MERVA ESA Components

Directory Services

Version 4 Release 1

SH12-6367-00

MERVA ESA Components

Directory Services

Version 4 Release 1

SH12-6367-00

Note!
Before using this information and the product it supports, be sure to read the general information under "Appendix C. Notices" on page 45.

First Edition, March 2000

This edition applies to Version 4 Release 1 of IBM MERVA Components (5648-B30) and to all subsequent releases and modifications until otherwise indicated in new editions.

Contents

About This Book

This book relates to the IBM licensed program Message Entry and Routing with Interfaces to Various Applications for ESA Version 4 Release 1 (abbreviated to MERVA ESA in this book).

This book describes:

- The installation and customization of the MERVA ESA Directory Services
- The application programming interface (API) to use S.W.I.F.T. data in customer applications
- The identification and correction of problems

Who Should Read This Book

If you need to install and customize the MERVA ESA Directory Services, write application programs, or identify and correct problems, this book is for you.

Sending Your Comments

Your feedback is important in helping to provide the most accurate and high-quality information. If you have any comments about this book or any other MERVA ESA documentation:

- Send your comments by e-mail to SWSDID@DE.IBM.COM. Be sure to include the name of the book, the part number of the book, the version of MERVA ESA, and, if applicable, the specific location of the text you are commenting on (for example, a page number or table number).
- Fill out one of the forms at the back of this book and return it by mail, by fax, or by giving it to an IBM representative.

Chapter 1. Introducing the MERVA ESA Directory Services

The MERVA ESA Directory Services is an interface to connect customer applications and the **S.W.I.F.T. Directory Services** *Service Description Release 2.0, Annex A-MT293*.

This release of MERVA ESA Directory Services supports the Treasury Directory of the S.W.I.F.T. Directory Services. It lets you collect and process information such as standing settlement instructions (SSI) related to foreign exchange, money market, and derivative trades.

Figure 1 shows the infrastructure of the S.W.I.F.T. Directory Services.

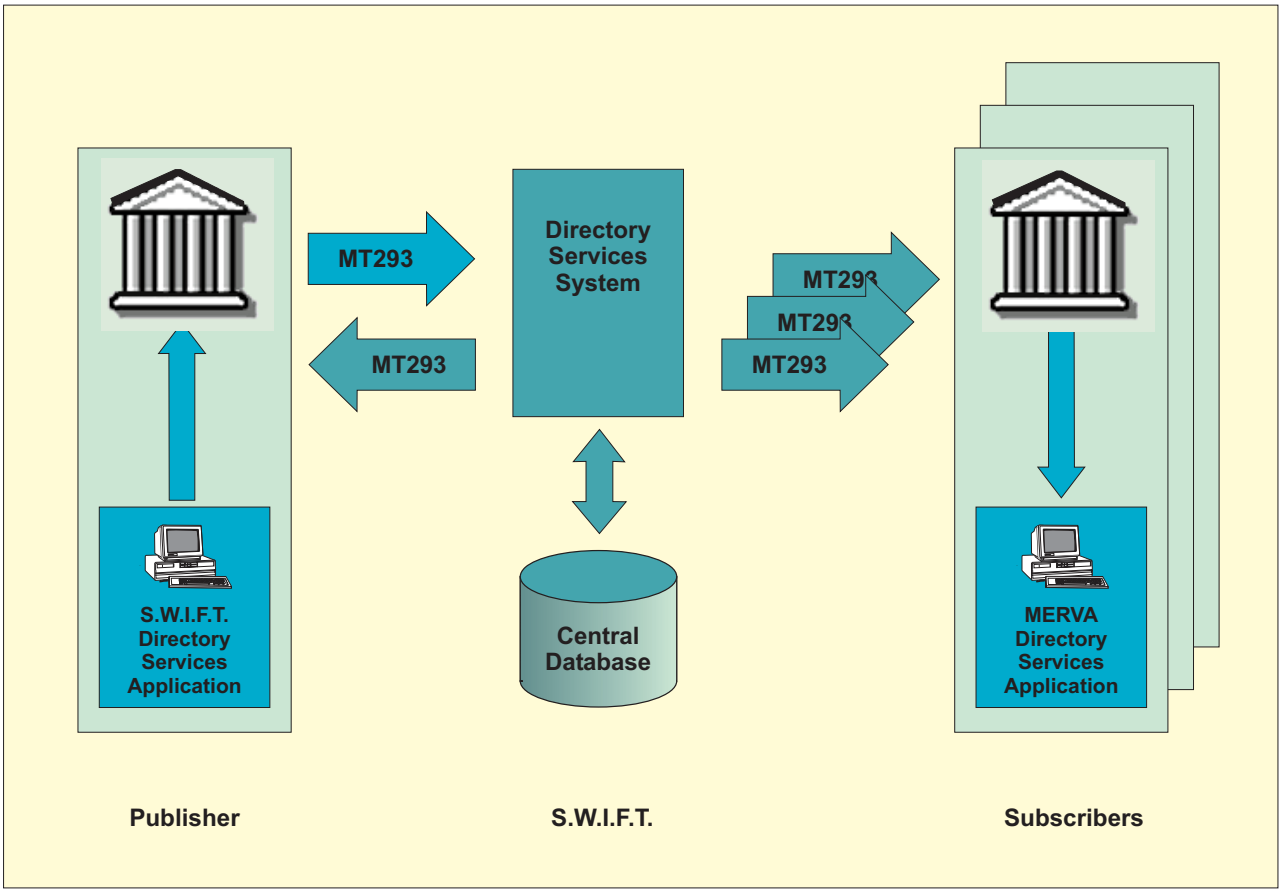

Figure 1. S.W.I.F.T. Directory Services

To exchange information between subscribers and the system an MT293 Information Service Message is used. See *S.W.I.F.T. Directory Services Service Description* for details about the MT293 message structure.

Note: This release of MERVA ESA Directory Services supports only MT293 messages where:

- The type of data is 3000 (the last part of field 14T).

and:

- The type of operation (field 22A) is A002 and the message type (field 0105) is INFODIST. This provides an update request.

  or:

- The type of operation (field 22A) is A001 and the message type (field 0105) is INFOANSW. This provides an answer to a query.

Figure 2 shows the MERVA Directory Services Application at the customer's site.

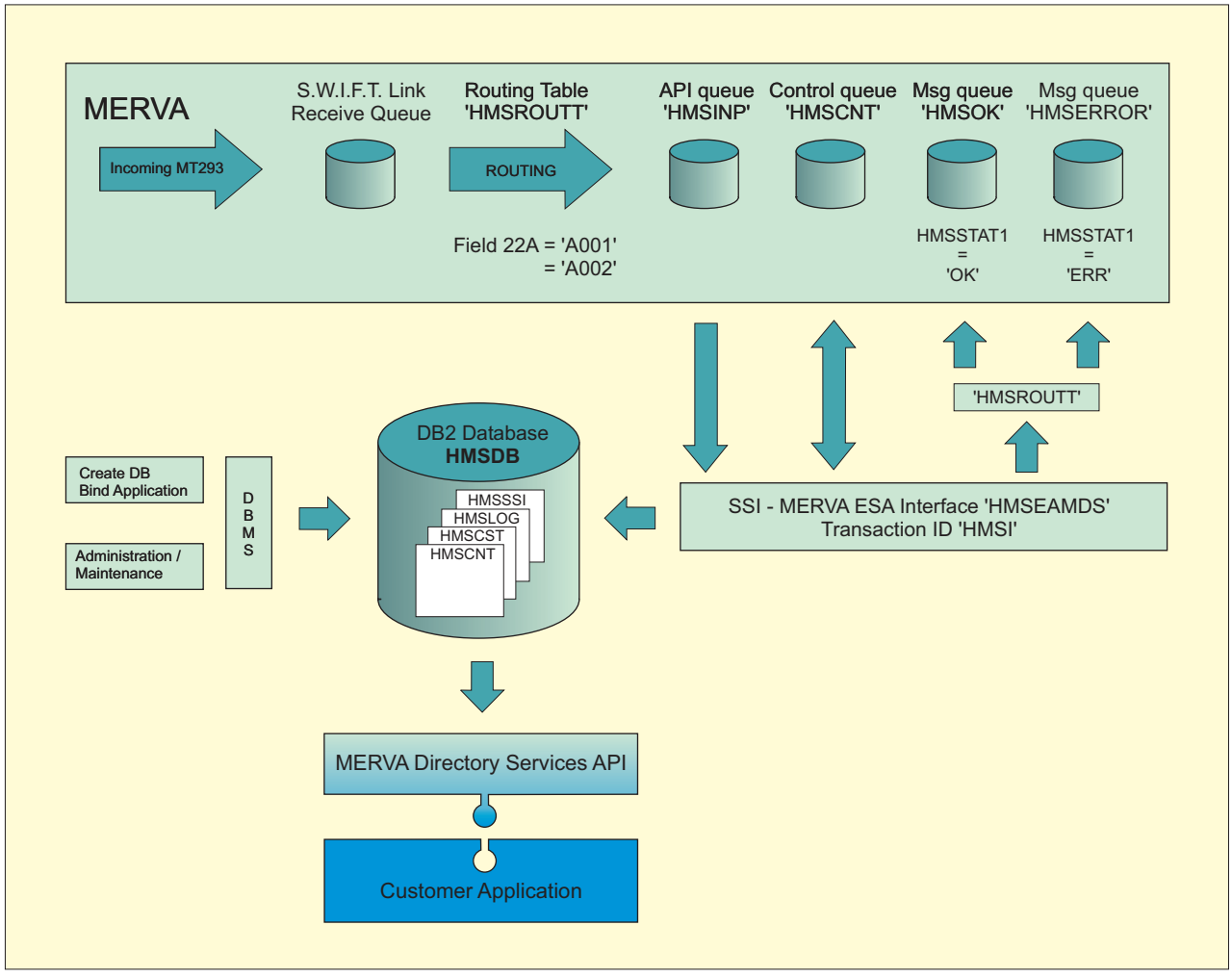

Figure 2. MERVA Directory Services Structure

The S.W.I.F.T. Link of MERVA ESA receives an MT293 message sequence and stores it in the S.W.I.F.T. Link receive queue. A sequence can contain one or more MT293 messages. Messages with 'A001' or 'A002' in field "22A" will be routed into the MERVA queue HMSINP. The incoming message starts the transaction HMSI

connected to the program HMSEAMDS. This program creates an internal control record for a new message or updates the record of a subsequent message in a sequence.

The message will be moved into the MERVA queue HMSCNT, where the program checks the completeness of a message sequence using the internal control records. When a message sequence is complete, HMSEAMDS concatenates all messages of the sequence to one large message.

The HMSEAMDS starts parsing this message. It collects the SSI data and checks it against the database. It adds the SSI record to the database, or deletes the record if the field '30Z0' contains 'NOOP'.

The program routes the message into the MERVA queue HMSOK or HMSERR, depending on the error condition.

The process ends with the deletion of the internal control record and the commit of the database updates.

The information content of the MT293 is now stored on the internal database and can be used by other internal applications.

Chapter 2. Planning for the MERVA ESA Directory Services

This chapter describes hardware and software requirements you need to install and use the MERVA ESA Directory Services.

Hardware Requirements

To work with MERVA ESA Directory Services you need no additional hardware on top of the MERVA ESA base installation. The base MERVA ESA machine requirements are described in detail in the *MERVA for ESA Installation Guide*.

Processors

Any processor that supports the specified operating system environment.

Peripheral Equipment

- At least one 3270-compatible display station, for example, 3278-2 or 3279-2
- An IBM 3268 printer or any printer operating in 3270 mode or SCS mode is optional
- A tape or cartridge drive for initial loading
- At least one direct-access storage device (DASD) for data sets and libraries; as supported by the operating system

Software Requirements

You need the following software components for MERVA ESA Directory Services:

- Operating system:
  - MVS/ESA SP Version 4 Release 2 or higher (5695-047) for JES2
  - MVS/ESA SP Version 4 Release 2 or higher (5695-048) for JES3
  - MVS/ESA SP Version 5 Release 1.0 (5655-068) for JES2
  - MVS/ESA SP Version 5 Release 1.0 (5655-069) for JES3
  - OS/390 Release 1 or higher (5645-001)
- Database:
  - DB2 for OS/390 Version 5 (5655-DB)
  - DB2/MVS (5655-DB2)
- IBM Language Environment for MVS/VM Version 1 Release 4 (5688-198)
- MERVA for ESA Version 3 for MVS/CICS (5655-039)
- MERVA for ESA Version 3 for MVS/IMS (5655-040)

Chapter 3. Installing and Customizing the Directory Services

The installation process for MERVA ESA Directory Services is described in the *Program Directory for MERVA ESA Directory Services*, which is delivered together with the machine-readable material.

To install MERVA ESA Directory Services, modify the following system areas:

- Update MERVA ESA
- Update DB2®
- Add definitions for CICS/ESA®
- Add definitions for IMS/ESA®

Updating MERVA ESA

MERVA ESA starts the program HMSEAMDS as soon as a message is routed into a queue. To enable this program to operate, do the following:

- Update the field definition table **DSLFDTT ASSEMBLE**

  Insert the member HMSFDTTC COPY, then compile and link the field definition table.

  The fields HMSSTAT1 and HMSKEY01 are defined in this table. The field HMSKEY01 contains a unique key for each message, created from the fields SW20 and SW28 of the message index. The field HMSSTAT1 holds status information needed to route messages out of the HMSCNT queue.
- Update the function table **DSLFNTT ASSEMBLE**

  Insert the member HMSFNTTC COPY, then compile and link the function table.

