MAC is generated on the partner side.
- **usBufferLen**(unsigned short) - input

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

  Address of the area that holds the MAC key that is received from the partner side. The address can be up to 32 bytes long.

Replacing Security User Exits

This section describes the provided sample security user exits. It also shows you how to generate and use your own security user exits. To do this, you must generate them on the Client Application System and the MERVA Server System.

The following sets of sample security user exits are provided (for details refer to "User Exit Interfaces" on page 37):

enm4ssec These routines contain sample code for encryption and authentication. They show you how to access the variables of the formal parameter list in the function call. They do not provide genuine security.

enm4snil These routines do not contain code. Use this file if you want to avoid encryption or authentication.

In the Remote MERVA API Client, the shared library that contains the user exits is called **enmnxit.dll**. If you want to add your own user exits, you must replace the library **enmnxit.dll** with your own library. To do this, use **enmnxit.mak**.

In the Remote MERVA API Server, the dynamic link library that contains the user exits must have the name **enmcrxit.dll**. The shipped version of **enmcrxit.dll** is a copy of the sample library **enmcrnil.dll**. If you want to use **enmcrsec.c** of the sample user exit routines, build **enmcrxit.dll** from **enmcrsec.c** using **enmnrsec.mak**.

Generating Security User Exits on the Remote MERVA API Client

On the Remote MERVA API Client, you must use the user exit routines from shared libraries. To replace the sample user exits by your own routines, use **enm4ssec.c** as a skeleton. You then generate a shared library that replaces **enmnsxit.dll**.

To do this, replace all occurrences of **enm4snil** in **enmnsxit.mak** with **enm4ssec**. To start the compilation of the **dll**, enter:

nmake all /f enmnsxit.mak

This generates a new **enmnsxit.dll** with the user exits that are contained in **enm4ssec.c**. Replace the previous version of the **dll** with the new one.

Generating Security User Exits on the MERVA Server System

On the Remote MERVA API Server, you must use the user exit routines from shared libraries. If you want to replace the sample user exits with your own routines, use **enmcrsec.c** as a skeleton.

The **userexit** subdirectory of the installed MERVA Server System contains the make file **enmnrsec.mak**. To create the new **dll** from **enmcrsec.c**, enter:

nmake /f enmnrsec.mak

Replace the previous **enmcrxit.dll** with the new **enmcrxit.dll**.

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

Appendix A. Diagnosis Information

The diagnosis information is written to log files on the Remote MERVA API Client system and on the Remote MERVA API Server system.

Log Files on the Remote MERVA API Client

On the Remote MERVA API Client, the diagnosis log and the programmer's log are created. You can specify the names and directories of the logs in the corresponding MERVA Connection/NT profile. For details refer to "Customizing MERVA Connection/NT" on page 5.

Each message that is written to the logs consists of a message header and a message body.

The following example shows a diagnosis log with API trace entries for MERVA Connection/NT:

```
*  19990402192358ENM4RAPI ENMRestartRAPI 00000 00000
ENM9153: API function ENMRestartRAPI called.
         Parameters:
         App: SAMPLE3

*  19990402192357ENM4RUTL APIInit 00000 00000
ENM9108: Error in CPIC Call  CMALLC  RC = 19.

*  19990402192413ENM4RAPI ENMRestartRAPI
ENM9109: Error in RAPI Initialization.

*  19990402192413ENM4RAPI ENMRestartRAPI
ENM9152: API function returned with reason code 2130.
```

Figure 5. Diagnosis Log with API Trace Entries for MERVA Connection/NT

Diagnosis Log

The diagnosis log contains the following information:

- Error messages that help you recover from errors that occur when you use the API calls or from errors that refer to the communication with the MERVA system.
- Trace information when the API trace is started with the call **ENMTrace**. For details refer to *MERVA USE & Branch for Windows NT Application Programming Guide*.

Programmer's Log

The programmer's log is a debugging tool. It contains the same entries as the diagnosis log. Additionally, it contains information that can be analyzed by your IBM representative.

