MERVA ESA Components

MERVA Connection/400

Version 4 Release 1

SH12-6340-01

MERVA ESA Components

MERVA Connection/400

Version 4 Release 1

SH12-6340-01

Note!

Before using this information and the product it supports, be sure to read the general information under “Appendix E. Notices” on page 95.

Second Edition, December 1999

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.

Contents

About This Book

Read this book to find out how to work with MERVA Connection/400. You learn how to install and customize MERVA Connection/400, 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 the 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 AS/400 (R).

This book also helps you install and customize MERVA Connection/400.

It is assumed that you have prior knowledge of and experience with:
- AS/400
- Operating System/400 (R) (OS/400) (R)
- Windows NT
- Systems Network Architecture (SNA)
- Application Programming Interface (API) of MERVA

How This Book is Organized

The first chapter of this book provides general information about MERVA Connection/400 by giving an overview of the product. Chapter 2 describes how to install MERVA Connection/400. Chapter 3 tells you how to define and customize the network. Chapter 4 tells you how to work with the API and how to build API programs. It also helps you understand resynchronization. Chapter 5 shows you the API for RPG/ILE. Chapter 6 covers aspects of security, such as encryption, authentication, and user exits. The appendixes contain diagnosis information and several samples.

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.

Chapter 1. General Introduction to MERVA Connection/400

This chapter introduces MERVA Connection/400 and briefly describes the facilities supported by MERVA Connection/400.

Objectives

There is a wide range of banking applications available for the AS/400 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. MERVA also provides an application program interface (API) to access specific MERVA services.

To use AS/400 applications as banking applications, you must transfer messages that are created on the AS/400 to MERVA. Messages that you receive from one of these networks must be transferred from MERVA to the AS/400.

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 the AS/400 to MERVA. The MERVA system then works as if it were a component of the application.

MERVA Connection/400 is your direct connection. It is, however, not a ready-to-use S.W.I.F.T. interface on the AS/400. It does not have a user interface.

MERVA Connection/400 provides you with an interface for application programs on your operating system. It is called the Remote MERVA API. By using the Remote MERVA API, you can create an application on the AS/400 to send messages to MERVA and receive messages from MERVA with a minimum of effort.

Functions

MERVA Connection/400 has the same functions as the MERVA API on the MERVA system. Additionally, it offers you the following functions:
- Calls that help you establish an intersystem connection.
- A real-time interface that allows you to connect to MERVA.
- A C/ILE-Language interface that guarantees easy portability of MERVA API programs between Windows NT and AS/400.
- An interface for RPG/ILE and other ILE languages.
- Code conversion from EBCDIC to ASCII, and from ASCII to EBCDIC formats.
- A flexible user exit interface with which you can handle security aspects.
- A resynchronization mechanism ensures that the Remote MERVA API program on the AS/400 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/400:

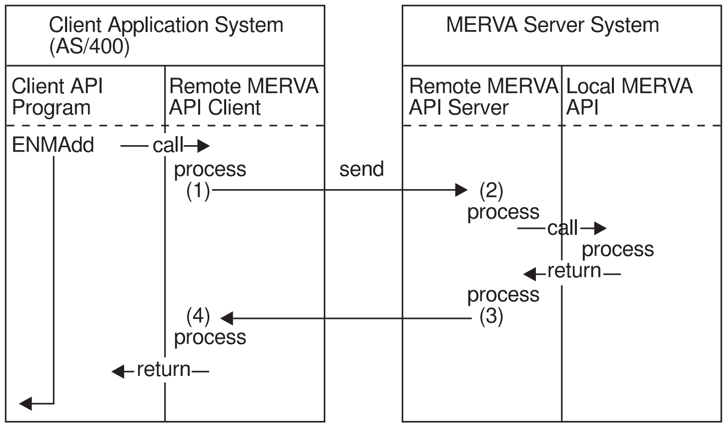

Figure 1. Concept of MERVA Connection/400

MERVA Connection/400 has the following components:

- The Remote MERVA API Client is installed and runs in the Client Application System. The Client Application System is the AS/400. MERVA is not installed in the Client Application System.
- The Remote MERVA API Server is installed and runs in the MERVA Server System. The MERVA Server System is a Windows NT 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 on the AS/400. 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 of MERVA Connection/400 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/400.

Machine Requirements

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

- You can install the Remote MERVA API Client on any OS/400 system.
- You must connect the Client Application System of MERVA Connection/400 to the MERVA Server System with a Data Communication Link. The Data Communication Service (SNA APPC) defines the type of intersystem link that you can use, such as token ring, SDLC, or Twinax.

Program Requirements

The following section shows the program requirements for MERVA Connection/400:

- OS/400 Version 3.1, including the latest available Corrective Service Diskette (CSD) level, or a subsequent release.
- A compiler for one of the following languages:
  - IBM C/ILE
  - IBM RPG/ILE
  - IBM COBOL/ILE

Installing MERVA Connection/400

The MERVA Connection/400 installation tape contains the objects **ENMRAPI.SAVF** and **ENMRSMP.SAVF**. The files contain the following libraries:

ENMRAPI A library saved with the command **SAVLIB** that contains the programs of MERVA Connection/400 and the compiled sample user exits.

ENMRSMP A library saved with the command **SAVLIB** that contains the compiled sample programs and the following files:

APILOG	Diagnosis and programmer's logging members
DATA	Message members used by C/ILE sample programs
INI	Sample profiles (for details refer to "Customizing MERVA Connection/400" on page 4)
MIP	Message integrity control file members
QRPGLESRC	Source of sample API programs in RPG/ILE language

QSECSRC	Source of sample security user exits in C/ILE language
RSAMPLE1	Messages used by RPG sample program
MERVA	Sample Communication Side Information for MERVA (for details refer to "Customizing MERVA Connection/400")

To restore the libraries **ENMRAPI** and **ENMRSMP** from the save files, enter the command **RSTLIB** in the command line.

Customizing MERVA Connection/400

To customize MERVA Connection/400 in the Client Application System, you have to change the information in the MERVA Connection/400 profile and libraries. You must do this in the search list of the job that runs the API program.

Changing Profile Settings

The profile of MERVA Connection/400 contains information about logging, network partner, and internal customization parameters. You can have several profiles. To define the profile that is to be used by the calling program, use the API call **ENMSetProfile**. For a description of this call refer to "Structure of the MERVA API Program on the Client Side" on page 9.

Each profile is a member of a source physical file.

The following table shows you the format of a MERVA Connection/400 profile. The sample profiles are **ENMRSMP/INI(API)**.

Table 1. MERVA Connection/400 Profile

Line	Information	Sample
1	Logging level (1..4)	1
2	Name of programmer's log	ENMRSMP/APILOG(PLOG)
3	Name of diagnosis log	ENMRSMP/APILOG(DIAG)
4	Name of CSI[1] object	MERVA
5	Name of message integrity[2] control file	ENMRSMP/MIP
6	System type (AS400)	AS400

Note:

[1] Communication Side Information - object in CPI Communications containing initialization parameters. These are, for example:

- The name of the partner program (such as the Remote MERVA API Server) with which a program can establish a conversation.
- The name of the logical unit (LU) at the partner program's node that CPI Communications needs to establish a conversation.

The system-recognized identifier on the AS/400 for the object type is *CSI.

[2] The message integrity control file is used by the internal integrity processing of MERVA Connection/400, described in Figure 4 on page 20.

Note: Concurrently running jobs must refer to different profiles, even if they call the same API program. You must specify different logging files in each profile (lines 2 and 3 of the profile) and different message integrity control file names (line 5 of the profile).

Required Libraries

The following libraries must be in the search list of the job that runs the API program:

QSYS2	CPI-C programs used by MERVA Connection/400
ENMRAPI	MERVA Connection/400 programs
ENMRSMP	MERVA CSI object.

Network Definitions

The connection between the AS/400 and the Windows NT system must be an LU 6.2 session.

To use the CPI Communications interface, you must define communications side information (CSI). For samples of CSI, mode, device, and controller definitions refer to "Appendix B. Sample Network Definitions for the AS/400" on page 87. The library **ENMRSMP** that is supplied with MERVA Connection/400 contains a sample for CSI objects.

For the APPC connection, you must define controller, device, and mode description. For samples refer to "Appendix B. Sample Network Definitions for the AS/400" on page 87.

You can also define only the LU 6.2 session on the Windows NT system and establish the session to the AS/400 from this session. For example, you can define a 5250 emulation and start it. The **QLUS** system job of the AS/400 creates the necessary descriptions automatically if the parameter **AUTOCRTCTL** is set to **YES** in the line description.

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:

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

Customizing the Communications Server

MERVA Connection/400 can use SNA APPC services for the communication between the Remote MERVA API Client and Server.

You must install and customize Personal Communications for Windows NT in the server system to bind APPC sessions between the two partner systems.

Basic SNA Customization

You can connect any MERVA Connection/400 system to MERVA USE & Branch for Windows NT.

For a description of the respective customization, refer to the corresponding documentation about Communications Server for Windows NT listed in "Bibliography" on page 105.

For a sample of the SNA customization that is independent of MERVA Connection/400, refer to "Appendix C. Sample Network Definitions for Windows NT" on page 89.

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**.

Verifying the Installation

To verify that the installation and customization of MERVA Connection/400 is correct, run the sample program **SAMPLE4**. Before you can run the sample program, the user ID **SAMPLE** with the password **SAMPLE1** has to be defined in MERVA. The user ID has to be approved for application programs. The program checks that the queues **API_IN** and **API_OUT** are customized.

For a detailed description of the prerequisites for **SAMPLE1**, **SAMPLE2**, **SAMPLE2S**, and **SAMPLE3**, refer to the *MERVA USE & Branch for Windows NT Application Programming Guide*.

Chapter 4. The Application Programming Interface

The following description of the API is based on the description in the *MERVA USE & Branch for Windows NT Application Programming Guide*.

Structure of the MERVA API Program on the Client Side

The MERVA API program on the AS/400 must call functions that connect and disconnect to and from the MERVA system. The following figure shows 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 start the Remote MERVA API Client on the AS/400 by calling the function **ENMSetProfile**. This function tells the Remote MERVA API Client the name of the profile. For a description of the profile refer to “Changing Profile Settings” on page 4.

2 After the profile name is set, you can connect to the Remote MERVA API Server on MERVA. To do this, call the function **ENMStartRAPI**. After this function is called, the Remote MERVA API Client is started, and the network connection to the Remote MERVA API Server is established.

With the call **ENMStartRAPI**, you can call the API functions as if the program runs locally on MERVA.

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 stop the program. The Remote MERVA API Server does then not receive the next connection request after the API program is restarted.

C Language Data Types

The file **enm4rapi.h** contains the data types and prototypes of the MERVA API functions. When you compile a MERVA API program locally on Windows NT, the file **enmcapi.h** is automatically included. When you compile a MERVA Connection/400 API program on the AS/400, you must include the file **enm4rapi.h** instead of **enmcapi.h**.

The meaning of the data types that are used to describe the API calls of this book are shown in the following table:

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

Additional Functions

MERVA Connection/400 provides more API calls than the MERVA API.