  This table contains the definition and the characteristics of the queues. It defines:
  - The transaction to start after a message is stored into HMSINP
  - The routing for a message coming out of the HMSCNT queue
  - The TOF to be used as key1 to get messages accessed by a key into the internal TOF buffer.

  For IMS only: The program HMSEAMDS supports the [,MSGLIM=(nnnnn[,CHKP | NOCHKP])].

  The first subparameter defines the number of messages that can be processed in a single schedule before control is returned to IMS. You can specify a number between 1 and 65535 for *nnnnn*. For more details see the DSLFNT macro in the *MERVA for ESA Macro Reference*.

Updating DB2

This section describes what you need to define, bind, and maintain the DB2 database.

Defining the Database

This version of MERVA ESA Directory Services uses the DB2 database **HMSDB**, which contains the following tables.

HMSSSI stores SSI-related information.

HMSLOG logs add, delete, and update processing on SSI records.

HMSCST contains the definition of the entry parameter for the transaction program HMSEAMDS.

HMSCNT stores control information for the transaction program HMSEAMDS.

The following Data Definition Language (DDL) members contain the table definitions:

1. HMSTS defines storage groups database and table space.

2. HMSTB defines tables and indexes.

3. HMSVIEW defines DB2 views.

The installation provides default values that do not require customizing. See the "HMSCST database table" on page 9 for these default settings.

However, you can change the following parameters to fit your needs:

- MAXDAYS
- TRACELVL
- TRACEAREA
- QNAME

Important:

1. You need DB2 access authorization to create or define the tables.
2. To create the database follow the order of the DDL members in the sequence listed.
3. With the authorization ID and the name of your DB2 table space you can use the DLL members as input data for the DB2 Online Utility SPUFI.
4. If you change the name of the control queue (HMSCNT), change the function table HMSFNTTC COPY and the routing table HMSROUTT ASSEMBLE accordingly.
5. Do not change the program name HMSCAMDS.

Table 1. HMSCST Database Table

Field	Description	Default
PGMNAME	Program ID	HMSCAMDS
TRANNAME	Transaction ID	HMSI
HMSKEY01	Used for recovery	
MAXDAYS	Specifies the number of days the program waits for completion of a message sequence.	5
TRACELVL	**0** = no **1** = low **2** = med **3** = large **4** = max	0
TRACEAREA	**0** = no **1** = HMSEAMDS **2** = database layer **4** = API **8** = parse of message	0
STATE	Used for recovery	
NAME	Name of control queue	HMSCNT

Binding the Application

Before the MERVA Directory Services can define, retrieve, and update the data in the database, you need to bind a DB2 application plan. Use the JCL member HMSBPLAN and the Database Request Modules (DRMs) delivered with the product to bind the plan.

Space You Need for Your Data Sets

The number of the SSI records can vary widely.

You can use the following formula to calculate the size of the table space:

$$\text{Size}_{\text{table space}} = \frac{\text{Number}_{\text{records}}}{\text{Number}_{\text{records per page}}} \times \text{Size}_{\text{page}}$$

Figure 3. Size of Table Space

Where:

- One SSI record has the size of about 480 bytes
- One LOG record has the size of about 333 bytes
- For every SSI record written or deleted, a LOG record will be written

Specify the table space value within DDL HMSTS accordingly.

Maintaining the Database

The transaction program HMSEAMDS fills the tables HMSSSI and HMSLOG. Delete or move the data out of the tables occasionally to avoid table overflow. You can use the fields ENTRYTIME in HMSSSI and LOGTIME in HMSLOG as criterion.

What You Need to Define for CICS/ESA

This section describes what you need to define when you work under CICS/ESA.

Resource Definition Jobs

Use the CSD utility DFHCSDUP to define the CICS/ESA resources. To run the MERVA ESA Directory Services under CICS/ESA:

- Add HMSGROUP to the GROUPLIST used in the CICS/ESA startup job (GRPLIST parameter of the DFHSIT macro).

  Note: Ensure that the GROUPLIST also contains the CICS-specific group DFHRMI (CICS® resource manager interface).
- Make sure that you have a group for DB2 and the CEE language environment. If these groups are not defined in your system, use the jobs in the CICS/ESA program directory to define them.
- Make sure that you have an entry in the 'Resource Control Table' to access DB2 from CICS. This entry defines the transaction needed to access a certain DB2 plan.
- **HMSCSD33** for CICS/ESA 3.3.0
- **HMSCSD41** for CICS/ESA 4.1.0

The Define statements are placed into the COPY books:

- **HMSCDxx** (definitions in HMSGROUP)
- **HMSDBxx** (definitions in DB2GROUP)

where xx can represent the following:

- **33**: HMSCS33/HMSDB33
- **41**: HMSCS41/HMSDB41

Add the following groups to the relevant GROUP LIST:

- **DB2GROUP**
- **DFHRMI**, the CICS resource manager interface

For more detailed information refer to the *CICS/ESA Resource Definition (Online)*.

Resource Control Table Entries

Add the following COPY book to the CICS/ESA Resource Control Table (**DSNCRCTx** for CICS330 or **DSN2CTxx** for CICS410):

- **HMSCRCT** (definitions of transactions and DB2 plans)

Note: CICS/ESA needs access to the **DSNCRCTx/DSN2CTxx** resource control table. Therefore it must be:

- A member of an APF-authorized data set
- Added to the STEPLIB data set concatenation of the CICS/ESA startup job

For more detailed information refer to the *DB2 Administration Guide*.

What You Need to Define for IMS/ESA

This section describes what you need to define when you work under IMS/ESA.

- Generate a PSB for the program HMSEAMDS
- Generate an ACB for the program HMSEAMDS
- Application definition

  To introduce the SSI application to your IMS, you need an entry in the IMS application definition (HMSIMSAP).

How to Initialize Your Installation

You have now finished your installation, but your database is still empty. MERVA ESA must receive an MT293 message and the MERVA ESA Directory Services API program must process the message to get the data into the database.

To request an MT293 send a query message to the S.W.I.F.T. Directory Services.

You will receive a query reply sent by the S.W.I.F.T. Directory Services, containing the SSI information records.

See an example of an “MT293 query message” on page 12.

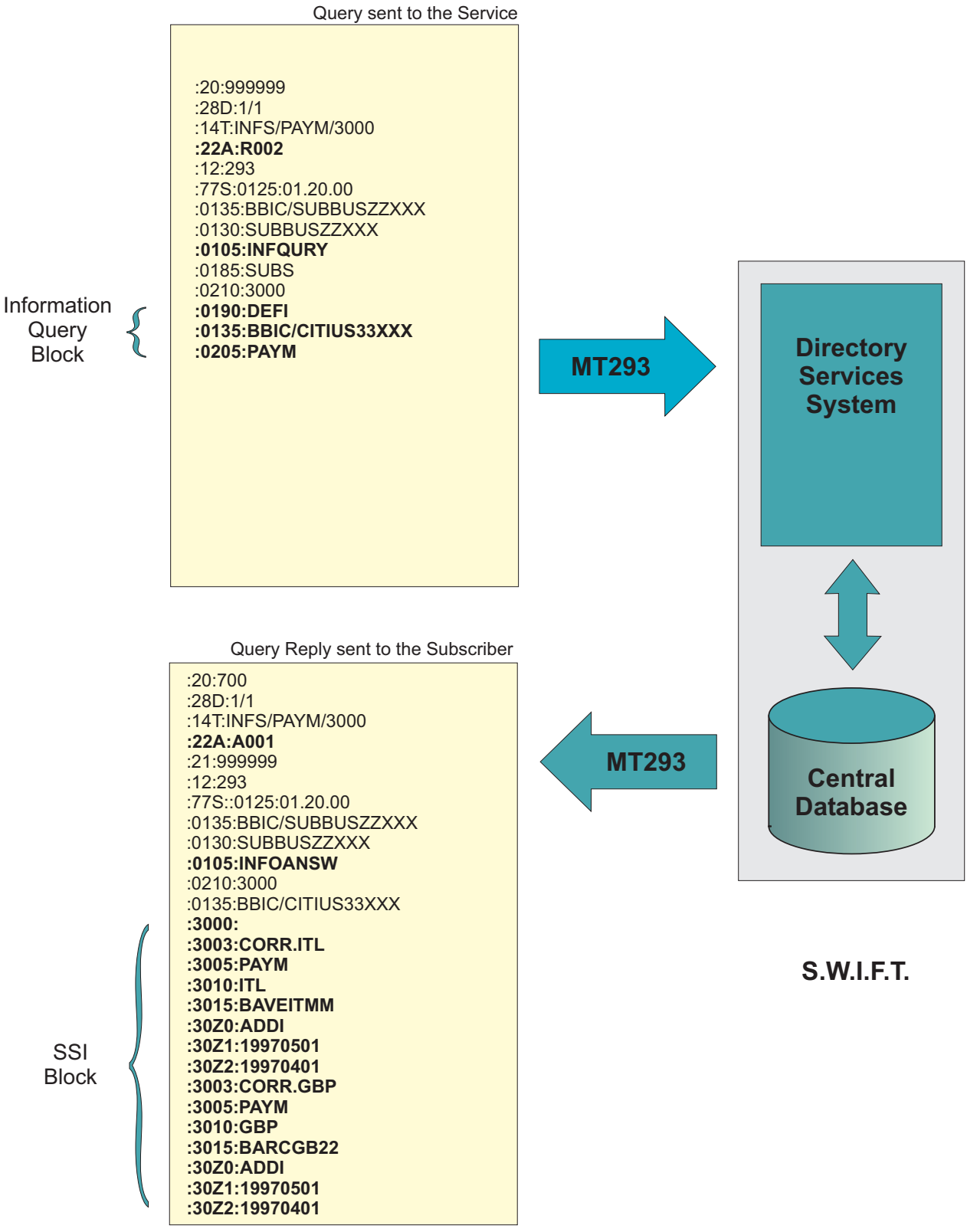

Figure 4. MT293 Query Message Example

For a description of the MT293 specifications, refer to *S.W.I.F.T. Directory Services Service Description*.

Chapter 4. Using the MERVA Directory Services API

This version of MERVA Directory Services supports the Treasury Directory. It allows to work with Standing Settlement Instructions (SSIs) supplied by the S.W.I.F.T. Directory Services (refer to the *S.W.I.F.T. Directory Services Service Description*).

The MERVA Directory Services Application Programming Interface allows other applications to access the directory service information (for example, SSI records). This interface lets the program read, add, delete, and update the SSI records.

API Data Types

This chapter lists definitions and API data types the API provides to facilitate the use of the interface.

HMS API Handle (HMSPHANDLE)

This data item must be provided with each API function call. The MERVA Directory Services API Handle holds all information needed to control the interface function calls. The memory is allocated inside *HMSInitApplication ()* and is freed with *HMSDisconnect()*. HMSPHANDLE is defined as:

```
typedef void HMSHANDLE
typedef HMSHANDLE *HMSPHANDLE
```

Service Type (HMSESERVICE)

The service type is an enumerated data type of valid Directory Services for accessing a Directory Services Database Table. Only Directory Service for SSI is available currently.

```
typedef enum {
   ESSISERVICE,
 } HMSESERVICE;
```

Application ID (HMSID)

The application ID data type is a character string containing the identification to an application. The name is used for logging purposes. The API supplies a length definition and the data type:

```
#define LSZID       9
typedef UCHAR    HMSID[LSZID];
typedef HMSID   *HMSPID;
```

Field Type (HMSEFIELDTYPE)

The field type is an enumerated data type of valid field names for data associated with an information record. The API supplies the data type:

```
typedef enum {
   EENTRYTIME
   EOWNER,
   ESERVICER,
   EMESSAGETYPE,
   ETYPEOFMARKET,
   ETYPEOFDISTRBUTION,
   ECONTACTPERSON,
   EACKNOWLEDGEMENTIND,
   EUNIQUEID,
   EMARKETAREA,
   ECURRENCYCODE,
```

```
   ECORRESPONDENTBIC,
   EINTERMEDIARYBIC,
   ECORRESPONDENTACCOUNT,
   EONLYFORNEWDEALS,
   ERECONFIRMATION,
   ETYPEOFOPERATION,
   EEFFECTIVEDATE,
   EPUBLISHDATE,
   ERELATEDREFERENCE,
   ETRN,
} HMSEFIELDTYPE;
```

The following table shows whether a field is mandatory or not when working with the SSI Directory Services.

Table 2. Fields and Message Tags

Field	Message Tag	Mandatory
EOWNER	:0135:	N
ESERVICER	:0130:	Y
EMESSAGETYPE	:0105:	Y
ETYPEOFMARKET	part of 14T	Y
ETYPEOFDISTRBUTION	:0185:	see note
ECONTACTPERSON	:0140:	N
EACKNOWLEDGEMENTIND	:0170:	see note
EUNIQUEID	:3003:	Y
EMARKETAREA	:3005:	Y
ECURRENCYCODE	:3010	Y
ECORRESPONDENTBIC	:3015:	Y
EINTERMEDIARYBIC	:3020:	N
ECORRESPONDENTACCOUNT	:3030:	N
EONLYFORNEWDEALS	:3035:	N
ERECONFIRMATION	:3040:	N
ETYPEOFOPERATION	:22A:	Y
EEFFECTIVEDATE	:30Z1:	Y
EPUBLISHEDDATE	:30Z2:	N
ERELATEDREFERENCE	:21:	N
ETRN	:20:	N

Note: This field is mandatory only when MESSAGETYPE = INFODIST.