The layout of the header is:

Date In the form of **YYYYMMDD**, where **YYYY** denotes the year, **MM** the month, and **DD** the day.

Time	In the form of **HHMMSS**, where **HH** denotes the hour, **MM** the minutes, and **SS** the seconds.
Module name	A code that consists of 8 characters. It identifies the module from which the message comes.
Function name	A code that consists of 15 characters. It identifies the function from which the message comes.

The layout of the message is:

Message	The message that is to be recorded. The length of the message can vary. For an explanation of the message, refer to *Messages and Codes*. To access *Messages and Codes* double-click on the Messages and Codes icon on your desktop, or select it and press Enter.

Note: Log entries are appended to the existing files. If MERVA Connection/NT should create new log files, you must delete the old log files.

Log Files on the Remote MERVA API Server System

MERVA log files contain diagnosis information about the Remote MERVA API Server program. The diagnosis log contains error and trace information. The programmer's log contains IBM service information.

You can view the contents of the diagnosis log file by using the function **Display Diagnosis Log** of the MERVA Main Menu program.

The log files are located in the MERVA instance logging directory. For more information refer to the *MERVA USE & Branch for Windows NT Diagnosis Guide*.

Appendix B. Sample SNA Definitions for MERVA Connection/NT

MERVA Connection/NT uses LU 6.2 sessions for the communication between the Remote MERVA API Client and Server in the SNA Data Communication environment. To bind the required sessions, you can customize the data communication subsystems in the client and server systems in several ways.

The first customization example uses an APPN (R) network node. You can use this example only if an APPN network node is available in the LAN.

The second example uses an APPC peer-to-peer connection for the communication. You do not need a network node to use it.

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

APPN1
The name of the sample network.

NNA The name of the control point. The sample token ring address of **NNA** is **10005aa99ff0**.

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

EN1 The control point name (CP) of the end node.

TR1 The name of the Token Ring Link Station in **EN1** that provides the link to the network node server (**NNA**) in example 1 or to the peer node (**EN2**) in example 2 .

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

The following naming conventions apply for the second node, the MERVA Server System:

EN2 The control point name (CP) of the end node.

TR2 The name of the Token Ring Link Station in **EN2** that provides the link to the network node server (**NNA**) in example 1.

LU2 The name of an independent LU 6.2 in **EN2**.

ENMRAS
The name of the transaction program (MERVA Connection/NT Server) on the server node.

Customizing an APPN End Node

For a detailed description of how to configure an end node in a two-node APPN network, refer to the *Communications Server for Windows NT User's Guide*. It is assumed that you are familiar with this description.

To set up the local node, start the **SNA Node Configuration** of the Communications Server and enter the corresponding parameters.

The following tables show the different APPN configuration parameters.

Table 1. Configure Node

Parameter	Server Side	Client Side
Basic		
Fully qualified CP name	APPN1.EN2	APPN1.EN1
CP alias	EN2	EN1
Local node ID	No changes	No changes
Advanced		
Registration with network node server	Yes	Yes
Registration with central directory server	No	No
All others	No changes	No changes

Table 2. Configure Devices (DLC:LAN)

Parameter	Server Side	Client Side
Adapter number	0 (Use the first available adapter number)	0
All others	No changes	No changes

Table 3. Configure Connections (DLC:LAN)

Parameter	Server Side	Client Side
Basic		
Link station name	TR2	TR1
Destination address	–	Insert network address of the network node[1] (10005aa99ff0)
Advanced		
APPN support	–	–
Activate link at start	Yes	Yes
Link to preferred NN server	Yes	Yes
Adjacent Node		
Adjacent CP name	APPN1.NNA	APPN1.NNA
Adjacent CP type	Network node	Network node
All others	–	No changes

[1] On Windows NT, the network address of a network adapter can be retrieved with the Windows NT Diagnose program (Select **Network** and click **Transports**.