The calls contain the following categories:
- Functions to start and end the conversation
- Functions for error handling

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 functions:

ENMSetProfile	Select a Profile
ENMStartRAPI	Establish connection to MERVA
ENMRestartRAPI	Reconnect Remote API program to MERVA
ENMEndRAPI	Disconnect from MERVA

Each function is described in detail in the following sections.

ENMSetProfile - Select a Profile

Specifies the name of the profile that you want to use. For details refer to "Changing Profile Settings" on page 4.

C Definition

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

Parameter Description

The following parameters are required:

- **pucProfileName**(PUCHAR)

  Pointer to a null-terminated string up to 80 characters.

  Note: If several API programs run concurrently, you must use a different name for each program. For example, you can use the transaction code as a unique name.

Remarks

For a description of the format and contents of the profile, refer to "Changing Profile Settings" on page 4.

C Language Example

```
 #include "enm4rapi.h"

ENMSetProfile ("ENMRSMP/INI(API)");
```

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 15. The reason code is also written to the diagnosis log of the AS/400. For details refer to “Appendix A. Diagnosis Information” on page 85. 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 name for each program. For example, you can use the transaction code as a unique name.

Remarks

This call establishes the APPC 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 "enm4rapi.h"

USHORT rc = 0;

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

ENMRestartRAPI - Reconnect Remote API Program 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 15. The reason code is also written to the diagnosis log of the AS/400. For details refer to “Appendix A. Diagnosis Information” on page 85. 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. This name is registered by the Remote MERVA API Server.

  Note: If several API programs run concurrently, you must use a different name for each program. For example, you can use the transaction code as a unique name.

Remarks

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

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

For details refer to Figure 4 on page 20.

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 use **ENMRestartRAPI** if the previous connection did not end abnormally.

C Language Example

```
#include "enm4rapi.h"

USHORT rc = 0;

if ((rc = ENMRestartRAPI ( "APPLID" )) == 0 )
    puts("APPC 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 15. The reason code is also written to the diagnosis log of the AS/400. For details refer to “Appendix A. Diagnosis Information” on page 85. If it is an internal error of the MERVA API, the reason code is **0**.

Remarks

The APPC conversation to MERVA is stopped.

C Language Example

```
 #include "enm4rapi.h"

USHORT rc = 0;

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

Handling Errors

If you want the API call to return reason codes, use the function **ENMGetReason**.

ENMGetReason

This call returns the reason code for an internal error in MERVA Connection/400.

If an internal error occurs in MERVA Connection/400 or in the local MERVA API, an API call returns the return code **2**. If it is an error of MERVA Connection/400, **ENMGetReason** returns a more specific reason code. Otherwise, the reason code is **0**.

C Definition

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

Parameter Description

The following parameter is required:

retCode(USHORT) - output

Code	**Meaning**
2xxx	Reason codes from **2000** to **2999** indicate communication problems.
2110	The APPC connection cannot be established or is canceled.
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 been stopped already.
2150	The conversation was interrupted while the program tried to receive data.
2200	An empty data buffer was received.
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 allocated 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 cannot 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 that is specified in **ENMSetProfile** does not exist.

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

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

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

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

8017 Conversation has not been started with **ENMStartRAPI**.

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

C Language Example

```
#include "enm4rapi.h"

   USHORT   rc     = 0;
   USHORT   reason = 0;

   rc = ENMFree();
   if (rc) {
     reason = ENMGetReason();
     if (reason) {
     printf ("Internal error in Connection/400 occurred, reason code %d",
            reason);
      }
   }
```

Building API Programs

This section describes how to compile MERVA Connection/400 programs in C and RPG programming languages.

The following figure gives you an overview of the MERVA Connection/400 API program.

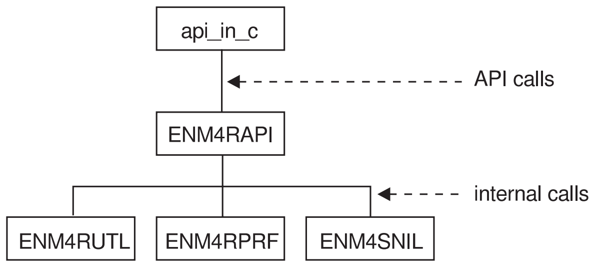

Figure 3. The MERVA Connection/400 API Program

Building ILE C/400 Language Programs

To build ILE C/400 (R) language programs:

- Compile your program to create modules.
- Bind your program.

Compiling your API Program with the CRTCMOD Command

To compile your program, use the command **CRTCMOD**. The following example shows you how to do this.

```
CRTCMOD MODULE  (MYAPILIB/api_in_c)
        SRCFILE (MYAPILIB/QCLESRC)
        SRCMBR  (api_in_c)
        TEXT    ('ILE C/400 API program for MERVA')
```

Connecting your API Program to MERVA Connection/400 Programs

The entry points for the API program on the AS/400 are located in the program **ENMRAPI/ENM4RAPI**. The program **ENM4RAPI** uses functions of the following programs:

- **ENMRAPI/ENM4RUTL**
- **ENMRAPI/ENM4RPRF**
- **ENMRAPI/ENM4SNIL**

To use the API, you must bind your compiled module to the modules that are delivered with MERVA Connection/400. To do this, use the command **CRTPGM**. The following example shows you how to do this.

```
CRTPGM PGM    (MYAPILIB/api_in_c)
       MODULE (
               MYAPILIB/api_in_c
               ENMRAPI/ENM4RAPI
               ENMRAPI/ENM4RUTL
               ENMRAPI/ENM4RPRF
               ENMRAPI/ENM4SNIL
              )
```

You can then call the API program **api_in_c** with the command **CALL**.

RPG/ILE Language Program

To build RPG/ILE language programs:

- Compile your program to create modules.
- Bind your program.

Compiling your API Program with the CRTRPGMOD Command

To compile your program, use the command **CRTRPGMOD**. The following example shows you how to do this.

```
CRTRPGMOD MODULE  (MYAPILIB/ApiInRPG)
          SRCFILE (MYAPILIB/QRPGLESRC)
          SRCMBR  (ApiInRPG)
          TEXT    ('RPG/ILE API program for MERVA')
```

The entry points for RPG/ILE language calls are located in the program **ENMRAPI/ENM4RRPG**. **ENM4RRPG** uses the same entry points of **ENM4RAPI** as an API program written in C language. The program **ENM4RAPI** uses functions of the following programs:

- **ENMRAPI/ENM4RUTL**
- **ENMRAPI/ENM4RPRF**
- **ENMRAPI/ENM4SNIL**

To use the API, you must bind your compiled module to the modules that are delivered with MERVA Connection/400. To do this, use the command **CRTPGM**. The following example shows you how to do this.

```
CRTPGM PGM    (MYAPILIB/api_in_rpg)
       MODULE (
               MYAPILIB/api_in_rpg
               ENMRAPI/ENM4RRPG
               ENMRAPI/ENM4RAPI
               ENMRAPI/ENM4RUTL
               ENMRAPI/ENM4RPRF
               ENMRAPI/ENM4SNIL
              )
```

You can then call the API program **api_in_rpg** with the command **CALL**.

"Chapter 5. API for RPG/ILE" on page 23 shows you how to call the entry points for API calls in RPG/ILE.

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 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 4 on page 20. 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:

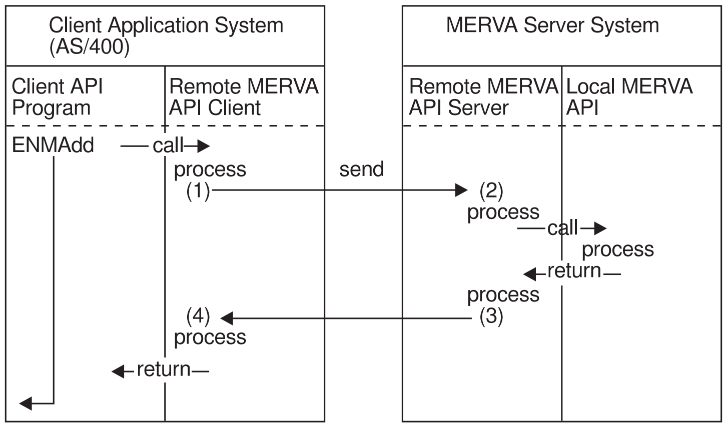

Figure 4. Resynchronization Support

The return code **ERR_SYSTEM** of the API call and the corresponding reason code (**2000** to **2999**) of an additional **ENMGetReason** call indicate whether the network connection is interrupted. MERVA Connection/400 does not know whether the call completed successfully, or whether it is not executed in the MERVA system. In the example of Figure 4, the API program does not know whether the message is added to the MERVA queue.

With MERVA Connection/400, 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.

Regarding to the example shown in Figure 4, the API program recreates the specific message and calls **ENMAdd** to add the message to the MERVA queue.

MERVA Connection/400 stores the previous state of the API call. If the call is completed in the previous run, the Remote MERVA API Server returns the return code that is created in the previous run. If the call is not completed, it is executed after restart.

An internal integrity logic of MERVA Connection/400 ensures that this procedure is carried out. Control files on the AS/400 or the MERVA side contain internal control information. Each application has a control file. The name of the control file is identical with the application name that is passed with the call **ENMSetProfile/SETPROF**. In this profile, you can define where the integrity control information is to be saved. For more information about the profile, refer to "Changing Profile Settings" on page 4.

Resynchronization is supported for the following API functions:
- **ENMAdd**
- **ENMDelete**
- **ENMPut**
- **ENMRouteAdd**
- **ENMRoutePut**

Note: If the call of **ENMAdd** or **ENMRouteAdd** fails, call **ENMClear** to clear the message space. For details refer to the *MERVA USE & Branch for Windows NT Application Programming Guide*. If the call of **ENMAdd** or **ENMRouteAdd** fails after you reestablish the connection with **ENMRestartRAPI**, **ENMClear** returns the error message **ERR_NO_MSG_CREATED**. You can ignore this message. The same applies to the call **ENMFree** that returns the error message **ERR_NO_MSG_LOCKED** after the call **ENMDelete**, **ENMPut**, or **ENMRoutePut** fails.

MERVA Connection/400 does not save type or input data of the API call that failed due to the network failure. Therefore, the API program on the AS/400 must ensure that the same call is repeated after reconnecting to MERVA.

You can use the function **ENMRestartRAPI** to establish the connection in the same way as **ENMStartRAPI**, regardless of whether a network failure occurred in the previous run of the API program. In **ENMStartRAPI**, the information about the previous state is deleted. This is different from **ENMRestartRAPI**. In **ENMRestartRAPI**, the information about the previous state is not deleted. It is used if a recovery is necessary.

Note: When you use **ENMRestartRAPI**, you must repeat the previous API calls if they are abended by a network failure in the previous run. MERVA Connection/400 does not recognize an incorrect API call.

The call is not executed if the internal state signifies that the last API call from the previous run has been executed, but only the return code had not been passed to the calling API program.

Chapter 5. API for RPG/ILE

This chapter describes API calls that use C/ILE API. For details refer to "Chapter 4. The Application Programming Interface" on page 9. For a description of the return codes, refer to "Appendix A. Diagnosis Information" on page 85. The API trace is written for internally called functions, not for the interface that is described in this chapter.