Field (HMSUFIELD)

The field data type is a union of data fields associated with an information record and accessible through the API. The API supplies the length definition for the fields, the union of fields, and a pointer to the union data type:

```
#define LSZENTRYTIME              27
#define LSZOWNER                  22
#define LSZSERVICER               12
#define LSZMESSAGETYPE             9
#define LSZTYPEOFMARKET            5
#define LSZTYPEOFDIST             22
```

```
#define LSZCONTACTPERSON          241
#define LSZUNIQUEID                17
#define LSZMARKETAREA               5
#define LSZCURRENCYCODE             4
#define LSZCORRESPONDENTBIC        12
#define LSZINTERMEDIARYBIC         12
#define LSZCORRESPONDENTACCOUNT    41
#define LSZTYPEOFOPERATION          5
#define LSZEFFECTIVEDATE            9
#define LSZPUBLISHDATE              9
#define LSZTRN                     17
#define LSZRELREF                  17

typedef union {
   UCHAR szEntryTime[LSZENTRYTIME];
   UCHAR szOwner[LSZOWNER];
   UCHAR szServicer[LSZSERVICER];
   UCHAR szMessageType[LSZMESSAGETYPE];
   UCHAR szTypeOfMarket[LSZTYPEOFMARKET];
   UCHAR szTypeOfDistribtion[LSZTYPEOFDIST];
   UCHAR szContactPerson[LSZCONTACTPERS];
   UCHAR szAcknowledgementInd;
   UCHAR szUniqueId[LSZUNIQUEID];
   UCHAR szMarketArea[LSZMARKETAREA];
   UCHAR szCurrencyCode[LSZCURRENCYCODE];
   UCHAR szCorrespondentBic[LSZCORRESPONDENTBIC];
   UCHAR szIntermediaryBic[LSZSSIINTERMEDIARYBIC];
   UCHAR szCorrespondentAccount[LSZCORRESPONDENTACCOUNT];
   UCHAR szOnlyForNewDeals;
   UCHAR szReconfirmation[LSZRECONFIRMATIOM];
   UCHAR szTypeOfOperation[LSZTYPEOFOPERATION];
   UCHAR szEffectiveDate[LSZEFFECTIVEDATE];
   UCHAR szPublishDate[LSZPUBLISHDATE];
   UCHAR szTrn[LSZTRN];
   UCHAR szRelatedRef[LSZRELATEDREF];
 } HMSUFIELD;
typedef HMSUFIELD   *HMSPFIELD
```

Key Type (HMSEKEYTYPE)

The key type is an enumerated data type of valid key types for searching for an information record. The API supplies the data type:

```
typedef enum {
   ECURRENCYKEY,
   EMARKETKEY,
   ECURRMARKETKEY,
   ECORRESPONDENTKEY,
   EUNIQUEIDKEY,
   EENTRYTIMEKEY,
 } HMSEKEYTYPE;
```

Search Key (HMSUKEY)

The search key data type is a union of key types. It is used when retrieving an information record. The API supplies the length definitions and the data type:

```
typedef union {
   UCHAR szCurrencyCode[LSZCURRENCYCODE];
   UCHAR szMarket[LSZMARKETARERA];
   UCHAR szCorrespondentBic[LSZCORRESPONDENTBIC];
   UCHAR szUniqueID[LSZCUNIQUEID];
   UCHAR szEntryTime[LSZENTRYTIME];
 } HMSUKEY;
typedef HMSUKEY      *HMPSUKEY;
```

ErrRep (HMSSTERRORREPORT)

The error report data type is a structure of different data items. It passes error information to the error function provided with the *HMSInitApplication()* function call.

```
#define LSZERRORTEXT  241
typedef struct {
   HMSEERRORTYPE  eType;
   INT iRc;
   INT iRs;
   INT iLine
   UCHAR szErrorInfo[LSZERRORTEXT];
   UCHAR szErrorMsg[LSZERRORTEXT];
} HMSSTERRORREPORT;
```

where HMSEERRORTYPE is an enumeration of values that specifies the area where the error occurred.

```
typedef enum {
   ESQLERROR,
   EOSERROR,          /* Operating System error occurred        */
   ESYSERROR,         /* MERVA Directory Service error occurred */
   EUSERERROR,
} HMSEERRORTYPE;
```

API Functions

This section contains, in alphabetical order, descriptions of each MERVA Directory Services API function. The description of each function has the following structure:

Purpose
A short description of the function's purpose

Format
The syntax of the function; its name in mixed case, the number and order of parameters

Input Parameters
A description of each input parameter of the function in the following format:

Parameter name (data type of parameter)
Description of the parameter

Output Parameters
A list of return values valid for this function in the following format:

rc (numerical return code)
Description of the return codes

Processing (optional)
Some hints on special conditions that may occur while processing this function

Example
An example of how to call the function

Note: The examples provided are without error handling and not related to a specific system.

HMSAdd

Purpose

Use this function to add a created information record to the Directory Services Database Table.

Format

HMSAdd (pvHandle)

Input Parameters

pvHandle (HMSPHANDLE)
Application handle to information used by the MERVA Directory Services API.

Output Parameters

rc(INT)
Possible return codes are:

0 (HMSDNOERROR)
Function completed successfully.

8 (HMSDERROR)
Function not completed successfully. See "Appendix B. Reason Codes" on page 41 for information about the error.

Example

The following example shows you how to use the HMSAdd call to add a new SSI information record to the MERVA ESA Directory Services Database.

```
#include <stdio.h>
#include <string.h>
#include <stdlib.h>

/* - MERVA Directory Services include files - */
#include "hmscaapi.h"

/* ---------------------------------------- */
/* - simple SSI record structure definition - */
/*   used to hold information of one record   */
/*   ( not all possible fields are used! )    */

typedef struct {
 char szTypeOfMarket [LSZTYPEOFMARKET];         /* part of :14T:      */
 char szServicer [LSZSERVICER];                 /* :0130:             */
 char szMessageType [LSZMESSAGETYPE];           /* :0105:             */
 char szTypeOfDistribution [LSZTYPEOFDIST];     /* :0185:             */
 char cAcknowledgmentIndicator;                 /* :0170:             */
 char szUniqueId [LSZUNIQUEID];                 /* :3003:             */
                                       /* identifies each corres.*/
 char szMarketArea [LSZMARKETAREA];             /* :3005:             */
 char szCurrencyCode [LSZCURRENCYCODE];         /* :3010:             */
 char szCorrespondentBic [LSZCORRESPONDENTBIC]; /* :3015:             */
 char szTypeOfOperation [LSZTYPEOFOPERATION];   /* :30Z0:             */
 char szEffectiveDate [LSZEFFECTIVEDATE];       /* :30Z1:             */
} MYSSIREC;


/* ---------------------------------------- */
/* - function prototypes defined within     - */
/*   this module.                             */

int   PrintError( HMSSTERRORREPORT*  pError );
```

```
/* ---------------------------------------- */
/* - global used variables                  - */

static HMSHANDLE * pHandle;

#ifdef __HMSIMS__
/* Datatype IO_PCB_TYPE comes from <ims.h> */
IO_PCB_TYPE * pPCB;
#else
void * pPCB = (void*)NULL;
#endif

int main( int argc, char *argv[] )
{
   INT             irc = 0;
   HMSUKEY         ssikey;
   HMSUFIELD       Field;
   MYSSIREC        aSSIRec =
   { "FXMM", "SERVICERXXX", "MESGTYPE", "TYPEOFDISTRIBUTIONXXX", 'Y',
     "UNIQUEIDXXXXXXXX", "FXCH", "XXX", "CORRESPONDE", "XXZ0", "EFFEDATE"
};

   irc = HMSInitApplication( &pHandle, PrintError, (void*) pPCB );
   if ( !irc ) {

      irc = HMSConnect( pHandle, "HMSEXAPI" )
      if (!irc) {

         irc = HMSCreate( pHandle, ESSISERVICE );

         strcpy( Field.szUniqueId, aSSIRecs.szUniqueId );
         irc += HMSWriteField( pHandle, EUNIQUEID, &Field  );
         strcpy( Field.szCurrencyCode, aSSIRecs.szCurrencyCode );
         irc += HMSWriteField( pHandle, ECURRENCYCODE, &Field  );
         strcpy( Field.szMarketArea, aSSIRecs.szMarketArea );
         irc += HMSWriteField( pHandle, EMARKETAREA, &Field  );
         strcpy( Field.szCorrespondentBic, aSSIRecs.szCorrespondentBic);
         irc += HMSWriteField( pHandle, ECORRESPONDENTBIC, &Field  );
         strcpy( Field.szTypeOfOperation , aSSIRecs.szTypeOfOperation );
         irc += HMSWriteField( pHandle, ETYPEOFOPERATION, &Field  );
         strcpy( Field.szEffectiveDate   , aSSIRecs.szEffectiveDate );
         irc += HMSWriteField( pHandle, EEFFECTIVEDATE, &Field  );
         strcpy( Field.szServicer        , aSSIRecs.szServicer );
         irc += HMSWriteField( pHandle, ESERVICER, &Field  );
         strcpy( Field.szTypeOfMarket    , aSSIRecs.szTypeOfMarket  );
         irc += HMSWriteField( pHandle, ETYPEOFMARKET, &Field  );
         strcpy( Field.szMessageType     , aSSIRecs.szMessageType );
         irc += HMSWriteField( pHandle, EMESSAGETYPE, &Field  );
         strcpy( Field.szTypeOfDistribution,
                 aSSIRecs.szTypeOfDistribution);
         irc += HMSWriteField( pHandle, ETYPEOFDISTRIBUTION, &Field  );
         Field.cAcknowledgmentIndicator = aSSIRecs.cAcknowledgmentIndicator;
         irc += HMSWriteField( pHandle, EACKNOWLEDGMENTIND, &Field  );
         if (!irc) {
            irc = HMSAdd( pHandle );
            if ( irc < 8 )
               irc = HMSCommit( pHandle );
            else
               irc = HMSRollback( pHandle );
         }
         irc = HMSDisconnect( &pHandle );
      }
   }
   return( irc );
}

int PrintError( HMSSTERRORREPORT*  pError )
```

```
{
static char TypeText [][5] = {{"SQL"},{"OS"},{"SYS"},{"USER"}};
static char szOutputString [sizeof(HMSSTERRORREPORT)+80];

   sprintf( szOutputString, "ERROR: %d Area %s\n",
            (INT)pError->eType, TypeText[(INT)pError->eType] );
   sprintf( szOutputString, "%src=%d, rs=%d, Line=%d\n",
            szOutputString, pError->iRc, pError->iRs, pError->iLine );
   sprintf( szOutputString, "%serrinfo %.80s\n",
             szOutputString, pError->szErrorInfo );
   sprintf( szOutputString, "%serrmesg %.80s\n",
             szOutputString,pError->szErrorMsg  );
   printf( szOutputString );

   return (0);
}
```

HMSClear

Purpose

Use this function to free the information record space created with the HMSCreate function.

Format

HMSClear (pvHandle)

Input Parameters

pvHandle (HMSPHANDLE)
Application handle to information used by the MERVA Directory Services API.

Output Parameters

rc(INT)
Possible return codes are:

0 (HMSDNOERROR)
Function completed successfully.

8 (HMSDERROR)
Function not completed successfully. See "Appendix B. Reason Codes" on page 41 for information about the error.

Example

The following example shows you how to use the HMSClear call to clear the internal record buffer. The type of the record buffer is not changed.

```
....
int main( int argc, char *argv[] )
....
         irc = HMSCreate( pHandle, ESSISERVICE );
         while( NotReady ) {
            /* insert data into SSI structure */
            SetSSIData(&aSSIRecs);

            /* insert data into SSI record buffer */
            /* with HMSWriteField()               */
            SetSSIRecordBuffer(pHandle, &aSSIRecs);

            /* insert data into MERVA Directory   */
            /* Services database; record type is  */
            /* set by HMSCreate to ESSISERVICES   */
            irc = HMSAdd(pHandle);

            /* clear the MERVA Directory Services */
            /* SSI record space created with      */
            /* HMSCreate()                        */
            irc = HMSClear(pHandle);

            /* test if all records processed      */
            NotReady = SetReady();

         } /* end while */

         .....
```

HMSCommit

Purpose
Use this function to set all changes permanent.

Note: When you run your application under CICS or IMS, the CICS/IMS transaction calls, like file operations or terminal output, are committed.

Format

HMSCommit (pvHandle)

Input Parameters

pvHandle (HMSPHANDLE)
Application handle to information used by the MERVA Directory Services API.

Output Parameters

rc(INT)
Possible return codes are:

0 (HMSDNOERROR)
Function completed successfully.

8 (HMSDERROR)
Function not completed successfully. See "Appendix B. Reason Codes" on page 41 for information about the error.

Processing
This function uses:

- EXEC SQL COMMIT
- EXEC CICS SYNCPOINT under **CICS**
- ctdli (..., "chkp", ...) under **IMS**

Example
The following example code is an extract of the HMSAdd example. You find the complete coding example "HMSADD" on page 17.

```
...
        if (!irc) {
           irc = HMSAdd( pHandle );
           if ( irc < 8 )
              irc = HMSCommit( pHandle );
           else
              irc = HMSRollback( pHandle );
         }
...
```

HMSConnect

Purpose

Connect the application to the MERVA Directory Services Database. pvDBHandle is allocated by function *HMSInitApplication().*

Note: This function may not be used under CICS/IMS. In this case the logging entry field is empty.