Table 4. Configure Partner LU 6.2

Parameter	Server Side	Client Side
Basic		
Partner LU name	APPN1.LU1	APPN1.LU2

Table 4. Configure Partner LU 6.2 (continued)

Parameter	Server Side	Client Side
Partner LU alias	LU1	LU2
Fully qualified CP name	APPN1.NNA	APPN1.NNA
Advanced		
Conversation security support	Yes	Yes
All others		

Table 5. Configure Modes

Parameter	Server Side	Client Side
Basic		
Mode name	APPCLU62	APPCLU62
Advanced		
Maximum RU size	1 024	1 024
All others	No changes	No changes

Table 6. Configure Local LU 6.2

Parameter	Server Side	Client Side
Basic		
Local LU name	LU2	LU1
Local LU alias	LU2	LU1
LU session limit	0 (No limit)	16
All others		

Table 7. Configure CPI-C Side Information

Parameter	Server Side	Client Side
Basic		
Symbolic destination name	–	MERVA
Mode name	–	APPCLU62
Partner LU name	–	APPN1.LU2
TP name	–	ENMRAS
Service TP	–	No
Security		
Conversation security	–	Program
Security user ID	–	SAMPLE (Your choice)
Security password	–	SAMPLE1 (Provide password)

Table 8. Configure Transaction Programs

Parameter	Server Side	Client Side
Basic		
TP name	ENMRAS	–
Complete path name	C:\Merva\Use_Branch\bin\	enmcrtpi.exe

Table 8. Configure Transaction Programs (continued)

Parameter	Server Side	Client Side
Program parameters	ENMRAS merva1 TS=0 TP=C:\temp	–
Conversation security required	Yes	–
All others	No changes	
Advanced		
Background process	Yes	–

Table 9. Configure User ID and Password

Parameter	Server Side	Client Side
Security user ID	SAMPLE (Your choice)	–
Security password	SAMPLE1 (Applicable password)	–

Customizing an APPC Peer-to-Peer Connection

The following tables describe a basic peer-to-peer connection between the client and the server side of MERVA Connection/NT. If you want to use the example, replace the network name **NETNAME** with your network name. You must also use the LAN destination address of your own server.

To configure the profiles, start the **SNA Node Configuration** of the Communications Server and enter the corresponding parameters.

The following tables show the different APPC peer-to-peer configuration parameters.

Table 10. Configure Node

Parameter	Server Side	Client Side
Basic		
Fully qualified CP name	NETNAME.EN2	NETNAME.EN1
CP alias	EN2	EN1
Local node ID	No changes	No changes
Advanced		
Registration with network node server	No	No
Registration with central directory server	No	No
All others	No changes	No changes

Table 11. Configure Devices (DLC:LAN)

Parameter	Server Side	Client Side
Adapter number	0 (Use the first available adapter number)	0 (Use the first available adapter number)

Table 11. Configure Devices (DLC:LAN) (continued)

Parameter	Server Side	Client Side
All others	No changes	No changes

Table 12. Configure Connections (DLC:LAN)

Parameter	Server Side	Client Side
Basic		
Link station name	–	TR1
Destination address	–	Insert network address [1]
Advanced		
APPN support	–	No
Adjacent Node		
Adjacent CP name	–	NETNAME.EN2
Adjacent CP type	–	End node
All others	–	No changes

[1] On Windows NT, the network address of a network adapter can be retrieved with the Windows NT Diagnose program (Select **Network** and click **Transports**.