For formal parameters the following rules apply:

- All parameters must be passed by reference.
- The data type **string(n+1)** is a character buffer that ends with a binary zero. **n** is the maximum character of the string. The binary zero is not included.

  Note: If the string does not end with a zero, the buffer have the minimum character length **n+1**. The string must also be padded with blanks. The called function stops the string with zero.

Each function is described in detail in the following sections.

Establishing a Session to MERVA

You can use the following functions to handle the session with MERVA.

STRTAPPC

This function calls **ENMStartRAPI**.

ENMStartRAPI starts the APPC conversation to MERVA and initializes internal buffers and variables. After this function is called, the program must call **ENDAPPC** before it ends.

C Definition

```
VOID STRTAPPC( UCHAR  * ApplicationName,
               USHORT * RetVal);
```

Parameters

Name	Type	I/O	Comments
ApplicationName	string(8+1)	I	Unique name of the application attaching to MERVA. See the description of the **ENMStartRAPI** call for more information.
RetVal	integer(2)	O	2 bytes, return code from API call. For the values refer to **ENMStartRAPI**.

RPG/ILE Language Example

```
D DSRET           DS
D  RETVAL                 1      2B 0 INZ(0)

C                   CALLB     'STRTAPPC'
C                   PARM      'APPL'        APPLN           10
C                   PARM                    RETVAL
```

RSTRTAPC

This function calls **ENMRestartRAPI**.

ENMRestartRAPI starts the APPC conversation to MERVA and initializes internal buffers and variables. After this function is called, the program must call **ENDAPPC** before it ends.

C Definition

```
VOID RSTRTAPC( UCHAR  * ApplicationName,
               USHORT * RetVal);
```

Parameters

Name	Type	I/O	Comments
ApplicationName	string(8+1)	I	Unique name of the application attaching to MERVA. See the description of the **ENMRestartRAPI** call for more information.
RetVal	integer(2)	O	2 bytes, return code from API call. For the values refer to **ENMRestartRAPI**.

RPG/ILE Language Example

```
     D DSRET           DS
     D  RETVAL                 1      2B 0 INZ(0)

     C                   CALLB     'RSTRTAPC'
     C                   PARM      'APPL'        APPLN           10
     C                   PARM                    RETVAL
```

ENDAPPC

This function calls **ENMEndRAPI**.

ENMEndRAPI stops the conversation between the Remote MERVA API Client and the Remote MERVA API Server that is started by a **STRTAPPC** call. Another API program with the same application name can then connect to MERVA. The application name is passed by the **STRTAPPC** call.

C Definition

```
VOID ENDAPPC( USHORT * RetVal);
```

Parameters

Name	Type	I/O	Comments
RetVal	integer(2)	O	2 bytes, return code from API call. For the values refer to **ENMEndRAPI**.

RPG/ILE Language Example

```
D DSRET           DS
D  RETVAL                1      2B 0 INZ(0)

C                   CALLB     'ENDAPPC'
C                   PARM                    RETVAL
```

SETPROF

This function calls **ENMSetProfile**.

ENMSetProfile specifies the name of the profile that is to be used. For details refer to “Changing Profile Settings” on page 4.

C Definition

```
VOID SETPROF( UCHAR * ProfileName);
```

Parameters

Name	Type	I/O	Comments
ProfileName	string(80+1)	I	For example, ENMRSMP/INI(API)

RPG/ILE Language Example

```
D DSRET           DS
D  RETVAL                 1      2B 0 INZ(0)
D DSPRF           DS
D  PRFNAM                 1     20    INZ('ENMRSMP/INI(API)')
D  TERM                  21     22B 0 INZ(0)

C                   RESETPRFNAM
C                   CALLB     'SETPROF'
C                   PARM                    DSPRF
C                   PARM                    RETVAL
```

Connecting to MERVA

The following functions start and stop the connection to MERVA.

ATTACH

This function calls **ENMAttach**.

ENMAttach connects the application program to MERVA. It prepares the interface data structures to pass messages from MERVA to the application, and to pass messages from the application to MERVA.

C Definition

```
VOID ATTACH( UCHAR  * UserID,
             UCHAR  * Password,
             UCHAR  * FunctionID,
             USHORT * RetVal);
```

Parameters

Name	Type	I/O	Comments
UserID	string(8+1)	I	User ID defined in MERVA.
Password	string(8+1)	I	Password defined in MERVA.
FunctionID	string(3+1)	I	Must be API.
RetVal	integer(2)	O	2 bytes, return code from API call. For the values refer to **ENMAttach**.

Processing

The function checks with your user ID, password, and function ID whether the program is authorized to connect to MERVA.

Remarks

A MERVA API program that runs on the AS/400 is not identified by its program name but by the application name that is passed with the call **STRTAPPC**. This is different from a local MERVA API program that runs on the local Windows NT system.

RPG/ILE Language Example

```
D DSRET           DS
D  RETVAL                1      2B 0 INZ(0)

C                   CALLB     'ATTACH'
C                   PARM      'USER1'       USERID           9
C                   PARM      'MERVA2'      PASSWD           9
C                   PARM      'API'         FUNCID           4
C                   PARM                    RETVAL
```

CLEAR

This function calls **ENMClear**.

ENMClear deletes a message space that is created with the function **CREATE**.

C Definition

```
VOID CLEAR( USHORT * RetVal);
```

Parameters

Name	**Type**	**I/O**	**Comments**
RetVal	integer(2)	O	2 bytes, return code from API call. For the values refer to **ENMClear**.

RPG/ILE Language Example

```
D DSRET           DS
D  RETVAL                 1      2B 0 INZ(0)

C                   CALLB     'CLEAR'
C                   PARM                    RETVAL
```

CREATE

This function calls **ENMCreate**.

ENMCreate prepares a new, empty message for the application.

C Definition

```
VOID CREATE( USHORT * RetVal);
```

Parameters

Name	Type	I/O	Comments
RetVal	integer(2)	O	2 bytes, return code from API call. For the values refer to **ENMCreate**.

Processing

This function requests a message reference number (MRN) from MERVA to identify the new message. A **READFLD** call can set a reference to the MRN. For further processing of the message, use the call **ADDMSG** or **ROUTEADD**.

A created message is locked immediately. Use a **CLEAR** call to free the API message space. **FREE** can only be used to unlock a message that is already located in a MERVA queue.

Remarks

A pointer to the API internal message buffer is not returned because the message buffer address must be passed for each call that processes the message contents (for example, **ADDMSG** or **ROUTEMSG**).

RPG/ILE Language Example

```
D DSRET           DS
D  RETVAL                 1      2B 0 INZ(0)

C                   CALLB     'CREATE'
C                   PARM                    RETVAL
```

DETACH

This function calls **ENMDetach**.

ENMDetach disconnects the API program from MERVA.

C Definition

```
VOID DETACH( USHORT * RetVal);
```

Parameters

Name	Type	I/O	Comments
RetVal	integer(2)	O	2 bytes, return code from API call. For the values refer to **ENMDetach**.

RPG/ILE Language Example

```
     D DSRET           DS
     D  RETVAL                1      2B 0 INZ(0)

     C                   CALLB     'DETACH'
     C                   PARM                    RETVAL
```

Adding, Changing, and Deleting Messages in MERVA Queues

The following functions allow you to handle messages in MERVA message queues.

ADDMSG

This function calls **ENMAdd**.

ENMAdd adds a message that is created with the call **CREATE** to a MERVA queue. The name of the queue that is passed with the call must belong to the API purpose group. This call has the same functions as **ENMAdd**.

C Definition

```
VOID ADDMSG( QNAME    QueueId,
             UCHAR  * MsgBuf,
             USHORT * MsgLen,
             USHORT * RetVal);
```

Parameters

Name	Type	I/O	Comments
QueueID	string(8+1)	I	Name of queue, null terminated.
MsgBuf	character buffer	I	Text of message to add.
MsgLen	integer(2)	I	Length of message buffer
RetVal	integer(2)	O	2 bytes, return code from API call. **Code** **Meaning** **1001** No internal message space is available. You must call function CREATE or one of the functions which get messages before calling ADDMSG. **Others** For the values refer to **ENMAdd**.

Remarks

If the internal buffer of MERVA Connection/400 is filled with the call **PUTBUFF**, pass the value **0** for **MsgLen**. **ADDMSG** does not refer to the parameter **MsgBuf**. Therefore, you can pass a null pointer.

If the message contains several binary zeros but does not end with the first binary zero, you must specify the message length in the field **MSG_LEN** before **ADDMSG** is called.

RPG/ILE Language Example

```
D DSRET           DS
D  RETVAL                1      2B 0 INZ(0)
D DSLEN           DS
D  LENGTH                1      2B 0
D DSMSG           DS
D  MSGSTR                1    256

C                   Z-ADD     255           LENGTH
C                   CALLB     'ADDMSG'
C                   PARM      'API_IN'      QUEUE             9
C                   PARM                    DSMSG
C                   PARM                    DSLEN
C                   PARM                    RETVAL
```

DELETMSG

This function calls **ENMDelete**.

ENMDelete deletes the currently locked message from the MERVA queue.

C Definition

```
VOID DELETMSG( USHORT * RetVal);
```

Parameters

Name	Type	I/O	Comments
RetVal	integer(2)	O	2 bytes, return code from API call. For the values refer to **ENMDelete**.

Remarks

With **DELETMSG**, you can delete messages that are locked by the calls **FIRSTMSG**, **KEYNEXT**, **KEYREAD**, **LASTMSG**, **NEXTMSG**, or **PREVMSG** from the MERVA queue.

RPG/ILE Language Example

```
D DSRET          DS
D  RETVAL                1      2B 0 INZ(0)

C                  CALLB     'DELETMSG'
C                  PARM                    RETVAL
```

PUTMSG

This function calls **ENMPut**.

ENMPut returns a message to the previous position in the queue from which it was retrieved. The message is then unlocked.

C Definition

```
VOID PUTMSG( UCHAR  * MsgBuf,
             USHORT * MsgLen,
             USHORT * RetVal);
```

Parameters

Name	Type	I/O	Comments
MsgBuf	character buffer	I	Text of message to add.
MsgLen	integer(2)	I	Length of message buffer.
RetVal	integer(2)	O	2 bytes, return code from API call. **Code** **Meaning** **1001** No internal message space is available. You must call function CREATE or one of the functions which get messages before calling PUTMSG. **Others** For the values refer to **ENMRouteAdd**.

Remarks

If the internal buffer of MERVA Connection/400 is filled with the call **PUTBUFF**, pass the value **0** for **MsgLen**. **PUTMSG** does not refer to the parameter **MsgBuf**. Therefore, you can pass a null pointer.

If the message contains several binary zeros but does not end with the first binary zero, you must specify the message length in the field **MSG_LEN** before **PUTMSG** is called.

RPG/ILE Language Example

```
D DSRET           DS
D  RETVAL                1      2B 0 INZ(0)
D DSLEN           DS
D  LENGTH                1      2B 0
D DSMSG           DS
D  MSGSTR                1    256

C                   Z-ADD     255           LENGTH
C                   CALLB     'PUTMSG'
C                   PARM                    DSMSG
C                   PARM                    DSLEN
C                   PARM                    RETVAL
```

ROUTEADD

This function calls **ENMRouteAdd**.