Format

HMSConnect (pvHandle, ApplicationID)

Input Parameters

pvHandle (HMSPHANDLE)
Application handle to information used by the MERVA Directory Services API.

ApplicationID (HMSPID)
A string specifying the application ID used for logging. This string must end in NULL or a zero.

Output Parameters

rc(INT)
Possible return codes are:

0 (HMSDNOERROR)
Function completed successfully.

8 (HMSDERROR)
Function not completed successfully. See "Appendix B. Reason Codes" on page 41 for information about the error.

Example

The following example code is an extract of the HMSAdd example. You find the complete coding example "HMSADD" on page 17.

```
...
   irc = HMSInitApplication( &pHandle, PrintError, (void*) pPCB );
   if ( !irc ) {

      irc = HMSConnect( pHandle, "HMSEXAPI" )
      if (!irc) {
...
      }
...
   }
...
```

HMSCreate

Purpose
Use this function to create a new information record for the application.

Format

HMSCreate (pvHandle, eServiceType)

Input Parameters

pvHandle (HMSPHANDLE)
Application handle to information used by the MERVA Directory Services API.

eServiceType (HMSESERVICE)
Specifies the Directory Service for which a record is created (currently ESSISERVICE only).

Output Parameters

rc(INT)
Possible return codes are:

0 (HMSDNOERROR)
Function completed successfully.

8 (HMSDERROR)
Function not completed successfully. See "Appendix B. Reason Codes" on page 41 for information about the error.

Processing
This function resets an internal memory buffer and prepares it to hold an information record of the type "eService".

The buffer write/read operation is done with HMSWriteField() and HMSReadField().

Note: HMSWriteField() overwrites the buffer. All information fetched or created previously will be lost.

Example
The following example code is an extract of the HMSAdd example. You find the complete coding example "HMSADD" on page 17.

```
...
       irc = HMSConnect( pHandle, "HMSEXAPI" )
       if (!irc) {
...
          irc = HMSCreate( pHandle, ESSISERVICE );
          strcpy( Field.szUniqueId,aSSIRecs.szUniqueId );
          irc += HMSWriteField( pHandle, EUNIQUEID, &Field  );
...
       }
```

HMSDelete

Purpose
Use this function to delete the currently fetched information record.

Format

HMSDelete (pvDBHandle)

Input Parameters

pvHandle (HMSPHANDLE)
Application handle to information used by the MERVA Directory Services API.

Output Parameters

rc(INT)
Possible return codes are:

0 (HMSDNOERROR)
Function completed successfully.

8 (HMSDERROR)
Function not completed successfully. See "Appendix B. Reason Codes" on page 41 for information about the error.

Processing
HMSDelete() uses field EENTRYTIME as key. This field is set by a previous call to HMSKeyRead(), HMSKeyReadNext(), or HMSWriteField(). HMSDelete() closes the cursor, which is opened by a previous function call to HMSKeyRead.

Note: A function call to HMSKeyReadNext fails after function HMSDelete(). HMSAdd() and HMSUpdate() write an entry in the MERVA ESA Directory Services Logging Database Table.

Example
The following example shows you how to use the HMSDelete call to delete an SSI information record. All records with the UniqueId "123456" will be deleted.

```
int DeleteRecords(void)
{
   int             irc = 0;
   HMSUKEY         ssikey;

   strcpy( ssikey.szUniqueId, "123456" );

   while ( !irc ) {

      irc = HMSKeyRead( pHandle,      /* can not use HMSKeyReadNext  */
                        ESSISERVICE,  /* because HMSDelete close     */
                        EUNIQUEIDKEY, /* the cursor and ..Next needs */
                        &ssikey,      /* an open cursor.             */
                        NULL );
      if ( !irc )
         irc = HMSDelete( pHandle );

   } /* endwhile */
   return (irc);
}
```

HMSDisconnect

Purpose

Use this function to disconnect the application from the MERVA Directory Services Database. It frees the interface data structure allocated by the *HMSInitApplication()* function call.

Format

HMSDisconnect (&pvHandle)

Input Parameters

&pvHandle(HMSPHANDLE*)
Reference to the database information handle used by the MERVA Directory Services API.

Output Parameters

rc(INT)
Possible return codes are:

0 (HMSDNOERROR)
Function completed successfully.

8 (HMSDERROR)
Function not completed successfully. See "Appendix B. Reason Codes" on page 41 for information about the error.

Example

The following example code is an extract of the HMSAdd example. You find the complete coding example "HMSADD" on page 17.

```
...

   irc = HMSInitApplication( &pHandle, PrintError, (void*) pPCB );
   if ( !irc ) {
...
      irc = HMSConnect( pHandle, "HMSEXAPI" )
      if (!irc) {
...
         }
         irc = HMSDisconnect( &pHandle );
      }
...
```

HMSGetErrorInfo

Purpose

Use this function to retrieve the address of the error report. This report contains information about errors that occurred inside an HMS-API function call.

For more details on errors see "Appendix A. Error Handling" on page 39.

Format

HMSGetErrorInfo(pvHandle, &pErrRep)

Input Parameters

&pvHandle(HMSPHANDLE)
Application handle to information used by the MERVA ESA Directory Services API.

Output Parameters

rc(INT)
Possible return codes are:

0 (HMSDNOERROR)
Function completed successfully.

8 (HMSDERROR)
Function not completed successfully. See "Appendix B. Reason Codes" on page 41 for information about the error.

Example

The following example shows you how to use the HMSGetErrorInfo call to retrieve the error structure. Because each HMS API function returns only 0 or 8, with this function you get access to the error information.

```
....

      irc = HMSKeyRead( pHandle,
                        ESSISERVICE,
                        EUNIQUEIDKEY,
                        &ssikey,
                        NULL );
      if ( irc ) {
         HMSPERRORREPORT  pstError;    /* see also HMSAdd() */

         irc = HMSGetErrorInfo( pHandle, &pstError );

         if ( pstError->iRs == HMSDSDBCRC ){
            /* checksum calculation error;               */
            /* fetched DB record is manipulated manually; */
            /* use HMSReadField() to read record field    */

            ....

         }

      } else {

....
```

HMSInitApplication

Purpose

This function allocates memory to store application-relevant information used by all application interface functions. The structure of the memory buffer is hidden. The reference to the data buffer is assigned to the first parameter *pvHandle*. The second parameter is a reference to a function that all interface functions call in case of an error condition.

Format

HMSInitApplication (&pvHandle, pfError)

Input Parameters

&pvHandle(HMSPHANDLE*)
Reference to the database information handle used by the MERVA ESA Directory Services API.

pfError(INT(*)(HMSPERRORREPORT))
Reference to a function called inside the interface functions when an error occurs.

Output Parameters

rc(INT)
Possible return codes are:

0 (HMSDNOERROR)
Function completed successfully.

8 (HMSDERROR)
Function not completed successfully. See "Appendix B. Reason Codes" on page 41 for information about the error.

Processing

The function is using the C-language function *malloc()* to allocate the memory.

Example

The following example code is an extract of the HMSAdd example. You find the complete coding example "HMSADD" on page 17.

```
...
   irc = HMSInitApplication( &pHandle, PrintError, (void*) pPCB );
   if ( !irc ) {

      irc = HMSConnect( pHandle, "HMSEXAPI" )
      if (!irc) {
...
      }
...
   }
...
```

HMSKeyRead

Purpose

This function searches the named directory service information for the first information record with the specified key.

Format

HMSKeyRead (pvHandle, eServiceType, eKeyType, puKey1, puKey2)

Input Parameters

pvHandle (HMSPHANDLE)
Application handle to information used by the MERVA Directory Services API.

eServiceType (HMSESERVICE)
Specifies the Directory Service for which a record is created (currently ESSISERVICE only).

eKeyType (HMSEKEYTYPE)

This parameter determines the search condition used for the specified Directory Service. When eKeyType is ECURRMARKETKEY, then puKey1 contains the currency value and puKey2 contains the market value.

puKey1 (HMSPUKEY)
This parameter contains the search argument according to the KEYTYPE.

puKey2 (HMSPUKEY)
This parameter contains a second search argument according to the KEYTYPE.

Output Parameters

rc(INT)
Possible return codes are:

0 (HMSDNOERROR)
Function completed successfully.

8 (HMSDERROR)
Function not completed successfully. See "Appendix B. Reason Codes" on page 41 for information about the error.

Processing

When no search argument is given, the search starts at the first element of the information records, sorted by KEYTYPE and record entry time.

Example

The following example code is an extract of the HMSKeyReadNext example. You find the complete coding example "HMSKeyReadNext" on page 31.

```
...
   HMSUKEY         ssikey1;
   HMSUKEY         ssikey2;

   strcpy( ssikey1.szCurrencyCode, "USD" );
   strcpy( ssikey2.szMarketArea,   "FXMM" );

   irc = HMSKeyRead( pHandle,
                     ESSISERVICE,
                     ECURRMARKETKEY,
                     &ssikey1,
                     &ssikey2 );
...
```

HMSKeyReadNext

Purpose

This function returns the next information record matching the conditions set by a previous HMSKeyRead function call (HMSESERVICE; HMSEKEYTYPE; HMSPUKEYs).

Format

HMSKeyReadNext (pvHandle)

Input Parameters

pvHandle (HMSPHANDLE)
Application handle to information used by the MERVA Directory Services API.

Output Parameters

rc(INT)
Possible return codes are:

0 (HMSDNOERROR)
Function completed successfully.

8 (HMSDERROR)
Function not completed successfully. See "Appendix B. Reason Codes" on page 41 for information about the error.

Processing

The function requires search conditions to be defined. Search conditions can only be defined by a call to the HMSKeyRead function.

Note: After a call to function HMSUpdate(), HMSDelete(), or HMSAdd(), the underlying DB2 cursor is closed. A subsequent call to HMSKeyReadNext() fails when no call to function HMSKeyRead() is done before.

Example

The following example shows you how to use the HMSKeyReadNext call to retrieve a list of SSI information records from the MERVA ESA Directory Services database. All records with the CurrencyCode “USD” and MarketArea “FXMM” will be retrieved.

```
...

   HMSUKEY         ssikey1;
   HMSUKEY         ssikey2;

   strcpy( ssikey1.szCurrencyCode, "USD" );
   strcpy( ssikey2.szMarketArea,   "FXMM" );

   irc = HMSKeyRead( pHandle,
                     ESSISERVICE,
                     ECURRMARKETKEY,
                     &ssikey1,
                     &ssikey2 );

   while ( !irc ) {

      /* PrintRecord() uses function HMSReadField() to retrieve */
      /* data of the specified fields.                          */

      (void) PrintRecord( pHandle );

      irc = HMSKeyReadNext( pHandle );

   } /* end while */

....
```

HMSReadField

Purpose

Use this function to return data associated with an information record.

Format

HMSReadField (pvHandle, FieldType, &pField)

Input Parameters

pvHandle (HMSPHANDLE)
Application handle to information used by the MERVA Directory Services API.

FieldType(HMSUFIELDTYPE)
The field type contains the name of the field the application wants to read.

&pField(HMSPUFIELD*)
The field union contains the reference of the field the application wants to read.

Output Parameters

rc(INT)
Possible return codes are:

0 (HMSDNOERROR)
Function completed successfully.

8 (HMSDERROR)
Function not completed successfully. See "Appendix B. Reason Codes" on page 41 for information about the error.

Example

The following example shows you how to use the HMSReadField call to retrieve a specified record field. In this example field "EffectiveDate" is used to find the current SSI record.

```
int  GetCurrent( char    *   pszDate )  /* pszDate => current date */
{
   int             irc    = 0;
   int             iFound = 0;
   HMSPUFIELD      pField;
   HMSUFIELD       ssiCurrentEffDate;
   HMSUFIELD       ssiCurrentEntryTime;
   HMSUKEY         ssikey1;
   HMSUKEY         ssikey2;

   strcpy( ssikey1.szCurrencyCode, "USD" );
   strcpy( ssikey2.szMarketArea,   "FXMM" );
   ssiCurrentEffDate.szEffectiveDate[0] = '\0';

   irc = HMSKeyRead( pHandle,
                     ESSISERVICE,
                     ECURRMARKETKEY,

                     &ssikey1,
                     &ssikey2 );
   while ( !irc ) {

      irc = HMSReadField( pHandle, EEFFECTIVEDATE, &pField );

      if ( strcmp(pszDate, pField->szEffectiveDate) >= 0  ) {
```

```
         if ( ( ssiCurrentEffDate.szEffectiveDate[0] == '\0') ||
              (strcmp(ssiCurrentEffDate.szEffectiveDate,
                      pField->szEffectiveDate) == -1 )) {
               iFound = 1;
               strcpy( ssiCurrentEffDate.szEffectiveDate,
                       pField->szEffectiveDate );

               irc = HMSReadField( pHandle, EENTRYTIME, &pField  );
               strcpy( ssiCurrentEntryTime.szEntryTime,
                       pField->szEntryTime );
         }
      } /* endif */

      irc = HMSKeyReadNext( pHandle );

   } /* endwhile */
   if ( iFound ) {

      irc = HMSKeyRead( pHandle,
                        ESSISERVICE,
                        EENTRYTIMEKEY,
                        &ssiCurrentEntryTime,
                        NULL );
      if ( !irc ) {
         /* record found */
         .....

      }
   }
   return( irc );
}
```

HMSRollback

Purpose
Use this function to reset all changes done to the database after the last start of the transaction.

