MERVA for ESA

Customization Guide

Version 4 Release 1

SH12-6380-01

MERVA for ESA

Customization Guide

Version 4 Release 1

SH12-6380-01

Note!
Before using this information and the product it supports, be sure to read the general information under "Appendix. Notices" on page 395.

Second Edition, May 2001

This edition applies to Version 4 Release 1 of IBM MERVA for ESA, Program Number 5648-B29 and to all subsequent releases and modifications until otherwise indicated in new editions.

Changes to this edition are marked with a vertical bar.

Contents

About This Book

This book helps the system programmer to customize the installed 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). MERVA ESA provides the following components that can be installed and customized as required:

- Base Functions
- SWIFT Link
- Telex Link
- MERVA Link
- FMT/ESA with MERVA Link

In this book, system programmers find detailed technical information with several examples showing coding methods.

MERVA ESA and its communication components are designed so that you can customize them to meet the requirements of your installation.

MERVA ESA and its components use tables to contain the definitions of the environment in which MERVA ESA runs. By specifying the tables or table parameters you can adapt the environment for MERVA ESA to the requirements of your installation.

MERVA ESA delivers examples for all tables used in the system. Before you start customizing, make sure that the correct MERVA ESA installation is verified.

Adapting DSLPRM and DSLTFDT is mandatory, all other tables can be customized optionally. You might want to customize message-processing functions and routing.

This book deals with the functions and items you are most likely to customize first, and continues with the less likely items later.

Prerequisites for Using This Book

You should be familiar with the parameters of the MERVA ESA macros described in the *MERVA for ESA Macro Reference*.

You should also be familiar with *MERVA for ESA Concepts and Components* which describes the function, services, and utilities supplied. It is for readers who want a general idea of the message concept, queues, routing, message handling, and the network links.

For the SWIFT Link it is assumed that you are familiar with the contents of the *S.W.I.F.T. User Handbook*, published by the Society for Worldwide Interbank Financial Telecommunication s.c. (S.W.I.F.T.). For the Telex Link you should be familiar with the telex terminology as defined by your local PTT[1].

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

Note: The term *CICS* is used to refer to CICS/ESA®, CICS Transaction Server (CICS TS), and CICS/VSE®. The term *IMS* is used to refer to IMS/ESA®.

Customizing Overview

MERVA ESA uses tables to contain the definitions of the environment in which MERVA ESA runs. By specifying the tables or table parameters you can adapt the environment for MERVA ESA to the requirements of your installation. The names of the tables are supplied as parameters in the MERVA ESA customizing parameter modules: DSLPRM, DWSPRM, and ENLPRM. Sample tables for MERVA ESA are provided with the machine-readable material.

The following list provides an overview of the ways in which you can customize your MERVA ESA installation:

- Define the message-processing functions

  You can define the functions used to process the messages. The functions are defined in the Function Table (DSLFNTT). All the functions available to the users of MERVA ESA are defined in this table. Details of how to customize the Function Table are given in "Defining the MERVA ESA Message-Processing Functions" on page 3.
- Define the path of a message

  You define the path of a message in the Function Table (DSLFNTT) by specifying one or both of the following:
  - The next message-processing function directly
  - The routing table that makes the message path dependent on the contents of the message

  Details of how to customize the Routing Tables are given in "Defining Message Paths within MERVA ESA" on page 54
- Assign commands to the PF keys

  The PF-Key Tables (DSLMPFxx) are used to assign the PF keys to commands for each panel type. Details of how to define these tables can be found in "Assigning the Program Function (PF) Keys" on page 77.
- Define files

  The files you access when requesting the general file services must be defined in the General File Table (DSLFLTT). Details of how to code the file table can be found in "Defining Files in MERVA ESA" on page 113.
- Define the format of the panels and printer pages

  You use the Terminal Features Definition Table (DSLTFDT) to define the format used for printing and displaying panels. Details of how to use the Terminal Features Definition Table are given in "Terminal Feature Definition Macro (DSLTFD)" on page 121.
- Define the parameters for MERVA ESA

  The basic customizing of MERVA ESA is carried out in the modules DSLPRM, DWSPRM, and ENLPRM (which use the macros DSLPARM, DWSPARM, and ENLPARM), and in the table DSLNSVT (which uses the macro DSLNSV). The use of these macros is described in "Chapter 2. The MERVA ESA Environment" on page 87, "Chapter 3. The SWIFT Link" on page 127, and "Chapter 5. The Telex Link" on page 149.
- Define lines to the SWIFT network

For the SWIFT network, data communication lines must be defined; the communication to the SWIFT network is via X.25. Details are contained in "Defining Communication Lines to the SWIFT Network" on page 130.