ENMRouteAdd adds a previously created message to the specified queue. The message is then routed immediately according to the routing conditions specified with the MERVA customization.

C Definition

```
VOID ROUTEADD( QNAME    QueueId,
               UCHAR  * MsgBuf,
               USHORT * MsgLen,
               USHORT * RetVal);
```

Parameters

Name	Type	I/O	Comments
QueueId	string(8+1)	I	Name of queue, null terminated.
MsgBuf	character buffer	I	Text of message to add.
MsgLen	integer(2)	I	Length of message buffer.
RetVal	integer(2)	O	2 bytes, return code from API call. **Code** **Meaning** **1001** No internal message space is available. You must call function CREATE or one of the functions which get messages before calling ROUTEADD. **Others** For the values refer to **ENMRouteAdd**.

Remarks

If the internal buffer of MERVA Connection/400 is filled with the call **PUTBUFF**, pass the value **0** for **MsgLen**. **ROUTEADD** does not refer to the parameter **MsgBuf**. Therefore, you can pass a null pointer.

If the message contains several binary zeros but does not end with the first binary zero, you must specify the message length in the field **MSG_LEN** before **ROUTEADD** is called.

RPG/ILE Language Example

```
D DSRET          DS
D  RETVAL                1      2B 0 INZ(0)
D DSLEN          DS
D  LENGTH                1      2B 0
D DSMSG          DS
D  MSGSTR                1    256

C                  Z-ADD     255           LENGTH
C                  CALLB     'ROUTEADD'
C                  PARM      'API_IN'      QUEUE            9
C                  PARM                    DSMSG
C                  PARM                    DSLEN
C                  PARM                    RETVAL
```

ROUTEPUT

This function calls **ENMRoutePut**.

ENMRoutePut returns a previously retrieved message to the queue from which it was retrieved. The message is then routed immediately according to the routing conditions specified with the MERVA customization.

C Definition

```
VOID ROUTEPUT( UCHAR  * MsgBuf,
               USHORT * MsgLen,
               USHORT * RetVal);
```

Parameters

Name	Type	I/O	Comments
MsgBuf	character buffer	I	Text of message to add.
MsgLen	integer(2)	I	Length of message buffer.
RetVal	integer(2)	O	2 bytes, return code from API call. **Code** **Meaning** **1001** No internal message space is available. You must call function CREATE or one of the functions which get messages before calling ROUTEPUT. **Others** For the values refer to **ENMRouteAdd**.

Remarks

If the internal buffer of MERVA Connection/400 is filled with the call **PUTBUFF**, pass the value **0** for **MsgLen**. **ROUTEPUT** does not refer to the parameter **MsgBuf**. Therefore, you can pass a null pointer.

If the message contains several binary zeros but does not end with the first binary zero, you must specify the message length in the field **MSG_LEN** before **ROUTEPUT** is called.

RPG/ILE Language Example

```
D DSRET           DS
D  RETVAL                 1      2B 0 INZ(0)
D DSLEN           DS
D  LENGTH                 1      2B 0
D DSMSG           DS
D  MSGSTR                 1    256

C                   Z-ADD     255           LENGTH
C                   CALLB     'ROUTEPUT'
C                   PARM                    DSMSG
C                   PARM                    DSLEN
C                   PARM                    RETVAL
```

Reading, Getting, and Releasing Messages in MERVA Queues

Use the following functions to lock and unlock messages in MERVA queues.

FIRSTMSG

This function calls **ENMFirstEntry**.

ENMFirstEntry gets the oldest message from a MERVA API queue.

C Definition

```
VOID FIRSTMSG( QNAME    QueueId,
               SWITCH * Lock,
               UCHAR  * MsgBuf,
               USHORT * MsgLen,
               USHORT * RetVal);
```

Parameters

Name	Type	I/O	Comments
QueueID	string(8+1)	I	Name of queue from which to get the message, null terminated.
Lock	integer(2)	I	Value 1 = Lock message. Value 0 = Do not lock message.
MsgBuf	character buffer	O	Message retrieved.
MsgLen	integer(2)	I,O	Input: Length of buffer. Output:Length of message.
RetVal	integer(2)	O	2 bytes, return code from API call. For the values refer to **ENMFirstEntry**.

Processing

With the calls **FIRSTMSG**, **NEXTMSG**, **PREVMSG**, and **LASTMSG**, the system sets a position pointer for each queue. An application can then switch queues and resume at the point from which it switched the queue.

If the message is read with the lock value set to **1**, other programs cannot change the message. **FREEMSG**, **ROUTEADD**, and **ROUTEPUT** unlock the message.

Remarks

If the message exceeds the maximum length that is specified with the parameter **MsgLen**, only the maximum length of bytes is copied to **MsgBuf**. The complete length of the message, however, is returned in **MsgLen**. The API program can then check whether the message is retrieved completely. You can read the complete message with one or more calls of the function **GETREST**. For details refer to “Handling the Internal Message Buffer of MERVA Connection/400” on page 52.

RPG/ILE Language Example

```
D DSRET           DS
D  RETVAL                      1      2B 0 INZ(0)
D DSLEN           DS
D  LENGTH                      1      2B 0
D DSMSG           DS
D  MSGSTR                      1    256
D DSLOCK          DS
D  LOCK                        1      2B 0
D DSMRN           DS
D  MRNBUF                      1     20

C                   Z-ADD     255           LENGTH
```

```
C                   CALLB     'FIRSTMSG'
C                   PARM      'API_OUT'     QUEUE            9
C                   PARM                    LOCK
C                   PARM                    DSMSG
C                   PARM                    DSLEN
C                   PARM                    RETVAL
```

FREEMSG

This function calls **ENMFree**.

ENMFree unlocks a previously locked message.

C Definition

```
VOID FREEMSG( USHORT * RetVal);
```

Parameters

Name	Type	I/O	Comments
RetVal	integer(2)	O	2 bytes, return code from API call. For the values refer to **ENMFree**.

Remarks

With **FREEMSG**, you can release messages that are previously locked by the calls **FIRSTMSG**, **KEYNEXT**, **KEYREAD**, **LASTMSG**, **NEXTMSG**, or **PREVMSG**.

RPG/ILE Language Example

```
     D DSRET           DS
     D  RETVAL                 1      2B 0 INZ(0)

     C                   CALLB     'FREEMSG'
     C                   PARM                    RETVAL
```

KEYNEXT

This function calls **ENMKeyNext**.

Use this function if **KEYREAD** has returned a message that matches the specified criteria, such as queue name and key (MRN or ISN). **KEYNEXT** returns other messages that match the same criteria.

C Definition

```
VOID KEYNEXT( SWITCH * Lock,
              UCHAR  * MsgBuf,
              USHORT * MsgLen,
              USHORT * RetVal);
```

Parameters

Name	Type	I/O	Comments
Lock	integer(2)	I	Value 1 = Lock message. Value 0 = Do not lock message.
MsgBuf	character buffer	O	Message retrieved.
MsgLen	integer(2)	I,O	Input: Length of buffer. Output: Length of message.
RetVal	integer(2)	O	2 bytes, return code from API call. For the values refer to **ENMKeyNext**.

Processing

If the message is read with the lock value set to **1**, other programs cannot change the message. **FREEMSG**, **ROUTEADD**, and **ROUTEPUT** unlock the message.

Remarks

If the message exceeds the maximum length that is specified with the parameter **MsgLen**, only the maximum length of bytes is copied to **MsgBuf**. The complete length of the message, however, is returned in **MsgLen**. The API program can then check whether the message is retrieved completely. You can read the complete message with one or more calls of the function **GETREST**. For details refer to "Handling the Internal Message Buffer of MERVA Connection/400" on page 52.

RPG/ILE Language Example

```
     D DSRET           DS
     D  RETVAL                 1      2B 0 INZ(0)
     D DSLEN           DS
     D  LENGTH                 1      2B 0
     D DSMSG           DS
     D  MSGSTR                 1    256
     D DSLOCK          DS
     D  LOCK                   1      2B 0

     C                   Z-ADD     255           LENGTH
     C                   CALLB     'KEYNEXT'
     C                   PARM                    DSLOCK
     C                   PARM                    DSMSG
     C                   PARM                    DSLEN
     C                   PARM                    RETVAL
```

KEYREAD

This function calls **ENMKeyRead**.

ENMKeyRead searches for the first message with the specified key in the specified queue.

C Definition

```
VOID KEYREAD( QNAME     QueueId,
              KEYTYPE * KeyType,
              KEY     * Key,
              SWITCH  * Lock,
              UCHAR   * MsgBuf,
              USHORT  * MsgLen,
              USHORT  * RetVal);
```

Parameters

Name	Type	I/O	Comments
QueueID	string(8+1)	I	Name of queue, null terminated.
KeyType	integer(2)	I	Value 0 = ISN. Value 1 = MRN.
Key	string(6+1 or 16+1)	I	Value of key, length of buffer must be 6+1 for ISN and 16+1 for MRN. It is null terminated by the called function.
Lock	integer(2)	I	Value 1 = Lock message. Value 0 = Do not lock message.
MsgBuf	character buffer	O	Message retrieved.
MsgLen	integer(2)	I,O	Input: Length of buffer. Output: Length of message.
RetVal	integer(2)	O	2 bytes, return code from API call. For the values refer to **ENMKeyRead**.

Processing

If the message is read with the lock value set to **1**, other programs cannot change the message. **FREEMSG**, **ROUTEADD**, and **ROUTEPUT** unlock the message. If the function is called more than once with the same key and queue name, the same message is returned. To retrieve further messages that match the specified criteria, call **KEYNEXT**.

Remarks

If the message exceeds the maximum length that is specified with the parameter **MsgLen**, only the maximum length of bytes is copied to **MsgBuf**. The complete length of the message, however, is returned in **MsgLen**. The API program can then check whether the message is retrieved completely. You can read the complete message with one or more calls of the function **GETREST**. For details refer to "Handling the Internal Message Buffer of MERVA Connection/400" on page 52.

RPG/ILE Language Example

```
D DSRET           DS
D  RETVAL                1      2B 0 INZ(0)
D DSLEN           DS
D  LENGTH                1      2B 0
D DSMSG           DS
D  MSGSTR                1    256
D DSKEY           DS
```

```
D  KEYSTR                 1     20    INZ('R03D501100000178')
D DSKTYP          DS
D  KEYTYP                 1      2B 0

C                   RESETDSKEY
C                   Z-ADD     1             KEYTYP
C                   Z-ADD     0             LOCK
C                   Z-ADD     255           LENGTH
C                   CALLB     'KEYREAD'
C                   PARM      'API_OUT'     QUEUE             9
C                   PARM                    DSKTYP
C                   PARM                    DSKEY
C                   PARM                    DSLOCK
C                   PARM                    DSMSG
C                   PARM                    DSLEN
C                   PARM                    RETVAL
```

LASTMSG

This function calls **ENMLastEntry**.

ENMLastEntry retrieves the message that was most recently added to the specified queue.