Table 13. Configure Partner LU 6.2

Parameter	Server Side	Client Side
Basic		
Partner LU name	NETNAME.LU1	NETNAME.LU2
Partner LU alias	LU1	LU2
Fully qualified CP name	NETNAME.EN1	NETNAME.EN2
Advanced		
Conversation security support	Yes	Yes
All others		

Table 14. Configure Modes

Parameter	Server Side	Client Side
Basic		
Mode name	APPCLU62	APPCLU62
Advanced		
Maximum RU size	1,024	1,024
All others	No changes	No changes

Table 15. Configure Local LU 6.2

Parameter	Server Side	Client Side
Basic		
Local LU name	LU2	LU1
Local LU alias	LU2	LU1
LU session limit	0 (No limit)	0 (No limit)

Table 15. Configure Local LU 6.2 (continued)

Parameter	Server Side	Client Side
All others		

Table 16. Configure CPI-C Side Information

Parameter	Server Side	Client Side
Basic		
Symbolic destination name	–	MERVA
Mode name	–	APPCLU62
Partner LU name	–	NETNAME.LU2
TP name	–	ENMRAS
Service TP	–	No
Security		
Conversation security	–	Program
Security user ID	–	SAMPLE (Your choice)
Security password	–	SAMPLE1

Table 17. Configure Transaction Programs

Parameter	Server Side	Client Side
Basic		
TP name	ENMRAS	–
Complete path name	`C:\Merva\Use_Branch\bin\enmcrtpi.exe`	–
Program parameters	ENMRAS merva1 TS=0 TP=C:\temp	–
Conversation security required	Yes	–
All others	No changes	
Advanced		
Background process	Yes	–

Table 18. Configure User ID and Password

Parameter	Server Side	Client Side
Security user ID	SAMPLE (Your choice)	–
Security password	SAMPLE1	–

Appendix C. Notices

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

IBM may have patents or pending patent applications covering subject matter described in this document. The furnishing of this document does not give you any license to these patents. You can send license inquiries, in writing, to:

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

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

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

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

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

Licensees of this program who wish to have information about it for the purpose of enabling: (i) the exchange of information between independently created programs and other programs (including this one) and (ii) the mutual use of the information which has been exchanged, should contact:

IBM Deutschland
Informationssysteme GmbH
Department 3982
Pascalstrasse 100

70569 Stuttgart
Germany

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

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

The following paragraph does apply to the US only.

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

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:

- APPN
- CICS/ESA
- DB2
- IBM
- IMS/ESA
- MVS/ESA
- OS/2
- RACF
- VisualAge

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

Java and all Java-based trademarks and logos are trademarks of Sun Microsystems, Inc. in the United States, other countries, or both.

Microsoft, Windows, Windows NT, and the Windows logo are trademarks of Microsoft Corporation in the United States, other countries, or both.

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

Glossary of Terms and Abbreviations

This glossary defines terms and abbreviations as they are used in the MERVA books. 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

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

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

AP. Application.

APC. Application Control.

APAR. Authorized Program Analysis Report.

APDU. Application Protocol Data Unit.

API. Application Programming Interface.

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

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

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

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

AS. Application Support.

ASCII. American Standard Code for Information Interchange.

ASL. Application Support Layer.

ASP. Application Support Process.

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

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

authentication. The S.W.I.F.T. security check to ensure that a message is not changed during transmission and that a message is sent by an authorized sender.

authenticator key. A set of alphanumeric characters used to check the authentication of a message sent via the S.W.I.F.T. network.

authenticator-key file. A file that contains the keys to authenticate messages. It also contains a record for each correspondent bank.

B

Bank Identifier Code (BIC). The S.W.I.F.T. address of a bank as assigned by S.W.I.F.T. See also *S.W.I.F.T. address*.

BCR. Basic Card Reader.

BIC. Bank Identifier Code. See also *S.W.I.F.T. address*.

bi-directional key. A bilateral key that authenticates messages sent to and received from a correspondent.

bilateral key. A key that is generated inside an SCR. It authenticates financial messages interchanged with two correspondents. A bilateral key can be bi-directional or uni-directional.

bilateral key exchange (BKE) service. The S.W.I.F.T. USE service in which authenticator keys are generated in an SCR and exchanged via the S.W.I.F.T. network instead of being exchanged by mail.

BK. Bilateral Key.

BKE. Bilateral Key Exchange.