Format

HMSRollback (pvHandle)

Input Parameters

pvHandle (HMSPHANDLE)
Application handle to information used by the MERVA Directory Services API.

Output Parameters

rc(INT)
Possible return codes are:

0 (HMSDNOERROR)
Function completed successfully.

8 (HMSDERROR)
Function not completed successfully. See "Appendix B. Reason Codes" on page 41 for information about the error.

Example
The following example code is an extract of the HMSAdd example. You find the complete coding example "HMSADD" on page 17.

```
...
        if (!irc) {
           irc = HMSAdd( pHandle );
           if ( irc < 8 )
              irc = HMSCommit( pHandle );
           else
              irc = HMSRollback( pHandle );
        }
...
```

HMSUpdate

Purpose
Use this function to change the currently fetched information record.

Format

HMSUpdate (pvHandle)

Input Parameters

pvHandle (HMSPHANDLE)
Application handle to information used by the MERVA Directory Services API.

Output Parameters

rc(INT)
Possible return codes are:

0 (HMSDNOERROR)
Function completed successfully.

8 (HMSDERROR)
Function not completed successfully. See "Appendix B. Reason Codes" on page 41 for information about the error.

Processing
All changes to the Directory Services Database must be committed. Use *HMSCommit()* to confirm your changes.

You can use *HMSRollback()* if you want to set the state of the database content to a previous commit point.

Example
The following example shows you how to use the HMSUpdate call to update an SSI information record. The SSI record must be fetched before it can be updated. The example updates all CorrespondentBic to "DEUTDEFFXXX", where UniqueId is "123456". Because HMSUpdate closes the cursor, it is necessary to store the key of the last record updated. With this key it is possible to loop through the cursor up to the next valid record.

```
....

   int             irc = 0;
   HMSUKEY         ssikey;
   HMSUFIELD       Field;
   HMSPUFIELD      pField;
   HMSPUFIELD      pFieldKey;
   CHAR            szEntryTimeLast[LSZENTRYTIME] = "";

   strcpy( ssikey.szUniqueId, "123456" );

   while ( !irc ) {

      irc = HMSKeyRead( pHandle,
                        ESSISERVICE,
                        EUNIQUEIDKEY,
                        &ssikey,
                        NULL );

      irc = HMSReadField( pHandle, EENTRYTIME, &pFieldKey );
      while ( !irc &&                          /* already processed ? */
              (strcmp( szEntryTimeLast, pFieldKey->szEntryTime )>= 0) ){
```

```
      irc = HMSKeyReadNext( pHandle );
      irc = HMSReadField( pHandle, EENTRYTIME, &pFieldKey );
   }
   if (!irc) {
      strcpy( szEntryTimeLast, pFieldKey->szEntryTime );
      strcpy( Field.szCorrespondentBic, "DEUTDEFFXXX" );
      irc = HMSWriteField( pHandle, ECORRESPONDENTBIC, &Field );
      irc = HMSUpdate( pHandle );
   }

} /* endwhile */
....
```

HMSWriteField

Purpose
This function updates information associated with an information record.

Format

HMSWriteField (pvHandle, FieldType, pField)

Input Parameters

pvHandle (HMSPHANDLE)
Application handle to information used by the MERVA Directory Services API.

FieldType(HMSEFIELDTYPE)
The field type contains the name of the field the application wants to write.

pField(HMSPUFIELD)
The field union contains the contents of the field the application wants to write.

Output Parameters

rc(INT)
Possible return codes are:

0 (HMSDNOERROR)
Function completed successfully.

8 (HMSDERROR)
Function not completed successfully. See "Appendix B. Reason Codes" on page 41 for information about the error.

Example
The following example code is an extract of the HMSUpdate example. You find the complete coding example "HMSUpdate" on page 35.

```
...
        strcpy( szEntryTimeLast, pFieldKey->szEntryTime );
        strcpy(Field.szCorrespondentBic, "DEUTDEFFXXX" );
        irc = HMSWriteField( pHandle, ECORRESPONDENTBIC, &Field );
        irc = HMSUpdate( pHandle );
...
```

Appendix A. Error Handling

The function call *HMSInitApplication* enables the application to define a function which is called in case of an error condition. The parameter of this function is a pointer to structure HMSSTERRORREPORT. This structure is described on page 15.

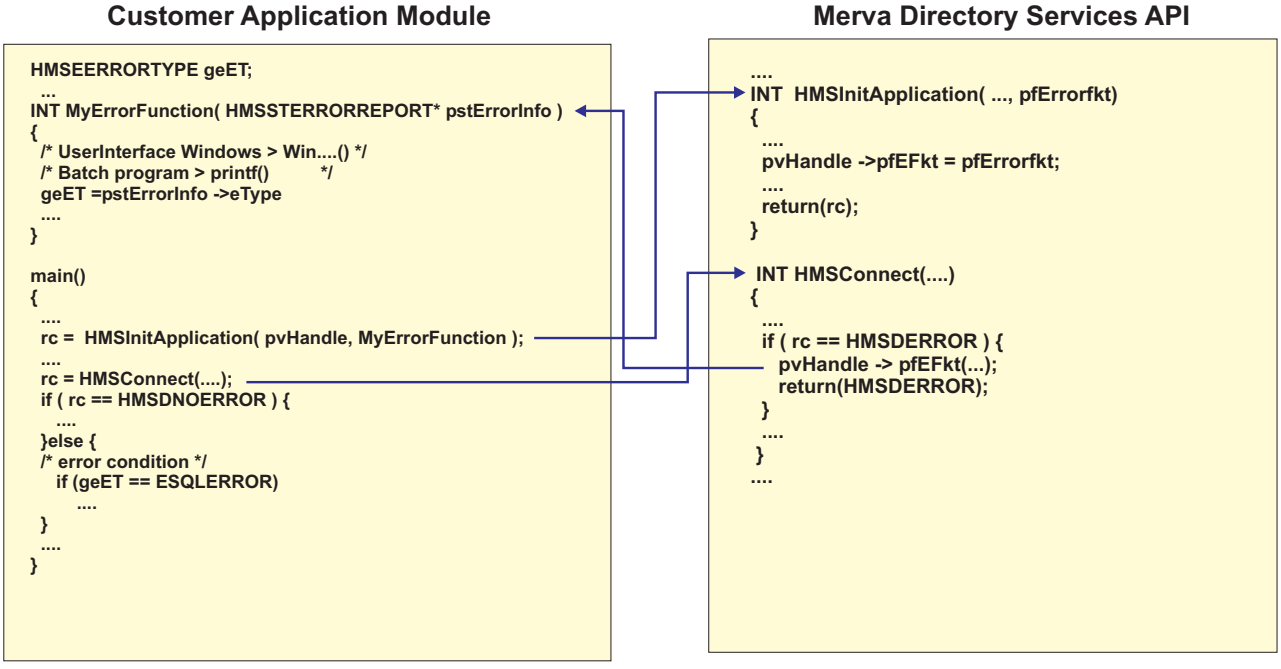

Figure 5. Error Handling

To avoid the use of global variables in the error handling, the function HMSGetErrorInfo () is provided. This function allows the application to access the error report.

The error report contains the following fields:

eType The error type. These error types are valid types:
- SQL errors
- User errors
- Operating system errors
- MERVA ESA Directory Services errors

iRc Return code. Contains the value 0 (no error) or 8 (error).

iRs Identifies the error reason. See "Appendix B. Reason Codes" on page 41 for details.

iLine Contains the line number in the source code, where the error has occurred.

szErrorInfo A general string containing eType, iRc, and iRs.

szErrorMsg A string containing detailed information about the error.

Appendix B. Reason Codes

This appendix lists error messages. You find a description of the area and the function in which an error occurred.

Understanding the Message Format

The reason codes are shown in mixed-case letters, as displayed. A variable part of the code is printed in *lowercase italics*, for example:

HMSnnnE message text *variable part*

The example shows a MERVA ESA program message consisting of message number and message text.

HMS The product identifier of MERVA ESA Directory Services.

nnnx This is a message identification number of 3 digits **nnn** with a 1-character action code **x**.

The following action codes are possible:

E Shows an error

I Shows an information message

nnnn This is a message identification number of 4 digits. No action code is provided.

message text
This is the text of the message as it appears on the workstation or as it is printed on a hardcopy or system printer.

The text of the message can contain *variable* information, for example, *rc* for return code, or other information that provides more details.

When a return code or a reason code appears in a message, it is described in the explanation given for that message, or a reference is made to the appropriate chapter in this book.

variable
A *variable* can be one of the following:

d Indicates a decimal number

i Indicates an integer

s Indicates a character

General Error Messages

HMS001E SQL ERROR Function *"s"*; **rc=***"d"*; **rs=***"d"*

Explanation: An SQL function call caused this error.

Action: The reason code contains the SQLCODE of the SQL function. The 'szErrorMsg' data field of the error report contains the explanation of the SQLCODE.

HMS002E USER ERROR Function *"s"*; **rc=***"d"*; **rs=***"d"*

Explanation: A USER error occurred.

Action: The 'szErrorMsg' data field of the error report contains the explanation of the USER error.

HMS003E SYS ERROR Function *"s"*; rc=*"d"*; rs=*"d"*

Explanation: A MERVA ESA Directory Services System Error occurred.

Action: The 'szErrorMsg' data field of the error report contains the explanation of the error. Contact your IBM representative.

HMS004E OS ERROR Function *"s"*; rc=*"d"*; rs=*"d"*

Explanation: An operating system error occurred.

Action: The 'szErrorMsg' data field of the error report contains the explanation of the error.

HMS005E /* Gnrl. error message */ *"s"*; rc=*"d"*; rs=*"d"*

Explanation: This error is not assigned to a function explicitly.

Action: The 'szErrorMsg' data field of the error report contains the explanation of the error. Contact your IBM representative.

Reason Codes

HMS100E The database handle is not valid.

Explanation: The handle passed to the function is not initialized properly.

Action: Internal error. Contact your IBM representative.

HMS101E Specified retrieve option is not defined.

Explanation: The retrieve option passed to the HmsDBRead() is not valid. Valid options are, foe example, EDBFIRST and EDBLAST.

Action: Internal error. Contact your IBM representative.

HMS102E The combination of table and index is not valid.

Explanation: The combination of table and index by function call HmsDBRead() is not valid. The index for the specified table is not defined.

Action: Internal error. Contact your IBM representative.

HMS103E It is not possible to change the direction of reading rows.

Explanation: The database cursor is open for only one direction of reading. Therefore it is not possible to change the direction within the cursor.

Action: Internal error. Contact your IBM representative.

HMS104E The program tries to fetch a record but no cursor is open.

Explanation: The program tries to fetch a record with no cursor open. Call function HmsDBRead() with EDBFIRST and EDBLAST, before using (EDBNEXT and EDBPREV).

Action: Internal error. Contact your IBM representative.

HMS105E The mandatory field *"s"* of the supplied data record is empty.

Explanation: A field of a data record, defined as mandatory, is empty.

Action: Use HmsWriteField() to set all mandatory fields.

HMS106E The pointer to the data record is NULL.

Explanation: None.

Action: Internal error. Contact your IBM representative.

HMS107E The pointer to IMS PCB is NULL; ctdli() will fail.

Explanation: Error occurred during Rollback or Commit. The pointer to IMS PCB needed for function call ctdli() is Null.

Action: Use HMSInitApplication() to set the pointer.

HMS150E The calculated and the stored check sum differ.

Explanation: The cycle redundancy check detected a database record change. The record might have been changed inside the database table directly.

Action: The record is supplied with error indication by the API function. Check the record fields and update the record.

HMS151E IMS: ctdli() error, stat_code=*"s"*.

Explanation: A call to IMS failed.

Action: Check the status code.

HMS200E The parameter *"s"* is empty or string is too long.

Explanation: The content of a passed parameter is empty or string is too long.

Action: Check the parameter of the function.

HMS201E The application handle is not valid.

Explanation: The application handle passed to the function is not initialized properly.

Action: Use HMSInitApplication() to create a valid handle.

HMS202E The type of record is not valid.

Explanation: The internal record buffer is not set correctly. (A previous call to HMSKeyRead() might have been unsuccessful.)

Action: Call HMSCreate() or HMSKeyRead() before the function call to HMSDelete(), HMSAdd(), and HMSUpdate().

HMS203E Passed key is not valid for specified service.

Explanation: The parameter of type HMSKEYTYPE is not valid.

Action: Check the value within function HMSKeyRead().

HMS204E Specified service is not defined.

Explanation: The parameter of type HMSESERVICE is not valid.

Action: Check the value within function HMSKeyRead().

HMS205E Wrong type of field specified.

Explanation: The parameter of type HMSFIELDTYPE is not valid.

Action: Check the value within function HMSReadField() or HMSWriteField().

HMS206E Wrong type of record specified by write.

Explanation: The application tried to write a field into the internal record buffer. The record buffer is not initialized properly.

Action: Use the function HMSCreate() before HMSWriteField().

HMS250E No valid record available.

Explanation: The application tried to read a field from the internal record buffer. The record buffer is not initialized properly. (A previous call to HMSKeyRead() might have been unsuccessful.)

Action: A valid database record must be fetched before. Check the return value.