C Definition

```
VOID LASTMSG( QNAME    QueueId,
              SWITCH * Lock,
              UCHAR  * MsgBuf,
              USHORT * MsgLen,
              USHORT * RetVal);
```

Parameters

Name	Type	I/O	Comments
QueueID	string(8+1)	I	Name of queue, null terminated.
Lock	integer(2)	I	Value 1 = Lock message. Value 0 = Do not lock message.
MsgBuf	character buffer	O	Message retrieved
MsgLen	integer(2)	I,O	Input: Length of buffer. Output: Length of message.
RetVal	integer(2)	O	2 bytes, return code from API call. For the values refer to **ENMLastEntry**.

Processing

With the calls **FIRSTMSG**, **NEXTMSG**, **PREVMSG**, and **LASTMSG**, the system sets a position pointer for each queue. An application can then switch queues and resume at the point from which it switched the queue.

If the message is read with the lock value set to **1**, other programs cannot change the message. **FREEMSG**, **ROUTEADD**, and **ROUTEPUT** unlock the message.

Remarks

If the message exceeds the maximum length that is specified with the parameter **MsgLen**, only the maximum length of bytes is copied to **MsgBuf**. The complete length of the message, however, is returned in **MsgLen**. The API program can then check whether the message is retrieved completely. You can read the complete message with one or more calls of the function **GETREST**. For details refer to “Handling the Internal Message Buffer of MERVA Connection/400” on page 52.

RPG/ILE Language Example

```
D DSRET           DS
D  RETVAL                 1      2B 0 INZ(0)
D DSLEN           DS
D  LENGTH                 1      2B 0
D DSMSG           DS
D  MSGSTR                 1    256
D DSLOCK          DS
D  LOCK                   1      2B 0
D DSMRN           DS
D  MRNBUF                 1     20

C                   Z-ADD     255           LENGTH
C                   CALLB     'LASTMSG'
```

```
C                   PARM      'API_OUT'     QUEUE            9
C                   PARM                    LOCK
C                   PARM                    DSMSG
C                   PARM                    DSLEN
C                   PARM                    RETVAL
```

NEXTMSG

This function calls **ENMNextEntry**.

ENMNextEntry returns the message that is next to the current position in the message queue sorted by time. If the application accesses the queue for the first time, the function returns the oldest message.

C Definition

```
VOID NEXTMSG( QNAME    QueueId,
              SWITCH * Lock,
              UCHAR  * MsgBuf,
              USHORT * MsgLen,
              USHORT * RetVal);
```

Parameters

Name	Type	I/O	Comments
QueueID	string(8+1)	I	Name of queue, null terminated.
Lock	integer(2)	I	Value 1 = Lock message. Value 0 = Do not lock message.
MsgBuf	character buffer	O	Message retrieved
MsgLen	integer(2)	I,O	Input: Length of buffer. Output: Length of message.
RetVal	integer(2)	O	2 bytes, return code from API call. For the values refer to **ENMNextEntry**.

Processing

With the calls **FIRSTMSG**, **NEXTMSG**, **PREVMSG**, and **LASTMSG**, the system sets a position pointer for each queue. An application can then switch queues and resume at the point from which it switched the queue.

If the message is read with the lock value set to **1**, other programs cannot change the message. **FREEMSG**, **ROUTEADD**, and **ROUTEPUT** unlock the message.

Remarks

If the message exceeds the maximum length that is specified with the parameter **MsgLen**, only the maximum length of bytes is copied to **MsgBuf**. The complete length of the message, however, is returned in **MsgLen**. The API program can then check whether the message is retrieved completely. You can read the complete message with one or more calls of the function **GETREST**. For details refer to “Handling the Internal Message Buffer of MERVA Connection/400” on page 52.

RPG/ILE Language Example

```
     D DSRET           DS
     D  RETVAL                 1      2B 0 INZ(0)
     D DSLEN           DS
     D  LENGTH                 1      2B 0
     D DSMSG           DS
     D  MSGSTR                 1    256
     D DSLOCK          DS
     D  LOCK                   1      2B 0
     D DSMRN           DS
     D  MRNBUF                 1     20
```

```
C                   Z-ADD     255           LENGTH
C                   CALLB     'NEXTMSG'
C                   PARM      'API_OUT'     QUEUE              9
C                   PARM                    LOCK
C                   PARM                    DSMSG
C                   PARM                    DSLEN
C                   PARM                    RETVAL
```

PREVMSG

This function calls **ENMPreviousEntry**.

ENMPreviousEntry returns the message that precedes the current position in the message queue sorted by time. If the application accesses the queue for the first time, the function returns the oldest message.

C Definition

```
VOID PREVMSG( QNAME    QueueId,
              SWITCH * Lock,
              UCHAR  * MsgBuf,
              USHORT * MsgLen,
              USHORT * RetVal);
```

Parameters

Name	Type	I/O	Comments
QueueID	string(8+1)	I	Name of queue, null terminated.
Lock	integer(2)	I	Value 1 = Lock message. Value 0 = Do not lock message.
MsgBuf	character buffer	O	Message retrieved
MsgLen	integer(2)	I,O	Input: Length of buffer. Output: Length of message.
RetVal	integer(2)	O	2 bytes, return code from API call. For the values refer to **ENMPreviousEntry**.

Processing

With the calls **FIRSTMSG**, **NEXTMSG**, **PREVMSG**, and **LASTMSG**, the system gets a position pointer for each queue. An application can then switch queues and resume at the point from which it switched the queue.

If the message is read with the lock value set to **1**, other programs cannot change the message. **FREEMSG**, **ROUTEADD**, and **ROUTEPUT** unlock the message.

Remarks

If the message exceeds the maximum length that is specified with the parameter **MsgLen**, only the maximum length of bytes is copied to **MsgBuf**. The complete length of the message, however, is returned in **MsgLen**. The API program can then check whether the message is retrieved completely. You can read the complete message with one or more calls of the function **GETREST**. For details refer to "Handling the Internal Message Buffer of MERVA Connection/400" on page 52.

RPG/ILE Language Example

```
D DSRET           DS
D  RETVAL                 1      2B 0 INZ(0)
D DSLEN           DS
D  LENGTH                 1      2B 0
D DSMSG           DS
D  MSGSTR                 1    256
D DSLOCK          DS
D  LOCK                   1      2B 0
D DSMRN           DS
D  MRNBUF                 1     20
```

```
C                   Z-ADD     255           LENGTH
C                   CALLB     'PREVMSG'
C                   PARM      'API_OUT'     QUEUE              9
C                   PARM                    LOCK
C                   PARM                    DSMSG
C                   PARM                    DSLEN
C                   PARM                    RETVAL
```

Handling the Internal Message Buffer of MERVA Connection/400

Messages in MERVA queues can be up to 28000 bytes long. In RPG/ILE, you can specify a maximum buffer size of 9999 characters. Therefore, an RPG/ILE API program cannot handle larger messages with the usual API calls to retrieve messages from and write messages to MERVA queues. To handle large messages, MERVA Connection/400 provides the following calls to write and read the internal message buffer of MERVA in more than one step.

PUTBUFF

Writes data to the internal message buffer of MERVA Connection/400.

With this call you can handle large messages in spite of the limited data buffer size of 9999 bytes in RPG/ILE.

C Definition

```
VOID PUTBUFF( UCHAR  * MsgBuf,
              USHORT * MsgLen,
              USHORT * RetVal);
```

Parameters

Name	Type	I/O	Comments
MsgBuf	character buffer	I	Data buffer to write.
MsgLen	integer(2)	I	Length of data to write on input, length of data written on output.
RetVal	integer(2)	O	2 bytes, return code from API call. **Code** **Meaning** **1001** No internal message space is available. You must call function CREATE or one of the functions which get messages before calling PUTBUFF. **1002** Message has been truncated, internal message space too small.

Remarks

When the function **PUTBUFF** is called for the first time, the internal buffer is written. It starts with offset **0**. The offset is held internally in MERVA Connection/400. After the first call, the value of the internal offset is identical to the buffer length passed. Each time, you call **PUTBUFF**, the passed buffer is appended at the position that is specified with the internal offset.

When **CREATE**, **FIRSTMSG**, **NEXTMSG**, **LASTMSG**, **PREVMSG**, **KEYREAD**, or **KEYNEXT** is called, the internal offset is reset to **0**.

RPG/ILE Language Example

```
     D DSRET           DS
     D  RETVAL                 1      2B 0 INZ(0)
     D DSLEN           DS
     D  LEN                    1      2B 0 INZ(255)
     D  LENNUL                 1      2B 0 INZ(0)
     D DSMSG           DS
     D  BUF1                   1    255
     D  BUF2                 256    510
     D  BUF3                 511    765

     C                   RESETDSLEN
     C                   CALLB     'PUTBUFF'
     C                   PARM                    BUF1
     C                   PARM                    LEN
```

```
C                    PARM                    RETVAL
C                    CALLB     'PUTBUFF'
C                    PARM                    BUF2
C                    PARM                    LEN
C                    PARM                    RETVAL
C                    CALLB     'PUTBUFF'
C                    PARM                    BUF3
C                    PARM                    LEN
C                    PARM                    RETVAL
C                    CALLB     'ADDMSG'
C                    PARM      'API_IN'      QUEUE              9
C                    PARM                    DSMSG
C                    PARM                    LENNUL
C                    PARM                    RETVAL
```

GETREST

Reads data from the internal message buffer of MERVA Connection/400.

With this call you can handle large messages in spite of the limited data buffer size of 9999 bytes in RPG/ILE.

C Definition

```
VOID GETREST( UCHAR  * MsgBuf,
              USHORT * MsgLen,
              USHORT * RetVal);
```

Parameters

Name	Type	I/O	Comments
MsgBuf	character buffer	I	Data buffer to write retrieved data to.
MsgLen	integer(2)	I,O	Length of buffer on input, length of remaining message part in internal message space when entering the function on output.
RetVal	integer(2)	O	2 bytes, return code from API call. **Code** **Meaning** **1001** No internal message space is available. You must call one of the functions which get messages before calling GETREST.

Remarks

If **FIRSTMSG**, **NEXTMSG**, **LASTMSG**, **PREVMSG**, **KEYREAD**, or **KEYNEXT** returns a message length that exceeds the buffer that is specified in the call, only the first part of the retrieved message is written to the buffer. You can read the rest of the message data with one or more calls of **GETREST**. An internal offset address is stored in MERVA Connection/400.

After one of the previously listed calls, it is set to the buffer length already read. After each call of GETREST, it is incremented by the value of MsgLen specified in the call. GETREST returns the remaining number of bytes in MsgLen.

RPG/ILE Language Example

```
     D DSRET           DS
     D  RETVAL                 1      2B 0 INZ(0)
     D DSLEN           DS
     D  LEN                    1      2B 0 INZ(255)
     D  LENNUL                 1      2B 0 INZ(0)
     D DSMSG           DS
     D  BUF1                   1    255
     D  BUF2                 256    510
     D  BUF3                 511    765

     C                   RESETDSLEN
     C                   CALLB     'FIRSTMSG'
     C                   PARM      'API_OUT'     QUEUE            9
     C                   PARM                    LOCK
     C                   PARM                    BUF1
     C                   PARM                    LEN
```

```
C                   PARM                    RETVAL
 * assumed that the value 765 has been returned in DSLEN
C                   CALLB     'GETREST'
C                   PARM                    BUF2
C                   PARM                    LEN
C                   PARM                    RETVAL
C                   CALLB     'GETREST'
C                   PARM                    BUF3
C                   PARM                    LEN
C                   PARM                    RETVAL
```

Handling Single Message Fields

The following functions allow you to work with single message fields.