BK ID. Bilateral Key Identifier. The BK ID has the following format:
- The first character is either B (Bilateral) or M (Manual).
- The second character is the BK type, as defined by S.W.I.F.T.
- Characters 3 to 8 denote the date.
- Characters 9 to 16 denote the key check value.

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

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

C

CBT. S.W.I.F.T. Computer-Based Terminal.

certificate. A guarantee by S.W.I.F.T. that the holder of a public key is genuine. You need a certificate for each public key that you want to generate before you can start bilateral key exchange.

CHK. checksum trailer.

CID. Central Institution Destination.

Communication Services (CS). With CS, you can use Communications Server or Personal Communications.

Control Center. See *MERVA Control Center*.

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

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

correspondents database. A database that contains the S.W.I.F.T. address, nickname, descriptive name, and address of each bank with which your bank corresponds. The file is used to store the descriptive names and addresses that are needed in the address expansion process.

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

CRC. Cyclic Redundancy Check.

CS. Communication Services.

CUG. Closed User Group.

CV. See *certificate*.

CV ID. Certificate Identity. A unique identifier of a certificate that consists of the destination, expiring date, and number of the certificate.

D

destination. For S.W.I.F.T., the first 8 characters of the S.W.I.F.T. address that consists of the bank, country, and location codes.

DTE. Data Terminal Equipment.

DTR. Data Terminal Ready.

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

E

emitting destination. The S.W.I.F.T. destination that is shown on messages sent to S.W.I.F.T. You must specify the emitting destination, for example, when you send a message to S.W.I.F.T. to request the blacklisting of a card reader.

F

FIN. Financial Application (S.W.I.F.T.).

four-eyes principle. A banking security concept in which changes and the approval of changes must always be done by two different people.

I

IAM. Interapplication Messaging.

ICC. Integrated Circuit Card.

IM-ASPDU. Interapplication Messaging Application Support PDU. It contains an application message and consists of a header and a body.

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

Interapplication messaging (IAM). Interapplication messaging is used as a MERVA Link message exchange protocol.

ISC. Intersystem Communication.

ISN. Input Sequence Number.

ISO. International Organization for Standardization.

K

kernel. A secret value stored on a USER ICC for each LT to define access rights to S.W.I.F.T. applications and to generate session keys. Each USER ICC has eight kernels.

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

key check value. (1) Part of the *BK ID*. If you encounter problems when you communicate with your correspondent, check whether the key check value is identical to your correspondent's key value. (2) Part of the *secure transmission key (STK)*, to check whether you have entered the remainder of the STK correctly.

KMA. Key Management Authority.

L

LAK. Login Acknowledgment Message. This message informs you that you have successfully logged on to the S.W.I.F.T. network.

LNK. S.W.I.F.T. login negative acknowledgment message. This message informs you that the login to the S.W.I.F.T. network has failed.

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

logging database. Contains all MERVA audit logging data.

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

LSN. Login Sequence Number.

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

LU name. Name of the Logical Unit.

M

MAC. Message Authentication Code.

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

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

MERVA Control Center. A program to:
- Start a MERVA instance.
- Stop a MERVA instance.
- Show the status of a MERVA instance.
- Maintain MERVA databases.

MERVA domain. See *domain*.

MERVA Link. The component to interconnect MERVA systems.

MERVA Workstation. Message Entry and Routing with Interfaces to Various Applications USE & Branch for Windows NT.

message. A string of fields in a predefined form to provide or request information. See also *S.W.I.F.T. message*.

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

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

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

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

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

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

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

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

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

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

message transfer. The name of the lower sublayer functionality of MERVA Link.

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

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

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

MRN. Message Reference Number.

msg ID. Message Identifier.

MSN. Message Sequence Number.

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

MTP. Message Transfer Process or Program.

N

nested message. A message that is composed of one or more message types. For example, SWIFT MT 195 could be used to request information about a S.W.I.F.T. MT 100. The S.W.I.F.T. MT 100 (only mandatory fields) is then nested in S.W.I.F.T. MT 195.

network identifier. A single character stored with the message in the MERVA message database that shows which network is to be used to send the message. For example, S for S.W.I.F.T.