HMS251E Bad table ID used by function KeyReadNext().

Explanation: The internal database table identifier is not valid. (The function HMSKeyread() sets this identifier.)

Action: Internal error. Contact your IBM representative.

HMS252E *"s"* **EXE CICS call failed, RESP=***"i"*, **RESP2=***"i"*, **RC=***"i"*.

Explanation: An EXEC CICS call failed.

Action: Make sure that the CICS system is started properly.

HMS253E *"s"* **INTFUNC=***"s"* **false, INTRC=***"s"* **RC=***"i"*.

Explanation: The initialization of the MERVA ESA API program failed.

Action: Make sure that the DSLAPI program is available in one of the STEPLIBs of your startup job.

HMS254E *"s"* **API function=***"s"* **false, INTRC=***"s"*.

Explanation: A function call to the MERVA ESA API program failed.

Action: Make sure that the DSLAPI program is available in one of the STEPLIBs of your startup job.

HMS255E *"s"* **API function=***"s"* **false, INTRC=***"s"*, **TOFTSVRC=***"i"* **TOFTSVRS=***"i"*.

Explanation: A function call to the MERVA ESA API program failed.

Action: Make sure that the DSLAPI program is available in one of the STEPLIBs of your startup job.

HMS256E *"s"* **DB function =***"s"* **failed, RC=***"i"*.

Explanation: An access to the database failed.

Action: See the previous SQL message.

HMS257E *"s"* **Message is longer than** *"i"* **days in the** *"s"* **queue, expected** *"i"*, **received only** *"i"* **messages.**

Explanation: The message sequence you received is incomplete. The system waiting for the missing parts of the sequence exceeded the defined time value MAXDAYS.

Action: Try to get the missing message of the sequence, then put the complete sequence into the HMSINP queue.

HMS258E Transaction is not startable from outside of MERVA.

Explanation: You have tried to start the transaction HMSI from outside of MERVA ESA. This is not possible.

Action: None.

HMS259E Call to IMS interface failed, function="s", PCB status field="s".

Explanation: A call to IMS failed.

Action: Check the status code of the failing IMS function.

HMS260E Read TOF field failed, fieldname="s".

Explanation: Reading the field named "s" from the MERVA ESA internal message buffer (TOF) failed.

Action: Correct the error in field "s" of the message and restart.

HMS261I TOF data size greater than target field length.

Explanation: The size of a TOF field is greater than the target buffer.

HMS262E Retrieve of MERVA internal variable failed, variable name="s".

Explanation: Reading the field named "s" from a message control block (MCB) failed.

Action: Check MCB and message for a definition of field "s".

HMS263E "s" Function="s" failed, iRC="i".

Explanation: A call to database services failed.

Action: See the previous SQL message.

HMS264E Final state machine state "i" is invalid.

Explanation: An internal error occurred. The state of the processing sequence is unexpected.

Action: Internal error. Contact your IBM representative.

HMS265E The value "s" is invalid for SW22, only "A001" and "A002" will be processed.

Explanation: A message was routed into the wrong queue.

Action: Make sure the **type of operation** field SW22 of your message contains the value **A001** or **A002**. Only these two values are allowed.

Note: Field SW22 works in conjunction with field SW105. See the following table for the two valid combinations of the fields SW22 and SW105.

	Field SW22	**Field SW105**
Combination 1	A001	INFOANSW
Combination 2	A002	INFODIST

HMS266E The value "s" is invalid for SW105, only "INFODIST" and "INFOANSW" are supported.

Explanation: A message was routed into the wrong queue.

Action: Make sure the **message type** field SW105 in your message contains the value **INFODIST** or **INFOANSW**. Field SW105 works in conjunction with field SW22. See the the table for message HMS265E for the two valid combinations of the fields SW22 and SW105.

HMS267E %s FETCH 'DSLAPI' failed.

Explanation: Load of the MERVA ESA API program failed.

Action: Make sure that the DSLAPI program is available in one of the STEPLIBs of your startup job.

HMS999E No message ID for supplied reason code.

Explanation: The system could not resolve the reason code into a message text.

Action: Internal error. Contact your IBM representative.

Appendix C. Notices

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

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

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

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

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

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

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

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

IBM Deutschland
Informationssysteme GmbH
Department 3982
Pascalstrasse 100

70569 Stuttgart
Germany

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

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

The following paragraph does apply to the US only.

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

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

COPYRIGHT LICENSE:

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

Trademarks

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

- AIX
- C/370
- CICS
- CICS/ESA
- CICS/MVS
- CICS/VSE
- DB2
- IBM
- IMS/ESA
- MQSeries
- MVS
- MVS/ESA
- MVS/XA
- OS/2
- OS/390
- QMF

- RACF
- S/390
- VSE/ESA
- VTAM

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

Pentium is a trademark of Intel Corporation.

PC Direct is a trademark of Ziff Communications Company and is used by IBM Corporation under license.

C-bus is a trademark of Corollary, Inc.

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

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

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

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

Glossary of Terms and Abbreviations

This glossary defines terms as they are used in this book. If you do not find the terms you are looking for, refer to the *IBM Dictionary of Computing*, New York: McGraw-Hill, and the *S.W.I.F.T. User Handbook*.

A

ACB. Access method control block.

ACC. MERVA Link USS application control command application. It provides a means of operating MERVA Link USS in USS shell and MVS batch environments.

access method control block (ACB). A control block that links an application program to VSAM or VTAM.

ACD. MERVA Link USS application control daemon.

ACT. MERVA Link USS application control table.

address. See *S.W.I.F.T. address*.

address expansion. The process by which the full name of a financial institution is obtained using the S.W.I.F.T. address, telex correspondent's address, or a nickname.

AMPDU. Application message protocol data unit, which is defined in the MERVA Link P1 protocol, and consists of an envelope and its content.

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

answerback code. A group of up to 6 letters following or contained in the answerback. It is used to check the answerback.

APC. Application control.

API. Application programming interface.

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

APPL. A VTAM definition statement used to define a VTAM application program.

application programming interface (API). An interface that programs can use to exchange data.

application support filter (ASF). In MERVA Link, a user-written program that can control and modify any data exchanged between the Application Support Layer and the Message Transfer Layer.

application support process (ASP). An executing instance of an application support program. Each application support process is associated with an ASP entry in the partner table. An ASP that handles outgoing messages is a *sending ASP*; one that handles incoming messages is a *receiving ASP*.

application support program (ASP). In MERVA Link, a program that exchanges messages and reports with a specific remote partner ASP. These two programs must agree on which conversation protocol they are to use.

ASCII. American Standard Code for Information Interchange. The standard code, using a coded set consisting of 7-bit coded characters (8 bits including parity check), used for information interchange among data processing systems, data communication systems, and associated equipment. The ASCII set consists of control characters and graphic characters.

ASF. Application support filter.

ASF. (1) Application support process. (2) Application support program.

ASPDU. Application support protocol data unit, which is defined in the MERVA Link P2 protocol.

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

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

authenticator-key file. The file that stores the keys used during the authentication of a message. The file contains a record for each of your financial institution's correspondents.

B

Back-to-Back (BTB). A MERVA Link function that enables ASPs to exchange messages in the local MERVA Link node without using data communication services.

bank identifier code. A 12-character code used to identify a bank within the S.W.I.F.T. network. Also called a S.W.I.F.T. address. The code consists of the following subcodes:

- The bank code (4 characters)
- The ISO country code (2 characters)
- The location code (2 characters)
- The address extension (1 character)

- The branch code (3 characters) for a S.W.I.F.T. user institution, or the letters "BIC" for institutions that are not S.W.I.F.T. users.

Basic Security Manager (BSM). A component of VSE/ESA Version 2.4 that is invoked by the System Authorization Facility, and used to ensure signon and transaction security.

BIC. Bank identifier code.

BIC Bankfile. A tape of bank identifier codes supplied by S.W.I.F.T.

BIC Database Plus Tape. A tape of financial institutions and currency codes, supplied by S.W.I.F.T. The information is compiled from various sources and includes national, international, and cross-border identifiers.

BIC Directory Update Tape. A tape of bank identifier codes and currency codes, supplied by S.W.I.F.T., with extended information as published in the printed BIC Directory.

body. The second part of an IM-ASPDU. It contains the actual application data or the message text that the IM-AMPDU transfers.

BSC. Binary synchronous control.

BSM. Basic Security Manager.

BTB. Back-to-back.

buffer. A storage area used by MERVA programs to store a message in its internal format. A buffer has an 8-byte prefix that indicates its length.

C

CBT. S.W.I.F.T. computer-based terminal.

CCSID. Coded character set identifier.

CDS. Control data set.

central service. In MERVA, a service that uses resources that either require serialization of access, or are only available in the MERVA nucleus.

CF message. Confirmed message. When a sending MERVA Link system is informed of the successful delivery of a message to the receiving application, it routes the delivered application messages as CF messages, that is, messages of class CF, to an ACK wait queue or to a complete message queue.

COA. Confirm on arrival.

COD. Confirm on delivery.

coded character set identifier (CCSID). The name of a coded set of characters and their code point assignments.

commit. In MQSeries, to commit operations is to make the changes on MQSeries queues permanent. After putting one or more messages to a queue, a commit makes them visible to other programs. After getting one or more messages from a queue, a commit permanently deletes them from the queue.

confirm-on-arrival (COA) report. An MQSeries report message type created when a message is placed on that queue. It is created by the queue manager that owns the destination queue.

confirm-on-delivery (COD) report. An MQSeries report message type created when an application retrieves a message from the queue in a way that causes the message to be deleted from the queue. It is created by the queue manager.

control fields. In MERVA Link, fields that are part of a MERVA message on the queue data set and of the message in the TOF. Control fields are written to the TOF at nesting identifier 0. Messages in S.W.I.F.T. format do not contain control fields.

correspondent. An institution to which your institution sends and from which it receives messages.

correspondent identifier. The 11-character identifier of the receiver of a telex message. Used as a key to retrieve information from the Telex correspondents file.

cross-system coupling facility. See *XCF*.

coupling services. In a sysplex, the functions of XCF that transfer data and status information among the members of a group that reside in one or more of the MVS systems in the sysplex.

couple data set. See *XCF couple data set*.

CTP. MERVA Link command transfer processor.

currency code file. A file containing the currency codes, together with the name, fraction length, country code, and country names.

D

daemon. A long-lived process that runs unattended to perform continuous or periodic systemwide functions.

DASD. Direct access storage device.

data area. An area of a predefined length and format on a panel in which data can be entered or displayed. A field can consist of one or more data areas.

data element. A unit of data that, in a certain context, is considered indivisible. In MERVA Link, a data

element consists of a 2-byte data element length field, a 2-byte data-element identifier field, and a field of variable length containing the data element data.

datagram. In TCP/IP, the basic unit of information passed across the Internet environment. This type of message does not require a reply, and is the simplest type of message that MQSeries supports.

data terminal equipment. That part of a data station that serves as a data source, data link, or both, and provides for the data communication control function according to protocols.

DB2. A family of IBM licensed programs for relational database management.

dead-letter queue. A queue to which a queue manager or application sends messages that it cannot deliver. Also called *undelivered-message queue*.

dial-up number. A series of digits required to establish a connection with a remote correspondent via the public telex network.

direct service. In MERVA, a service that uses resources that are always available and that can be used by several requesters at the same time.

display mode. The mode (PROMPT or NOPROMPT) in which S.W.I.F.T. messages are displayed. See *PROMPT mode* and *NOPROMPT mode.*

distributed queue management (DQM). In MQSeries message queuing, the setup and control of message channels to queue managers on other systems.

DQM. Distributed queue management.

DTE. Data terminal equipment.

E

EBCDIC. Extended Binary Coded Decimal Interchange Code. A coded character set consisting of 8-bit coded characters.

ECB. Event control block.

EDIFACT. Electronic Data Interchange for Administration, Commerce and Transport (a United Nations standard).

ESM. External security manager.

EUD. End-user driver.

exception report. An MQSeries report message type that is created by a message channel agent when a message is sent to another queue manager, but that message cannot be delivered to the specified destination queue.

external line format (ELF) messages. Messages that are not fully tokenized, but are stored in a single field in the TOF. Storing messages in ELF improves performance, because no mapping is needed, and checking is not performed.

external security manager (ESM). A security product that is invoked by the System Authorization Facility. RACF is an example of an ESM.

F

FDT. Field definition table.

field. In MERVA, a portion of a message used to enter or display a particular type of data in a predefined format. A field is located by its position in a message and by its tag. A field is made up of one or more data areas. See also *data area*.

field definition table (FDT). The field definition table describes the characteristics of a field; for example, its length and number of its data areas, and whether it is mandatory. If the characteristics of a field change depending on its use in a particular message, the definition of the field in the FDT can be overridden by the MCB specifications.

field group. One or several fields that are defined as being a group. Because a field can occur more than once in a message, field groups are used to distinguish them. A name can be assigned to the field group during message definition.

field group number. In the TOF, a number is assigned to each field group in a message in ascending order from 1 to 255. A particular field group can be accessed using its field group number.

field tag. A character string used by MERVA to identify a field in a network buffer. For example, for S.W.I.F.T. field 30, the field tag is **:30:**.

FIN. Financial application.

FIN-Copy. The MERVA component used for S.W.I.F.T. FIN-Copy support.

finite state machine. The theoretical base describing the rules of a service request's state and the conditions to state transitions.