READFLD

This function calls **ENMReadField**.

ENMReadField returns the contents of a field associated with the previously retrieved message or generated by a **CREATE** call.

C Definition

```
VOID READFLD( FIELDTYPE * FieldType,
              PFIELD      Field,
              USHORT    * RetVal);
```

Parameters

Name	Type	I/O	Comments
FieldType	integer(2)	I	Specifies the field type.
Field	integer or char	I	Type depends on specified field.
RetVal	integer(2)	O	2 bytes, return code from API call. For the values refer to **ENMReadField**.

RPG/ILE Language Example

```
     D DSRET           DS
     D  RETVAL                 1      2B 0 INZ(0)
     D DSMRN           DS
     D  MRNBUF                 1     17
     D DSNETW          DS
     D  NETWID                 1      2B 0
     D DSFLTY          DS
     D  FLDTYP                 1      2B 0

     C                   Z-ADD     0             FLDTYP
     C                   CALLB     'READFLD'
     C                   PARM                    FLDTY
     C                   PARM                    DSMRN
     C                   PARM                    RETVAL
     C                   Z-ADD2                  FLDTYP
     C                   CALLB     'READFLD'
     C                   PARM                    FLDTY
     C                   PARM                    DSNETW
```

The following example shows you a sample definition of a telex header structure.

```
     D DSTXHD          DS
     D  KEYCAL                 1      2    INZ('N ')
     D  KEYRC                  3      4    INZ('  ')
     D  KEYVAL                 5     20    INZ('  ')
     D  KEYCM1                21     57    INZ('  ')
     D  KEYCM2                58     93    INZ('  ')
     D  SNADR0                94    129    INZ('SENDERBANK ')
     D  SNADR1               130    165    INZ('  ')
     D  SNADR2               166    201    INZ('  ')
     D  SNADR3               202    237    INZ('  ')
     D  DATE                 238    244    INZ('  ')
     D  TOID                 245    256    INZ('  ')
     D  RCADR0               257    292    INZ('RECEIVERBANK ')
     D  RCADR1               293    328    INZ('  ')
     D  RCADR2               329    364    INZ('  ')
     D  RCADR3               365    400    INZ('  ')
```

```
D  LINE                  401    403    INZ('  ')
D  DLUP1                 404    424    INZ('049-1234-54321 ')
D  ANSBK1                425    445    INZ('  ')
D  DLUP2                 446    466    INZ('  ')
D  ANSBK2                467    487    INZ('  ')
D  TYPE                  488    489    INZ('N ')
D  TIMTIM                490    494    INZ('  ')
D  TIMDAT                495    501    INZ('  ')
D  REFTXT                502    518    INZ('  ')
D  NOTE                  519    583    INZ('  ')
```

WRITFLD

This function calls **ENMWriteField**.

ENMWriteField updates the contents of a field associated with the previously retrieved message or generated by a **CREATE** call.

C Definition

```
VOID WRITFLD( FIELDTYPE * FieldType,
              PFIELD      Field,
              USHORT    * RetVal);
```

Parameters

Name	Type	I/O	Comments
FieldType	integer(2)	I	Specifies field type.
Field	integer or char	I	Type depends on specified field.
RetVal	integer(2)	O	2 bytes, return code from API call. For the values refer to **ENMWriteField**.

RPG/ILE Language Example

```
     D DSRET           DS
     D  RETVAL                 1      2B 0 INZ(0)
     D DSACK           DS
     D  ACK1                   1      4
     D  ACK2                   5    128
     D DSNETW          DS
     D  NETWID                 1      2B 0
     D DSFLTY          DS
     D  FLDTYP                 1      2B 0

      * ------------- Write Network ID for SWIFT
     C                   Z-ADD     2             FLDTYP
     C                   Z-ADD     2             NETWID
     C                   CALLB     'WRITFLD'
     C                   PARM                    DSFLTY
     C                   PARM                    DSNETW
     C                   PARM                    RETVAL
      * ------------- Write "ACK" to MSGACK field
     C                   Z-ADD     5             FLDTYP
     C                   MOVE      'ACK '        ACK1
     C                   CALLB     'WRITFLD'
     C                   PARM                    DSFLTY
     C                   PARM                    DSACK
     C                   PARM                    RETVAL
```

Miscellaneous Calls

The following functions allow you to handle miscellaneous calls.

QUERYQU

This function calls **ENMQueryQueue**.

ENMQueryQueue returns the number of messages in the specified queue at the time of the call.

C Definition

```
VOID QUERYQU( UCHAR  * QueueId,
              USHORT * MessageCount,
              USHORT * RetVal);
```

Parameters

Name	Type	I/O	Comments
QueueID	string(8+1)	I	Name of queue, null terminated.
MsgCount	integer(2)	O	Returns the number of messages in the specified queue.
RetVal	integer(2)	O	2 bytes, return code from API call. For the values refer to **ENMQueryQueue**.

RPG/ILE Language Example

```
     D DSRET           DS
     D  RETVAL                 1      2B 0 INZ(0)
     D DSCNT           DS
     D  COUNT                  1      2B 0

     C                   CALLB     'QUERYQU'
     C                   PARM      'API_OUT'     QUEUE            9
     C                   PARM                    DSCNT
     C                   PARM                    RETVAL
```

TRACE

This function calls **ENMTrace**.

ENMTrace turns the MERVA API trace on or off. When the API trace is switched on, API calls are written to the diagnosis logs of the Windows NT system. In addition, the parameters (for buffers, only the first characters) are written to the diagnosis log.

C Definition

```
VOID TRACE( SWITCH * Status,
            USHORT * RetVal);
```

Parameters

Name	Type	I/O	Comments
Status	integer(2)	I	Value 1 = ON. Value 0 = OFF.
RetVal	integer(2)	O	2 bytes, return code from API call. For the values refer to **ENMTrace**.

RPG/ILE Language Example

```
D DSRET           DS
D  RETVAL                1      2B 0 INZ(0)
D DSSWIT          DS
D  ONOFF                 1      2B 0 INZ(1)

C                   CALLB     'TRACE'
C                   PARM                    DSSWIT
C                   PARM                    RETVAL
```

WHEREIS

This function calls **ENMWhereIs**.

ENMWhereIs returns the number of the purpose group in which the message with a specified key is located. An application can, for example, check whether a message is already sent to the S.W.I.F.T. network or waits to be sent.

C Definition

```
VOID WHEREIS( KEYTYPE * KeyType,
              KEY     * Key,
              PGROUP    Group,
              USHORT  * RetVal);
```

Parameters

Name	Type	I/O	Comments
KeyType	integer (2)	I	Value 0 = ISN, value 1 = MRN
Key	string(6+1 or 16+1)	I	Value of key, length of buffer **must** be 6+1 for ISN and 16+1 for MRN; it is null terminated by the called function.
Group	integer(2)	O	The meaning of the values is listed in the *MERVA USE & Branch for Windows NT Application Programming Guide*.
RetVal	integer(2)	O	2 bytes, return code from API call. For the values refer to **ENMWhereIs**.

RPG/ILE Language Example

```
D DSRET           DS
D  RETVAL                 1      2B 0 INZ(0)
D DSKEY           DS
D  KEYSTR                 1     17
D DSKTYP          DS
D  KEYTYP                 1      2B 0
D DSGRUP          DS
D  GROUP                  1      2B 0

C                   CALLB     'WHEREIS'
C                   PARM                    DSKTYP
C                   PARM                    DSKEY
C                   PARM                    DSGRUP
C                   PARM                    RETVAL
```

WRTTRACE

This function calls **ENMWriteTrace**.

ENMWriteTrace writes a string that is defined by the API program to the diagnosis log in MERVA.

C Definition

```
VOID WRTTRACE( UCHAR  * Userinfo,
               USHORT * RetVal);
```

Parameters

Name	Type	I/O	Comments
UserInfo	string(240+1)	I	Text to be written to the diagnosis log in MERVA.
RetVal	integer(2)	O	2 bytes, return code from API call. For the values refer to **ENMWriteTrace**.

RPG/ILE Language Example

```
     D DSRET           DS
     D  RETVAL                 1      2B 0 INZ(0)
     D DSTRAC          DS
     D  TRACE                  1    241    INZ('TRACE TEXT ...')

     C                   RESETTRACE
     C                   CALLB     'WRTTRACE'
     C                   PARM                    DSTRAC
     C                   PARM                    RETVAL
```

Functions Triggered by MERVA Alarms

The following functions handle semaphores.

CLRSEM

This function calls **ENMClearSem**.

ENMClearSem clears a semaphore unconditionally. Processes that are blocked on the semaphore, are restarted.

C Definition

```
VOID CLRSEM( ULONG  * SemHandle,
             USHORT * RetVal);
```

Parameters

Name	Type	I/O	Comments
SemHandle	integer(4)	I	4 bytes, generated by CRTSEM or OPNSEM.
RetVal	integer(2)	O	2 bytes, return code from API call. For the values refer to **ENMClearSem**.

Remarks

For return values, see the description of the call **ENMClearSem**.

RPG/ILE Language Example

```
     D DSRET           DS
     D  RETVAL                 1      2B 0 INZ(0)
     D DSSH1           DS
     D  SH1                    1      4B 0

     C                   CALLB     'CLRSEM'
     C                   PARM                    DSSH1
     C                   PARM                    RETVAL
```

CLSSEM

This function calls **ENMCloseSem**.

ENMCloseSem closes a handle of a semaphore, obtained with a **CRTSEM** or **OPNSEM** call.

C Definition

```
VOID CLSSEM( ULONG  * SemHandle,
             USHORT * RetVal);
```

Parameters

Name	Type	I/O	Comments
SemHandle	integer(4)	I	4 bytes, generated by CRTSEM or OPNSEM.
RetVal	integer(2)	O	2 bytes, return code from API call. For the values refer to **ENMCloseSem**.

Remarks

For return values, see the description of the call **ENMCloseSem**.

RPG/ILE Language Example

```
D DSRET           DS
D  RETVAL                1      2B 0 INZ(0)
D DSSH1           DS
D  SH1                   1      4B 0

C                   CALLB     'CLSSEM'
C                   PARM                    DSSH1
C                   PARM                    RETVAL
```

CRTSEM

This function calls **ENMCreateSem**.

ENMCreateSem creates a semaphore with which API programs can synchronize their access to resources or wait for MERVA alarms.

C Definition

```
VOID CRTSEM( ULONG  * SemHandle,
             UCHAR  * SemName,
             USHORT * RetVal);
```

Parameters

Name	Type	I/O	Comments
SemHandle	integer(4)	O	Length 4 bytes.
SemName	string(80+1)	I	This is a null-terminated string containing the name with which the semaphore is to be created.
RetVal	integer(2)	O	2 bytes, return code from API call. For the values refer to **ENMCreateSem**.

Remarks

For return values, see the description of the call **ENMCreateSem**.