NCP. Network Control Program.

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

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

O

OSI. Open System Interconnection.

OSN. Output Sequence Number.

P

PAC. Proprietary Authentication Code.

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

PDE. Possible Duplicate Emission.

PDU. Protocol Data Unit.

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

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

PIN. Personal Identification Number.

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

protocol data unit (PDU). In MERVA Link, a PDU consists of a structured sequence of implicit and explicit data elements:
- Implicit data elements contain other data elements.
- Explicit data elements do not contain any other data elements.

PSN. Public Switched Network (connection).

PSPDN. Packet Switched Public Data Network.

PSTN. Public Switched Telephone Network.

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

PTF. Program Temporary Fix.

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

PU. Physical Unit.

public key. A key with which an institution enciphers a bilateral key received from a correspondent. See also *secret key*.

purpose group. A logical grouping of queues associated with a function. The function processes the messages to all queues that belong to the purpose group.

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

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

Q

queue. See *message queue*.

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

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

R

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

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

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

routing condition. A logical test to determine the target queues to which messages are sent. Routing conditions are defined for source queues. A source queue is the queue from which messages are taken for further routing. You can check:
- The presence of a field within a message

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

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

S

scan pattern. A character string that is placed between message fields to identify where a field begins. It is also known as a tag.

SCR. Secure Card Reader.

SDLC. Synchronous Data Link Control.

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

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

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

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

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

SK. Session Key.

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

SLS. Secure Login and Select.

SMC. Security Management Center.

SNA. Systems Network Architecture.

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

SSN. Select Sequence Number.

STK. Secure Transmission Key.

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

S.W.I.F.T. Society for Worldwide Interbank Financial Telecommunication, s.c. (S.W.I.F.T.).

S.W.I.F.T. II. Refers to the S.W.I.F.T. II network of the Society for Worldwide Interbank Financial Telecommunication, s.c. (S.W.I.F.T.).

S.W.I.F.T. address. A code used to identify a bank within the S.W.I.F.T. network. The code is also called a bank identifier code (BIC) or a terminal identifier. It is assigned by S.W.I.F.T.

S.W.I.F.T. correspondents database. The database that contains the S.W.I.F.T. address or BIC, together with the name, postal address, and zip code of each financial institution in the BIC directory.

S.W.I.F.T. destination address. The first 8 characters of the S.W.I.F.T. address that consist of the bank, country, and location codes.

S.W.I.F.T. financial message. A message in the S.W.I.F.T. categories 1 to 9 that you can send or receive via the S.W.I.F.T. network. See *S.W.I.F.T. input message* and *S.W.I.F.T. output message.*

S.W.I.F.T. header. The leading part of a message that contains the sender and receiver of the message, the message priority, and the type of message.

S.W.I.F.T. input message. A S.W.I.F.T. message prepared by a user to be sent to the S.W.I.F.T. network.

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

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

S.W.I.F.T. output message. A S.W.I.F.T. message from the S.W.I.F.T. network.

S.W.I.F.T. system message. A message in S.W.I.F.T. category 0.

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

T

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

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

TCT. Terminal Control Table.

technology flag. A parameter that is controlled by the USOF. It tells S.W.I.F.T. which access technology, ICCs, or paper tables are used by the LTs of a particular destination.

TNG. Training trailer.

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

TRN. Transaction Reference Number.

U

UKMO. User Key Management Officer.

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

USE. User Security Enhancements.

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

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

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

user security officer (USOF). The administrator who is the holder of a USOF ICC. The USOF is responsible to control and manage ICCs, card readers, and their related data.

USOF. User Security Officer.