FMT/ESA. MERVA-to-MERVA Financial Message Transfer/ESA.

form. A partially-filled message containing data that can be copied for a new message of the same message type.

G

GPA. General purpose application.

H

HFS. Hierarchical file system.

hierarchical file system (HFS). A system for organizing files in a hierarchy, as in a UNIX system. OS/390 UNIX System Services files are organized in an HFS. All files are members of a directory, and each directory is in turn a member of a directory at a higher level in the HFS. The highest level in the hierarchy is the root directory.

I

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

IM-ASPDU. Interapplication messaging application support protocol data unit. It contains an application message and consists of a heading and a body.

incore request queue. Another name for the request queue to emphasize that the request queue is held in memory instead of on a DASD.

InetD. Internet Daemon. It provides TCP/IP communication services in the OS/390 USS environment.

initiation queue. In MQSeries, a local queue on which the queue manager puts trigger messages.

input message. A message that is input into the S.W.I.F.T. network. An input message has an input header.

INTERCOPE TelexBox. This telex box supports various national conventions for telex procedures and protocols.

interservice communication. In MERVA ESA, a facility that enables communication among services if MERVA ESA is running in a multisystem environment.

intertask communication. A facility that enables application programs to communicate with the MERVA nucleus and so request a central service.

IP. Internet Protocol.

IP message. In-process message. A message that is in the process of being transferred to another application.

ISC. Intersystem communication.

ISN. Input sequence number.

ISN acknowledgment. A collective term for the various kinds of acknowledgments sent by the S.W.I.F.T. network.

ISO. International Organization for Standardization.

ITC. Intertask communication.

J

JCL. Job control language.

journal. A chronological list of records detailing MERVA actions.

journal key. A key used to identify a record in the journal.

journal service. A MERVA central service that maintains the journal.

K

KB. Kilobyte (1024 bytes).

key. A character or set of characters used to identify an item or group of items. For example, the user ID is the key to identify a user file record.

key-sequenced data set (KSDS). A VSAM data set whose records are loaded in key sequence and controlled by an index.

keyword parameter. A parameter that consists of a keyword, followed by one or more values.

KSDS. Key-sequenced data set.

L

LAK. Login acknowledgment message. This message informs you that you have successfully logged in to the S.W.I.F.T. network.

large message. A message that is stored in the large message cluster (LMC). The maximum length of a message to be stored in the VSAM QDS is 31900 bytes. Messages up to 2 MB can be stored in the LMC. For queue management using DB2 no distinction is made between messages and large messages.

large queue element. A queue element that is larger than the smaller of:

- The limiting value specified during the customization of MERVA
- 32 KB

LC message. Last confirmed control message. It contains the message-sequence number of the application or acknowledgment message that was last confirmed; that is, for which the sending MERVA Link system most recently received confirmation of a successful delivery.

LDS. Logical data stream.

LMC. Large message cluster.

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

local queue. In MQSeries, a queue that belongs to a local queue manager. A local queue can contain a list of messages waiting to be processed. Contrast with *remote queue*.

local queue manager. In MQSeries, the queue manager to which the program is connected, and that provides message queuing services to that program. Queue managers to which a program is not connected are remote queue managers, even if they are running on the same system as the program.

login. To start the connection to the S.W.I.F.T. network.

LR message. Last received control message, which contains the message-sequence number of the application or acknowledgment message that was last received from the partner application.

LSN. Login sequence number.

LT. See *LTERM*.

LTC. Logical terminal control.

LTERM. Logical terminal. Logical terminal names have 4 characters in CICS and up to 8 characters in IMS.

LU. A VTAM logical unit.

M

maintain system history program (MSHP). A program used for automating and controlling various installation, tailoring, and service activities for a VSE system.

MCA. Message channel agent.

MCB. Message control block.

MERVA ESA. The IBM licensed program Message Entry and Routing with Interfaces to Various Applications for ESA.

MERVA Link. A MERVA component that can be used to interconnect several MERVA systems.

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

message channel. In MQSeries distributed message queuing, a mechanism for moving messages from one queue manager to another. A message channel comprises two message channel agents (a sender and a receiver) and a communication link.

message channel agent (MCA). In MQSeries, a program that transmits prepared messages from a transmission queue to a communication link, or from a communication link to a destination queue.

message control block (MCB). The definition of a message, screen panel, net format, or printer layout made during customization of MERVA.

Message Format Service (MFS). A MERVA direct service that formats a message according to the medium to be used, and checks it for formal correctness.

Message Integrity Protocol (MIP). In MERVA Link, the protocol that controls the exchange of messages between partner ASPs. This protocol ensures that any loss of a message is detected and reported, and that no message is duplicated despite system failures at any point during the transfer process.

message-processing function. The various parts of MERVA used to handle a step in the message-processing route, together with any necessary equipment.

message queue. See *queue*.

Message Queue Interface (MQI). The programming interface provided by the MQSeries queue managers. It provides a set of calls that let application programs access message queuing services such as sending messages, receiving messages, and manipulating MQSeries objects.

Message Queue Manager (MQM). An IBM licensed program that provides message queuing services. It is part of the MQSeries set of products.

message type (MT). A number, 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.

MFS. Message Format Service.

MIP. Message Integrity Protocol.

MPDU. Message protocol data unit, which is defined in P1.

MPP. In IMS, message-processing program.

MQH. MQSeries queue handler.

MQI. Message queue interface.

MQM. Message queue manager.

MQS. MQSeries nucleus server.

MQSeries. A family of IBM licensed programs that provides message queuing services.

MQSeries nucleus server (MQS). A MERVA component that listens for messages on an MQI queue, receives them, extracts a service request, and passes it via the request queue handler to another MERVA ESA instance for processing.

MQSeries queue handler (MQH). A MERVA component that performs service calls to the Message Queue Manager via the provided Message Queue Interface.

MSC. MERVA system control facility.

MSHP. Maintain system history program.

MSN. Message sequence number.

MT. Message type.

MTP. (1) Message transfer program. (2) Message transfer process.

MTS. Message Transfer System.

MTSP. Message Transfer Service Processor.

MTT. Message type table.

multisystem application. (1) An application program that has various functions distributed across MVS systems in a multisystem environment. (2) In XCF, an authorized application that uses XCF coupling services. (3) In MERVA ESA, multiple instances of MERVA ESA that are distributed among different MVS systems in a multisystem environment.

multisystem environment. An environment in which two or more MVS systems reside on one or more processors, and programs on one system can communicate with programs on the other systems. With XCF, the environment in which XCF services are available in a defined sysplex.

multisystem sysplex. A sysplex in which one or more MVS systems can be initialized as part of the sysplex. In a multisystem sysplex, XCF provides coupling services on all systems in the sysplex and requires an XCF couple data set that is shared by all systems. See also *single-system sysplex*.

MVS/ESA. Multiple Virtual Storage/Enterprise Systems Architecture.

N

namelist. An MQSeries for MVS/ESA object that contains a list of queue names.

nested message. A message that is composed of one or more message types.

nested message type. A message type that is contained in another message type. In some cases, only part of a message type (for example, only the mandatory fields) is nested, but this "partial" nested message type is also considered to be nested. For example, S.W.I.F.T. MT 195 could be used to request information about a S.W.I.F.T. MT 100 (customer transfer). The S.W.I.F.T. MT 100 (or at least its mandatory fields) is then nested in S.W.I.F.T. MT 195.

nesting identifier. An identifier (a number from 2 to 255) that is used to access a nested message type.

network identifier. A single character that is placed before a message type to indicate which network is to be used to send the message; for example, **S** for S.W.I.F.T.

network service access point (NSAP). The endpoint of a network connection used by the S.W.I.F.T. transport layer.

NOPROMPT mode. One of two ways to display a message panel. NOPROMPT mode is only intended for experienced SWIFT Link users who are familiar with the structure of S.W.I.F.T. messages. With NOPROMPT mode, only the S.W.I.F.T. header, trailer, and pre-filled fields and their tags are displayed. Contrast with *PROMPT mode*.

NSAP. Network service access point.

nucleus server. A MERVA component that processes a service request as selected by the request queue handler. The service a nucleus server provides and the way it provides it is defined in the nucleus server table (DSLNSVT).

O

object. In MQSeries, objects define the properties of queue managers, queues, process definitions, and namelists.

occurrence. See *repeatable sequence*.

option. One or more characters added to a S.W.I.F.T. field number to distinguish among different layouts for and meanings of the same field. For example, S.W.I.F.T. field 60 can have an option F to identify a first opening balance, or M for an intermediate opening balance.

origin identifier (origin ID). A 34-byte field of the MERVA user file record. It indicates, in a MERVA and SWIFT Link installation that is shared by several banks, to which of these banks the user belongs. This lets the user work for that bank only.

OSN. Output sequence number.

OSN acknowledgment. A collective term for the various kinds of acknowledgments sent to the S.W.I.F.T. network.

output message. A message that has been received from the S.W.I.F.T. network. An output message has an output header.

P

P1. In MERVA Link, a peer-to-peer protocol used by cooperating message transfer processes (MTPs).

P2. In MERVA Link, a peer-to-peer protocol used by cooperating application support processes (ASPs).

P3. In MERVA Link, a peer-to-peer protocol used by cooperating command transfer processors (CTPs).

packet switched public data network (PSPDN). A public data network established and operated by network common carriers or telecommunication administrations for providing packet-switched data transmission.

panel. A formatted display on a display terminal. Each page of a message is displayed on a separate panel.

parallel processing. The simultaneous processing of units of work by several servers. The units of work can be either transactions or subdivisions of larger units of work.

parallel sysplex. A sysplex that uses one or more coupling facilities.

partner table (PT). In MERVA Link, the table that defines how messages are processed. It consists of a header and different entries, such as entries to specify the message-processing parameters of an ASP or MTP.

PCT. Program Control Table (of CICS).

PDE. Possible duplicate emission.

PDU. Protocol data unit.

PF key. Program-function key.

positional parameter. A parameter that must appear in a specified location relative to other parameters.

PREMIUM. The MERVA component used for S.W.I.F.T. PREMIUM support.

process definition object. An MQSeries object that contains the definition of an MQSeries application. A queue manager uses the definitions contained in a process definition object when it works with trigger messages.

program-function key. A key on a display terminal keyboard to which a function (for example, a command) can be assigned. This lets you execute the function (enter the command) with a single keystroke.

PROMPT mode. One of two ways to display a message panel. PROMPT mode is intended for SWIFT Link users who are unfamiliar with the structure of S.W.I.F.T. messages. With PROMPT mode, all the fields and tags are displayed for the S.W.I.F.T. message. Contrast with *NOPROMPT mode*.

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

PSN. Public switched network.

PSPDN. Packet switched public data network.

PSTN. Public switched telephone network.

PT. Partner table.

PTT. A national post and telecommunication authority (post, telegraph, telephone).

Q

QDS. Queue data set.

QSN. Queue sequence number.

queue. (1) In MERVA, a logical subdivision of the MERVA queue data set used to store the messages associated with a MERVA message-processing function. A queue has the same name as the message-processing function with which it is associated. (2) In MQSeries, an object onto which message queuing applications can put messages, and from which they can get messages. A queue is owned and maintained by a queue manager. See also *request queue*.

queue element. A message and its related control information stored in a data record in the MERVA ESA Queue Data Set.

queue management. A MERVA service function that handles the storing of messages in, and the retrieval of messages from, the queues of message-processing functions.

queue manager. (1) An MQSeries system program that provides queuing services to applications. It provides an application programming interface so that programs can access messages on the queues that the queue manager owns. See also *local queue manager* and *remote queue manager*. (2) The MQSeries object that defines the attributes of a particular queue manager.

queue sequence number (QSN). A sequence number that is assigned to the messages stored in a logical queue by MERVA ESA queue management in ascending order. The QSN is always unique in a queue.

It is reset to zero when the queue data set is formatted, or when a queue management restart is carried out and the queue is empty.

R

RACF. Resource Access Control Facility.

RBA. Relative byte address.

RC message. Recovered message; that is, an IP message that was copied from the control queue of an inoperable or closed ASP via the **recover** command.

ready queue. A MERVA queue used by SWIFT Link to collect S.W.I.F.T. messages that are ready for sending to the S.W.I.F.T. network.

remote queue. In MQSeries, a queue that belongs to a remote queue manager. Programs can put messages on remote queues, but they cannot get messages from remote queues. Contrast with *local queue*.

remote queue manager. In MQSeries, a queue manager is remote to a program if it is not the queue manager to which the program is connected.

repeatable sequence. A field or a group of fields that is contained more than once in a message. For example, if the S.W.I.F.T. fields 20, 32, and 72 form a sequence, and if this sequence can be repeated up to 10 times in a message, each sequence of the fields 20, 32, and 72 would be an occurrence of the repeatable sequence.

In the TOF, the occurrences of a repeatable sequence are numbered in ascending order from 1 to 32767 and can be referred to using the occurrence number.

A repeatable sequence in a message may itself contain another repeatable sequence. To identify an occurrence within such a nested repeatable sequence, more than one occurrence number is necessary.

reply message. In MQSeries, a type of message used for replies to request messages.

reply-to queue. In MQSeries, the name of a queue to which the program that issued an MQPUT call wants a reply message or report message sent.

report message. In MQSeries, a type of message that gives information about another message. A report message usually indicates that the original message cannot be processed for some reason.

request message. In MQSeries, a type of message used for requesting a reply from another program.

request queue. The queue in which a service request is stored. It resides in main storage and consists of a set of request queue elements that are chained in different queues:

- Requests waiting to be processed
- Requests currently being processed
- Requests for which processing has finished

request queue handler (RQH). A MERVA ESA component that handles the queueing and scheduling of service requests. It controls the request processing of a nucleus server according to rules defined in the finite state machine.