RPG/ILE Language Example

```
D DSRET           DS
D  RETVAL                 1      2B 0 INZ(0)
D DSSAMP          DS
D  SMSAMP                 1     12    INZ('\SEM\SAMPLE2')
D  T3                    13     14B 0 INZ(0)
D DSSH2           DS
D  SH2                    1      4B 0

C                   RESETDSSAMP
C                   CALLB     'CRTSEM'
C                   PARM                    DSSH2
C                   PARM                    DSSAMP
C                   PARM                    RETVAL
```

OPNSEM

This function calls **ENMOpenSem**.

ENMOpenSem opens an existing semaphore that was created by another process with **CRTSEM**. The other process can run on the Windows NT system.

C Definition

```
VOID OPNSEM( ULONG  * SemHandle,
             UCHAR  * SemName,
             USHORT * RetVal);
```

Parameters

Name	Type	I/O	Comments
SemHandle	integer(4)	O	Length 4 bytes.
SemName	string(80+1)	I	This is a null-terminated string containing the name with which the semaphore was created.
RetVal	integer(2)	O	2 bytes, return code from API call. For the values refer to **ENMOpenSem**.

Remarks

For return values, see the description of the call **ENMOpenSem**.

RPG/ILE Language Example

```
D DSRET           DS
D  RETVAL                      1      2B 0 INZ(0)
D DSSAMP          DS
D  SMSAMP                      1     12    INZ('\SEM\SAMPLE2')
D  T3                         13     14B 0 INZ(0)
D DSSH2           DS
D  SH2                         1      4B 0

C                   RESETDSSAMP
C                   CALLB     'OPNSEM'
C                   PARM                    DSSH2
C                   PARM                    DSSAMP
C                   PARM                    RETVAL
```

SETSEM

This function calls **ENMSetSem**.

ENMSetSem sets a semaphore unconditionally, regardless of whether the semaphore is already set. In MERVA, the semaphore can be cleared by a MERVA alarm.

C Definition

```
VOID SETSEM( ULONG  * SemHandle,
             USHORT * RetVal);
```

Parameters

Name	Type	I/O	Comments
SemHandle	integer(4)	I	4 bytes, generated by CRTSEM or OPNSEM.
RetVal	integer(2)	O	2 bytes, return code from API call. For the values refer to **ENMSetSem**.

Remarks

For return values, see the description of the call **ENMSetSem**.

RPG/ILE Language Example

```
     D DSRET           DS
     D  RETVAL                 1      2B 0 INZ(0)
     D DSSH1           DS
     D  SH1                    1      4B 0

     C                   CALLB     'SETSEM'
     C                   PARM                    DSSH1
     C                   PARM                    RETVAL
```

WTSEMLST

This function calls **ENMWaitSemList**.

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

C Definition

```
VOID WTSEMLST( USHORT * Index,
               ULONG  * Timeout,
               USHORT * RetVal,
               ULONG  * SemHandle,
                               ...,
               (ULONG)   0);[1]
```

Parameters

Name	Type	I/O	Comments
Index	integer(2)	O	Index of semaphore that has been cleared. [2]
Timeout	integer(4)	I	Action taken when none of the semaphores satisfies the wait request (see table below).
RetVal	integer(2)	O	2 bytes, return code from API call. For the values refer to **ENMWaitSemList**.
Semhandle	integer(4)	I	Up to 16 SemHandles, generated by CRTSEM or OPNSEM.
0	integer(4)	I	Terminates list of semaphore handles.

Table 2. Possible Values for Parameter Timeouts and Their Meaning

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

Remarks

For return values, see the description of the call **ENMWaitSemList**.

RPG/ILE Language Example

```
D DSRET           DS
D  RETVAL                1      2B 0 INZ(0)
D DSSH1           DS
D  SH1                   1      4B 0
```

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

2. If the first semaphore of the input list is cleared, the index value is 0. For succeeding semaphores, the value is 1 to 15.

```
D DSSH2           DS
D  SH2                   1      4B 0
D DSSH3           DS
D  SH3                   1      4B 0
D DSTIM           DS
D  TIMOUT                1      4B 0 INZ(-1)
D DSIDX           DS
D  INDEX                 1      2B 0 INZ(1)

 *
 * ... create semhandles SH1/2 with OPNSEM or CRTSEM
 *

C                   Z-ADD     0             SH3
C                   Z-ADD     -1            TIMOUT
C                   CALLB     'WTSEMLST'
C                   PARM                    DSIDX
C                   PARM                    DSTIM
C                   PARM                    RETVAL
C                   PARM                    DSSH1
C                   PARM                    DSSH2
C                   PARM                    DSSH3
```

Error Handling

The following function handles errors.

REASON

This function calls **ENMGetReason**.

ENMGetReason returns a reason code for an internal error in MERVA Connection/400.

For a description of the values refer to the function **ENMGetReason** in “Handling Errors” on page 15.

C Definition

```
VOID REASON( USHORT * RetVal);
```

Parameters

Name	Type	I/O	Comments
RetVal	integer(2)	O	2 bytes, return code from API call.

Chapter 6. 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/400 supports encryption and authentication.

Encryption of Transferred Information

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 Information

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/400.
- A user exit is a routine that is written by you and called by MERVA Connection/400. 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 you an example of an API function that is 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

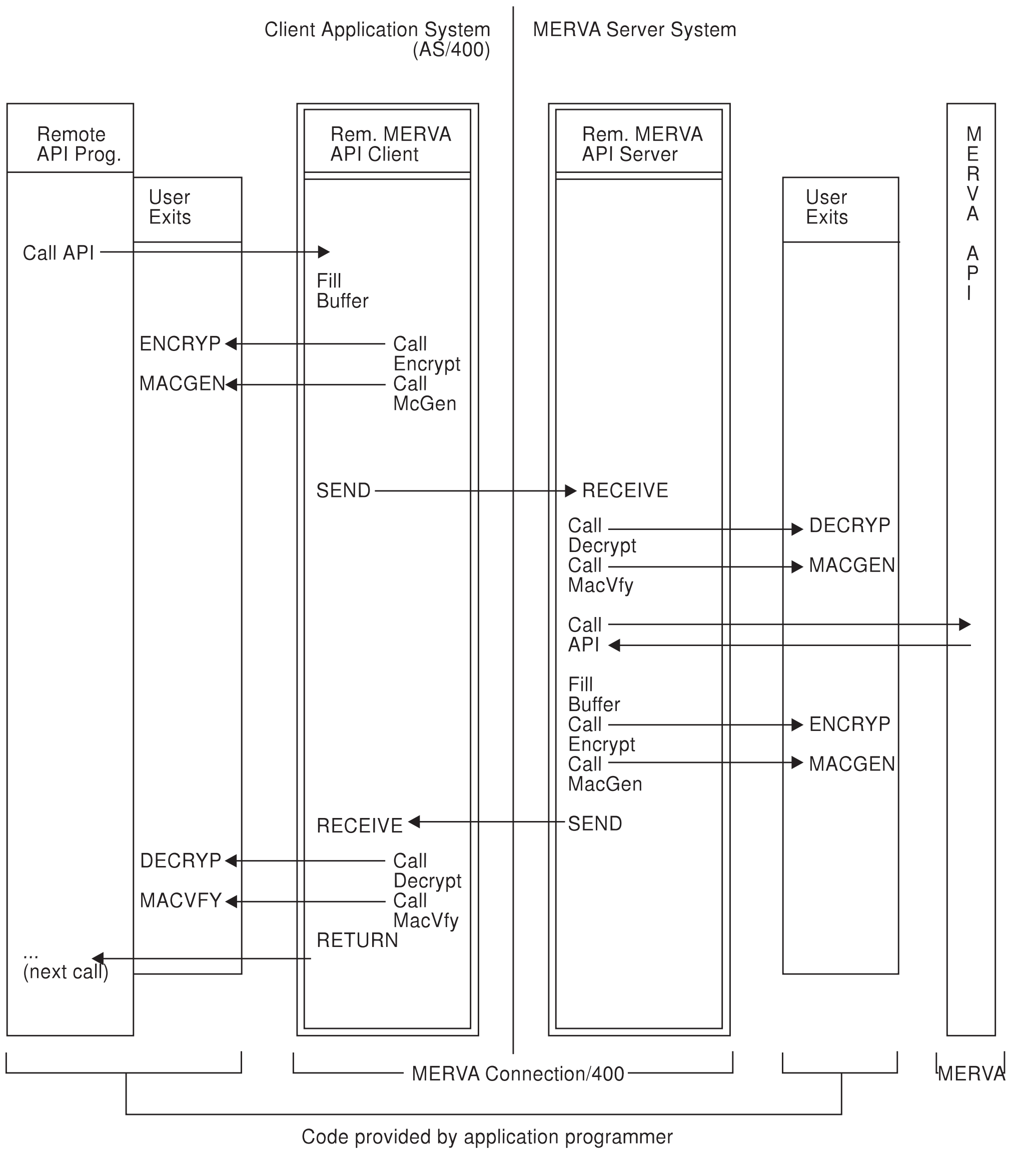

Figure 5. User Exit Points

User Exit Interfaces in C/ILE Language

The data types that are used in these routines can be different depending on the operating system on which they are implemented. For detailed information refer to "Appendix D. Sample Security User Exits" on page 93.

The following description of the user exit interface uses the sample program **ENM4SSEC**. The source and a compiled version of this program are delivered with MERVA Connection/400. To use the sample for your implementation of the user exits, do the following on the different operating systems:

AS/400 Link **ENM4SSEC** instead of **ENM4SNIL** as shown in "Generating and Activating Security User Exits on the AS/400" on page 84.

Windows NT	Copy **ENM4SSEC.DLL** to **ENM4SXIT.DLL**. In the delivered code, **ENM4SXIT.DLL** is a copy of **ENM4SNIL.DLL** in which all user exits are coded as empty functions.

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*)

  Address of a null-terminated string up to 8 characters long. The string contains the application identifier that is a parameter of the API call **ENMStartRAPI**. You can use this string to specify different encryption keys for different partner connections. You can also specify the connections or API programs for which the information is to be encrypted.
- **pucBuffer**(unsigned char*)

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

  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*)

  Address of a null-terminated string up to 8 characters long. The string contains the application identifier that is a parameter of the API call **ENMStartRAPI**. You can use this string to specify different decryption keys for different partner connections. You can also specify the connections or API programs for which the information is to be decrypted.
- **pucBuffer**(unsigned char*)

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

  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*)

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

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

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

  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 they do not match, set the return code to **1**.

Parameter Description

The following parameters are required:

- **pucApplId**(unsigned char*)

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

  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)

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

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

Replacing Security User Exits

This section describes how to generate and activate your own security user exits on AS/400 and Windows NT.

Generating and Activating Security User Exits on the AS/400

To access sample security exits on the AS/400, link **ENM4SSEC** instead of **ENM4SNIL**. For a detailed description refer to “User Exit Interfaces” on page 77. To replace the sample user exits by your own routines, use the module **ENM4SSEC** as a skeleton and compile it with the **CRTCMOD** command.