W

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

X

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

Bibliography

IBM Publications

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

- *MERVA Documentation CD*, SK2T-9752

MERVA ESA Components Books

- *MERVA ESA Components Licensed Program Specifications*, GH12-6333
- *MERVA USE & Branch for Windows NT User's Guide*, SH12-6334
- *MERVA USE & Branch for Windows NT Installation and Customization Guide*, SH12-6335
- *MERVA USE & Branch for Windows NT Application Programming Guide*, SH12-6336
- *MERVA USE & Branch for Windows NT Diagnosis Guide*, SH12-6337
- *MERVA USE & Branch for Windows NT Migration Guide*, SH12-6393
- *MERVA USE Administration Guide*, SH12-6338
- *MERVA Connection/NT*, SH12-6339
- *MERVA Connection/400*, SH12-6340
- *MERVA Message Processing Client for Windows NT User's Guide*, SH12-6341
- *MERVA Automatic Message Import/Export Facility User's Guide*, SH12-6389
- *MERVA Workstation Based Functions*, SH12-6383
- *MERVA ESA V4 Traffic Reconciliation Guide*, SH12-6392
- *MERVA ESA V4 Directory Services*, SH12-6367

MERVA ESA Books

- *MERVA ESA V4 Licensed Program Specifications*, GH12-6373
- *MERVA ESA V4 Application Programming Interface Guide*, SH12-6374
- *MERVA ESA V4 Operations Guide*, SH12-6375
- *MERVA ESA V4 User's Guide*, SH12-6376
- *MERVA ESA V4 Macro Reference*, SH12-6377
- *MERVA ESA V4 Installation Guide*, SH12-6378
- *MERVA ESA V4 Messages and Codes*, SH12-6379
- *MERVA ESA V4 Customization Guide*, SH12-6380
- *MERVA ESA V4 Concepts and Components*, SH12-6381
- *MERVA ESA V4 Diagnosis Guide*, SH12-6382
- *MERVA ESA V4 Advanced MERVA Link*, SH12-6390
- *MERVA ESA V4 System Programming Guide*, SH12-6366

Further IBM Publications

- *DB2 Administration Guide*, S10J-8157
- *DB2 Building Applications for Windows and OS/2 Environment*, S10J-8160
- *DB2 API Reference*, S10J-8167
- *DB2 Troubleshooting Guide*, S10J-8169
- *eNetwork Personal Communications Version 4.2 for Windows 95 and Windows NT Quick Beginnings*, GC31-8476
- *eNetwork Personal Communications Version 4.2 for Windows 95 and Windows NT Reference*, GC31-8477
- *CID Enablement Guidelines*, S10H-9666
- *CICS-RACF Security Guide*, SC33-1185
- *ITSC Redbook APPC Security: MVS/ESA, CICS/ESA, and OS/2*, GG24-3960
- *IMS/ESA Version 4 Data Communication Administration Guide*, SC26-3060

S.W.I.F.T. Publications

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

- *S.W.I.F.T. User Handbook*
- *S.W.I.F.T. Dictionary*
- *S.W.I.F.T. Directory*
- *S.W.I.F.T. FIN Security Guide*
- *S.W.I.F.T. Card Readers User Guide*
- *S.W.I.F.T. Security Features Technical*

Index

S

T

U

V

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

MERVA ESA Components
MERVA Connection/NT
Version 4 Release 1

Publication No. SH12-6339-02

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

	Very Satisfied	Satisfied	Neutral	Dissatisfied	Very Dissatisfied
Overall satisfaction	☐	☐	☐	☐	☐

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

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

Please tell us how we can improve this book:

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

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

Name _____ Address _____

Company or Organization _____

Phone No. _____

Readers' Comments — We'd Like to Hear from You
SH12-6339-02

Cut or Fold
Along Line

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

PLACE
POSTAGE
STAMP
HERE

IBM Deutschland Entwicklung GmbH
Information Development & User Centered Design
Dept. 0446
Schoenaicher Strasse 220
71032 Boeblingen
Germany

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

Cut or Fold
Along Line

Program Number: 5648-B30

SH12-6339-02