Resource Access Control Facility (RACF). An IBM licensed program that provides for access control by identifying and verifying users to the system, authorizing access to protected resources, logging detected unauthorized attempts to enter the system, and logging detected accesses to protected resources.

retype verification. See *verification*.

routing. In MERVA, the passing of messages from one stage in a predefined processing path to the next stage.

RP. Regional processor.

RQH. Request queue handler.

RRDS. Relative record data set.

S

SAF. System Authorization Facility.

SCS. SNA character string.

SCP. System control process.

SDI. Sequential data set input. A batch utility used to import messages from a sequential data set or a tape into MERVA ESA queues.

SDO. Sequential data set output. A batch utility used to export messages from a MERVA ESA queue to a sequential data set or a tape.

SDY. Sequential data set system printer. A batch utility used to print messages from a MERVA ESA queue.

service request. A type of request that is created and passed to the request queue handler whenever a nucleus server requires a service that is not currently available.

sequence number. A number assigned to each message exchanged between two nodes. The number is increased by one for each successive message. It starts from zero each time a new session is established.

sign off. To end a session with MERVA.

sign on. To start a session with MERVA.

single-system sysplex. A sysplex in which only one MVS system can be initialized as part of the sysplex. In

a single-system sysplex, XCF provides XCF services on the system, but does not provide signaling services between MVS systems. A single-system sysplex requires an XCF couple data set. See also *multisystem sysplex*.

small queue element. A queue element that is smaller than the smaller of:

- The limiting value specified during the customization of MERVA
- 32 KB

SMP/E. System Modification Program Extended.

SN. Session number.

SNA. Systems Network Architecture.

SNA character string. In SNA, a character string composed of EBCDIC controls, optionally mixed with user data, that is carried within a request or response unit.

SPA. Scratch pad area.

SQL. Structured Query Language.

SR-ASPDU. The status report application support PDU, which is used by MERVA Link for acknowledgment messages.

SSN. Select sequence number.

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

SVC. (1) Switched Virtual Circuit. (2) Supervisor call instruction.

S.W.I.F.T. (1) Society for Worldwide Interbank Financial Telecommunication s.c. (2) The network provided and managed by the Society for Worldwide Interbank Financial Telecommunication s.c.

S.W.I.F.T. address. Synonym for *bank identifier code*.

S.W.I.F.T. Correspondents File. The file containing the bank identifier code (BIC), together with the name, postal address, and zip code of each financial institution in the BIC Directory.

S.W.I.F.T. financial message. A message in one of 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 with an input header to be sent to the S.W.I.F.T. network.

SWIFT link. The MERVA ESA component used to link to the S.W.I.F.T. network.

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

S.W.I.F.T. output message. A S.W.I.F.T. message with an output header coming from the S.W.I.F.T. network.

S.W.I.F.T. system message. A S.W.I.F.T. general purpose application (GPA) message or a financial application (FIN) message in S.W.I.F.T. category 0.

switched virtual circuit (SVC). An X.25 circuit that is dynamically established when needed. It is the X.25 equivalent of a switched line.

sysplex. One or more MVS systems that communicate and cooperate via special multisystem hardware components and software services.

System Authorization Facility (SAF). An MVS or VSE facility through which MERVA ESA communicates with an external security manager such as RACF (for MVS) or the basic security manager (for VSE).

System Control Process (SCP). A MERVA Link component that handles the transfer of MERVA ESA commands to a partner MERVA ESA system, and the receipt of the command response. It is associated with a system control process entry in the partner table.

System Modification Program Extended (SMP/E). A licensed program used to install software and software changes on MVS systems.

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

T

tag. A field identifier.

TCP/IP. Transmission Control Protocol/Internet Protocol.

Telex Correspondents File. A file that stores data about correspondents. When the user enters the corresponding nickname in a Telex message, the corresponding information in this file is automatically retrieved and entered into the Telex header area.

telex header area. The first part of the telex message. It contains control information for the telex network.

telex interface program (TXIP). A program that runs on a Telex front-end computer and provides a communication facility to connect MERVA ESA with the Telex network.

Telex Link. The MERVA ESA component used to link to the public telex network via a Telex substation.

Telex substation. A unit comprised of the following:

- Telex Interface Program
- A Telex front-end computer
- A Telex box

Terminal User Control Block (TUCB). A control block containing terminal-specific and user-specific information used for processing messages for display devices such as screen and printers.

test key. A key added to a telex message to ensure message integrity and authorized delivery. The test key is an integer value of up to 16 digits, calculated manually or by a test-key processing program using the significant information in the message, such as amounts, currency codes, and the message date.

test-key processing program. A program that automatically calculates and verifies a test key. The Telex Link supports panels for input of test-key-related data and an interface for a test-key processing program.

TFD. Terminal feature definitions table.

TID. Terminal identification. The first 9 characters of a bank identifier code (BIC).

TOF. Originally the abbreviation of *tokenized form*, the TOF is a storage area where messages are stored so that their fields can be accessed directly by their field names and other index information.

TP. Transaction program.

transaction. A specific set of input data that triggers the running of a specific process or job; for example, a message destined for an application program.

transaction code. In IMS and CICS, an alphanumeric code that calls an IMS message processing program or a CICS transaction. Transaction codes have 4 characters in CICS and up to 8 characters in IMS.

Transmission Control Protocol/Internet Protocol (TCP/IP). A set of communication protocols that support peer-to-peer connectivity functions for both local and wide area networks.

transmission queue. In MQSeries, a local queue on which prepared messages destined for a remote queue manager are temporarily stored.

trigger event. In MQSeries, an event (such as a message arriving on a queue) that causes a queue manager to create a trigger message on an initiation queue.

trigger message. In MQSeries, a message that contains information about the program that a trigger monitor is to start.

trigger monitor. In MQSeries, a continuously-running application that serves one or more initiation queues. When a trigger message arrives on an initiation queue, the trigger monitor retrieves the message. It uses the information in the trigger message to start a process that serves the queue on which a trigger event occurred.

triggering. In MQSeries, a facility that allows a queue manager to start an application automatically when predetermined conditions are satisfied.

TUCB. Terminal User Control Block.

TXIP. Telex interface program.

U

UMR. Unique message reference.

unique message reference (UMR). An optional feature of MERVA ESA that provides each message with a unique identifier the first time it is placed in a queue. It is composed of a MERVA ESA installation name, a sequence number, and a date and time stamp.

UNIT. A group of related literals or fields of an MCB definition, or both, enclosed by a DSLLUNIT and DSLLUEND macroinstruction.

UNIX System Services (USS). A component of OS/390, formerly called OpenEdition (OE), that creates a UNIX environment that conforms to the XPG4 UNIX 1995 specifications, and provides two open system interfaces on the OS/390 operating system:

- An application program interface (API)
- An interactive shell interface

UN/EDIFACT. United Nations Standard for Electronic Data Interchange for Administration, Commerce, and Transport.

USE. S.W.I.F.T. User Security Enhancements.

user file. A file containing information about all MERVA ESA users; for example, which functions each user is allowed to access. The user file is encrypted and can only be accessed by authorized persons.

user identification and verification. The acts of identifying and verifying a RACF-defined user to the system during logon or batch job processing. RACF identifies the user by the user ID and verifies the user by the password or operator identification card supplied during logon processing or the password supplied on a batch JOB statement.

USS. UNIX System Services.

V

verification. Checking to ensure that the contents of a message are correct. Two kinds of verification are:

- Visual verification, in which you read the message and confirm that you have done so
- Retype verification, in which you reenter the data to be verified

Virtual LU. An LU defined in MERVA Extended Connectivity for communication between MERVA and MERVA Extended Connectivity.

Virtual Storage Access Method (VSAM). An access method for direct or sequential processing of fixed and variable-length records on direct access devices. The records in a VSAM data set or file can be organized in logical sequence by a key field (key sequence), in the physical sequence in which they are written on the data set or file (entry sequence), or by relative-record number.

Virtual Telecommunications Access Method (VTAM). An IBM licensed program that controls communication and the flow of data in an SNA network. It provides single-domain, multiple-domain, and interconnected network capability.

VSAM. Virtual Storage Access Method.

VTAM. Virtual Telecommunications Access Method (IBM licensed program).

X

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

XCF. Abbreviation for *cross-system coupling facility*, which is a special logical partition that provides high-speed caching, list processing, and locking functions in a sysplex. XCF provides the MVS coupling services that allow authorized programs on MVS systems in a multisystem environment to communicate with (send data to and receive data from) authorized programs on other MVS systems.

XCF couple data set. A data set that is created through the XCF couple data set format utility and, depending on its designated type, is shared by some or all of the MVS systems in a sysplex. It is accessed only by XCF and contains XCF-related data about the sysplex, systems, applications, groups, and members.

XCF group. The set of related members defined to SCF by a multisystem application in which members of the group can communicate with (send data to and receive data from) other members of the same group. All MERVA systems working together in a sysplex must pertain to the same XCF group.

XCF member. A specific function of a multisystem application that is defined to XCF and assigned to a group by the multisystem application. A member resides on one system in a sysplex and can use XCF services to communicate with other members of the same group.

Bibliography

MERVA ESA Publications

- *MERVA for ESA Version 4 Application Programming Interface Guide*, SH12-6374
- *MERVA for ESA Version 4 Advanced MERVA Link*, SH12-6390
- *MERVA for ESA Version 4 Concepts and Components*, SH12-6381
- *MERVA for ESA Version 4 Customization Guide*, SH12-6380
- *MERVA for ESA Version 4 Diagnosis Guide*, SH12-6382
- *MERVA for ESA Version 4 Installation Guide*, SH12-6378
- *MERVA for ESA Version 4 Licensed Program Specifications*, GH12-6373
- *MERVA for ESA Version 4 Macro Reference*, SH12-6377
- *MERVA for ESA Version 4 Messages and Codes*, SH12-6379
- *MERVA for ESA Version 4 Operations Guide*, SH12-6375
- *MERVA for ESA Version 4 System Programming Guide*, SH12-6366
- *MERVA for ESA Version 4 User's Guide*, SH12-6376

Other MERVA Publications

- *MERVA Directory Services*, SH12-6367
- *MERVA Extended Connectivity Installation and User's Guide*, SH12-6157
- *MERVA Extended Connectivity Licensed Program Specifications*, GH12-6186
- *MERVA Message Processing Client for Windows NT User's Guide*, SH12-6341
- *MERVA Traffic Reconciliation*, SH12-6392
- *MERVA USE Administration Guide*, SH12-6338
- *MERVA USE & Branch for Windows NT Installation and Customization Guide*, SH12-6335
- *MERVA Workstation Based Functions*, SH12-6383

S.W.I.F.T. Publications

The following 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. FIN Security Guide*
- *S.W.I.F.T. Card Readers User Guide*

Index

MERVA Requirement Request

Use the form overleaf to send us requirement requests for the MERVA product. Fill in the blank lines with the information that we need to evaluate and implement your request. Provide also information about your hardware and software environments and about the MERVA release levels used in your environment.

Provide a detailed description of your requirement. If you are requesting a new function, describe in full what you want that function to do. If you are requesting that a function be changed, briefly describe how the function works currently, followed by how you are requesting that it should work.

If you are a customer, provide us with the appropriate contacts in your organization to discuss the proposal and possible implementation alternatives.

If you are an IBM employee, include at least the name of one customer who has this requirement. Add the name and telephone number of the appropriate contacts in the customer's organization to discuss the proposal and possible implementation alternatives. If possible, send this requirement online to MERVAREQ at SDFVM1.

For comments on this book, use the form provided at the back of this publication.

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.

Send the fax to:

To: MERVA Development, Dept. 5640
Attention: Gerhard Stubbe

IBM Deutschland Entwicklung GmbH
Schoenaicher Str. 220
D-71032 Boeblingen
Germany

Fax Number: +49-7031-16-4881
Internet address:
mervareq@de.ibm.com

MERVA Requirement Request

To: MERVA Development, Dept. 5640
Attention: Gerhard Strubbe

IBM Deutschland Entwicklung GmbH
Schoenaicher Str. 220
D-71032 Boeblingen Germany

Fax Number: +49-7031-16-4881
Internet address:
mervareq@de.ibm.com

Page 1 of _____

Customer's Name _____

Customer's Address _____

_____

_____

Customer's
Telephone/Fax _____

Contact Person at
Customer's Location _____
Telephone/Fax _____

MERVA
Version/Release _____

Operating System _____
Sub-System
Version/Release _____

Hardware _____

Requirement
Description _____

_____

_____

_____

_____

_____

_____

_____

Expected Benefits _____

_____

_____

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

MERVA ESA Components
Directory Services

Publication No. SH12-6367-00

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
SH12-6367-00

Cut or Fold
Along Line

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

PLACE
POSTAGE
STAMP
HERE

IBM Deutschland Entwicklung GmbH
Information Development, Dept. 0446
Schoenaicher Strasse 220
71032 Boeblingen
Germany

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

SH12-6367-00

Cut or Fold
Along Line

Program Number: 5648-B30

Printed in Denmark by IBM Danmark A/S

SH12-6367-00

Spine information:

MERVA ESA Components

Directory Services

Version 4
Release 1