The following example shows you how to do this.

```
CRTCMOD PGM     (MYAPILIB/ENM4SSEC)
        SRCFILE (MYAPILIB/QCLESRC)
        SRCMBR  (ENM4SSEC)
        TEXT    ('ILE C/400 Security User Exit')
```

To bind the modules, use the the **CRTPGM** command as shown in the following example.

```
CRTPGM  PGM     (MYAPILIB/ENM4SSEC)
        MODULE  (MYAPILIB/ENM4SSEC
                 ENMRAPI/ENM4RAPI
                 ENMRAPI/ENM4RUTL
                 ENMRAPI/ENM4RPRF
                 ENMRAPI/ENM4SNIL)
```

If your module has a different name, replace **ENM4SSEC**.

Generating and Activating Security User Exits on the MERVA Server System

To access the sample security user exits on the Remote MERVA API Server on Windows NT, you must use the user exit routines from the shared library **enmcrxit.dll**. To replace the sample user exits by your own routines, use the file **enmcrsec.c** as a skeleton.

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

```
nmake /f enmnrsec.mak
```

Then, 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** in the make file **enmnrsec.mak** with your source file name.

Appendix A. Diagnosis Information

The diagnosis information is written to the log files on the AS/400 or the Windows NT system.

Log Files on the Remote MERVA API Client (AS/400)

On the AS/400, two log files are created. You can define the file names and the logging level in the MERVA Connection/400 profile.

For a description of the profile contents refer to "Changing Profile Settings" on page 4.

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 Windows NT.
- Trace information when the API trace is started with the call **ENMTrace**. For details refer to the *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 logging level in the MERVA Connection/400 profile specifies the amount of log information. The following logging levels are valid:

- Level 1: No data is written to the log.
- Level 2 and 3: No data is written to the log.
- Level 4: Records the activities of MERVA Connection/400 in detail. Use this level for debugging or demonstration purposes, for example, for the MERVA Connection/400 installation verification.

Log Message Layout

The log message consists of the message header and the message body. The following figure shows you an example of a diagnosis log:

```
*  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 APPC Initialization.

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

Figure 6. Example of Diagnosis Log

The layout of the message 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 body is:

Message	The message that is to be recorded. The length of the message can vary. For an explanation of the message, refer to the *MERVA USE & Branch for Windows NT Application Programming Guide*.

Note: The logs are not deleted automatically.

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 menu program.

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

Appendix B. Sample Network Definitions for the AS/400

This appendix shows you sample network definitions for the AS/400.

Communication Side Information (MERVA)

```
Side information  . . . . . . . . :   MERVA
Library . . . . . . . . . . . . . :   ENMRSMP
Remote location . . . . . . . . . :   CPNAME
Transaction program . . . . . . . :   ENMRAS
Device  . . . . . . . . . . . . . :   *LOC
Local location  . . . . . . . . . :   *LOC
Mode  . . . . . . . . . . . . . . :   *NETATR
Remote network identifier . . . . :   NETNAME       <= adapt to your needs
Text  . . . . . . . . . . . . . . :   Connection to MERVA Remote API server
```

Device Definition (MERVA)

```
Device description . . . . . . . . :   DEVD        MERVA
Option . . . . . . . . . . . . . . :   OPTION      *ALL
Category of device . . . . . . . . :               *APPC
Remote location  . . . . . . . . . :   RMTLOCNAME  CPNAME
Online at IPL  . . . . . . . . . . :   ONLINE      *NO
Local location . . . . . . . . . . :   LCLLOCNAME  AS400LU
Remote network identifier  . . . . :   RMTNETID    *NETATR
Attached controller  . . . . . . . :   CTL         MERVA
Message queue  . . . . . . . . . . :   MSGQ        QSYSOPR
  Library  . . . . . . . . . . . . :                 *LIBL
Local location address . . . . . . :   LOCADR      00
APPN-capable . . . . . . . . . . . :   APPN        *YES
Single session . . . . . . . . . . :   SNGSSN
  Single session capable . . . . . :               *NO
Text . . . . . . . . . . . . . . . :   TEXT        AUTOMATICALLY CREATED BY QLUS

-------------------------Mode--------------------------
*NETATR
```

Controller Definition (MERVA)

```
Controller description . . . . . . :   CTLD        MERVA
Option . . . . . . . . . . . . . . :   OPTION      *ALL
Category of controller . . . . . . :               *APPC
Link type  . . . . . . . . . . . . :   LINKTYPE    *LAN
Online at IPL  . . . . . . . . . . :   ONLINE      *YES
Active switched line . . . . . . . :               TOK1
Character code . . . . . . . . . . :   CODE        *EBCDIC
Maximum frame size . . . . . . . . :   MAXFRAME    16393
Remote network identifier  . . . . :   RMTNETID    NETNAME
Remote control point . . . . . . . :   RMTCPNAME   CPNAME
Initial connection . . . . . . . . :   INLCNN      *DIAL
Switched disconnect  . . . . . . . :   SWTDSC      *NO
Data link role . . . . . . . . . . :   ROLE        *NEG
LAN remote adapter address . . . . :   ADPTADR     FFFFFFFFFFFF
LAN DSAP . . . . . . . . . . . . . :   DSAP        04
LAN SSAP . . . . . . . . . . . . . :   SSAP        04
Text . . . . . . . . . . . . . . . :   TEXT        MERVA connection
Switched line list . . . . . . . . :   SWTLINLST

--------------------Switched Lines---------------------
TOK1
```

```
-------------------Attached Devices--------------------
MERVA

APPN-capable . . . . . . . . . . . :   APPN        *YES
APPN CP session support  . . . . . :   CPSSN       *YES
APPN node type . . . . . . . . . . :   NODETYPE    *CALC
APPN transmission group number . . :   TMSGRPNBR   *CALC
APPN minimum switched status . . . :   MINSWTSTS   *VRYONPND
Model controller description . . . :   MDLCTL      *NO
Control owner  . . . . . . . . . . :   CTLOWN      *USER

LAN frame retry  . . . . . . . . . :   LANFRMRTY   10
LAN connection retry . . . . . . . :   LANCNNRTY   10
LAN response timer . . . . . . . . :   LANRSPTMR   10
LAN connection timer . . . . . . . :   LANCNNTMR   70
LAN acknowledgement timer  . . . . :   LANACKTMR   1
LAN inactivity timer . . . . . . . :   LANINACTMR  100
LAN acknowledgement frequency  . . :   LANACKFRQ   1
LAN max outstanding frames . . . . :   LANMAXOUT   2
LAN access priority  . . . . . . . :   LANACCPTY   0
LAN window step  . . . . . . . . . :   LANWDWSTP   *NONE
Recovery limits  . . . . . . . . . :   CMNRCYLMT
  Count limit  . . . . . . . . . . :               2
  Time interval  . . . . . . . . . :               5
```

Mode Description for QPCSUPP

```
Mode description . . . . . . . . . :   MODD        QPCSUPP
Class-of-service . . . . . . . . . :   COS         #CONNECT
Maximum sessions . . . . . . . . . :   MAXSSN      64
Maximum conversations  . . . . . . :   MAXCNV      64
Locally controlled sessions  . . . :   LCLCTLSSN   0
Pre-established sessions . . . . . :   PREESTSSN   0
Inbound pacing value . . . . . . . :   INPACING    7
Outbound pacing value  . . . . . . :   OUTPACING   7
Maximum length of request unit . . :   MAXLENRU    *CALC
Text . . . . . . . . . . . . . . . :   TEXT        AS/400 PC Support mode entry
```

Note: Client Access/400 is not prerequisite for MERVA Connection/400. If Client Access/400 is installed, you can use the mode controller and device description that is configured for Client Access/400 also for MERVA Connection/400.

Appendix C. Sample Network Definitions for Windows NT

MERVA Connection/400 uses LU 6.2 sessions for the communication between the Remote MERVA API Client and Server in the SNA Data Communication environment.

This appendix shows a sample network definition for Windows NT.

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

APPN1
The name of the sample network.

ENA The control point name.

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

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

ASS400LU
The name of the partner LU (AS400).

Customizing an APPN End Node

For a detailed description on how to configure an end node in a two-node APPN (R) 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 3. Configure Node

Parameter	**Value**
Basic	
Fully qualified CP name	APPN, ENA
CP alias	ENA
Local node ID	No changes
Advanced	
Registration with network node server	Yes
Registration with central directory server	Yes
All others	No changes

Table 4. Configure Devices (DLC:LAN)

Parameter	**Value**
Adapter number	0 (Use the first available adapter number)
All others	No changes

Table 5. Configure Connections (DLC:LAN)

Parameter	**Value**
Basic	
Link station name	LINK0000
Destination address	4000400D6000
Advanced	
APPN support	Yes
Activate link at start	Yes
Link to preferred NN server	-
Adjacent Node	
Adjacent CP name	-
Adjacent CP type	APPN node
All others	No changes

Table 6. Configure Partner LU 6.2

Parameter	**Value**
Basic	
Partner LU name	APPN1, AS400LU
Partner LU alias	AS400LU
Fully qualified CP name	-
Advanced	
Conversation security support	Yes
All others	No changes

Table 7. Configure Modes

Parameter	**Value**
Basic	
Mode name	APPCLU62
Advanced	
Maximum RU size	1,024
All others	No changes

Table 8. Configure Local LU 6.2

Parameter	**Value**
Local LU name	LUA
Local LU alias	LUA
LU session limit	0 (no limit)
All others	No changes

Table 9. Configure Transaction Programs

Parameter	**Value**
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 10. Configure User ID Password

Parameter	**Value**
Security user ID	SAMPLE (or defined by you)
Security password	SAMPLE1 (corresponding password)

Appendix D. Sample Security User Exits

This appendix describes several sample security user exits that you can use.

Module ENM4SNIL - Empty Functions

The MERVA Connection/400 version supplied to you contains the module **ENM4SNIL**. Note that data that is transferred between the MERVA Server and the AS/400 is not encrypted. An authentication key is not built or transferred. You can use this program as a skeleton for your code.

Module ENM4SSEC - Sample Functions

The module ENM4SSEC shows you how to code your security functions. It includes basic encryption and authentication routines. Note that these routines do not provide genuine security.

Module ENM4SRPG - Calling Programs With One Entry Point

The module ENM4SRPG shows you how to call a single program for each security user exit. You can write the user exits in the RPG language.

User Exits as Single Programs on the AS/400

The following examples show you how to code your security functions. Basic encryption and authentication routines are included. Note that they do not provide genuine security.

- ENM4SMG – MAC Generation
- ENM4SMV – MAC Verification
- ENM4SEN – Encryption
- ENM4SDE – Decryption

Appendix E. 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:
- AIX
- AIX/6000
- AS/400
- C/2
- C/400
- C/ILE
- COBOL/400
- DATABASE 2
- DB2
- IBM
- MERVA
- Operating System/2
- OS/2
- OS/400
- Personal System/2
- PS/2
- RISC System/6000
- RPG/400

- RPG/ILE
- RS/6000
- SAA
- Systems Application Architecture

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

Pentium is a trademark of Intel Corporation.

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

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

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

Glossary of Terms and Abbreviations

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

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

MERVA ESA Components
MERVA Connection/400
Version 4 Release 1

Publication No. SH12-6340-01

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

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

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

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

Please tell us how we can improve this book:

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

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

Name	Address
Company or Organization	
Phone No.	

Readers' Comments — We'd Like to Hear from You
SH12-6340-01

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

SH12-6340-01

Cut or Fold
Along Line

Printed in Denmark by IBM Danmark A/S

SH12-6340-01