- Define SWIFT logical terminals

  For the SWIFT network, logical terminals must be defined. Details are contained in "Defining Logical Terminals for the SWIFT Network" on page 134.

- Define general test-key requirements

  The Telex Link Test-Key Requirement Table ENLTKRQT defines for specific message types and groups of message types whether messages of these types must be protected by the addition of a test key. Details of how to define the test-key requirements can be found in "Specifying General Test-Key Requirements" on page 150.

- Define the extraction fields for the test-key calculation

  The Extraction fields for the Test-Key Calculation are defined in the Message Control Block (MCB) of the SWIFT Message. Details of how to define the Extraction Fields is contained in "Defining the Extraction Fields for Test-Key Calculation" on page 154.

- Define extended field tags for the telex line format

  Details of how to add extended field tags to a SWIFT message in the telex network format for better readability can be found in "Defining Extended Field Tags for the Telex Line Format" on page 154.

- Define the MERVA Link partner systems

  Details of how to define the MERVA Link parameters can be found in "Defining Partner Table ASP and MTP Entries (Samples)" on page 161.

- Add your programs (filters and user exits) to the MERVA Link

  Details of how to add your programs to the MERVA Link are contained in "Application Support Filter" on page 195.

- Define the MERVA ESA USS partner systems

  Details of how to define the MERVA ESA USS parameters can be found in "Chapter 7. The MERVA Link for Unix System Services (USS)" on page 243.

- Define the parameters for FMT/ESA with MERVA Link

  Details of the set of FMT/ESA parameters and how to activate them can be found in "Customizing FMT/ESA with MERVA Link" on page 285.

- Define the parameters for MERVA-MQI Attachment

  Details of the set of MERVA-MQI Attachment parameters and how to activate them are contained in "Chapter 9. MERVA-MQI Attachment" on page 301.

- Define the formats of messages and fields

  The main purpose of MERVA ESA is to process messages for external networks, display terminals, and printers. MERVA ESA uses an internal format called a "tokenized form" (TOF) to format the messages for each of these external devices. The TOF services let you access and change the contents of any field of a message directly in the TOF using a symbolic name, and transform a message to an external format according to the corresponding MCB specifications. The fields and messages are defined using MERVA ESA macros and MCBs. All the MCBs are made available to MERVA ESA as entries in the message type table (DSLMTTT). Details of how to define field and message structures can be found in "Part 2. Defining Fields and Messages" on page 343.

Summary of Changes

This edition reflects the following changes:

MERVA-MQI Attachment for VSE
MQSeries for VSE/ESA now fully exploits COA and COD report message handling. Therefore, the restrictions concerning the COA and COD reports and the usage of the PASCMID parameter of the process table DSLKPROC have been removed from "Chapter 9. MERVA-MQI Attachment" on page 301.

FMT/ESA can now use MERVA-MQI Attachment
Financial Message Transfer/ESA (FMT/ESA) can now use MERVA-MQI Attachment as well as MERVA Link ESA to transfer SWIFT messages between two MERVA ESA systems (see "Using FMT/ESA with MERVA-MQI Attachment" on page 296).

MERVA-MQI Attachment message security
Using proprietary algorithms, MERVA-MQI Attachment can now encrypt, decrypt, and authenticate message data (see "Requesting Message Security" on page 314).

MERVA-MQI Attachment message conversion
The channel message exits DSLKCM1C and DSLKCM1M used for message conversion under MQSeries for MVS/ESA have been removed (see "Converting the Message Data" on page 336).

Part 1. Basic Customizing

Chapter 1. The MERVA ESA Applications

This chapter shows you how to:

- Define the MERVA ESA Message-Processing Functions to be used in your installation
- Define the routes a message will take in your installation
- Assign commands to the PF keys for each of the panels used in your installation.

Defining the MERVA ESA Message-Processing Functions

MERVA ESA supplies a sample of message-processing functions that you can use to create and process messages. The Function Table (DSLFNTT) defines all the functions in a MERVA ESA installation used for message processing, file maintenance, and operator command processing.

The table is coded as a sequence of DSLFNT macros. For a detailed description of the DSLFNT macro refer to the *MERVA for ESA Macro Reference*. When customizing MERVA ESA and associated network links like SWIFT Link, copy members are used to combine the function table definitions of MERVA ESA and its components in one function table. This is done using the DSLGEN macro. Figure 1 shows the source code for the function table.

Note: The function table DSLFNTT can be coded as a sequence of the DSLFNT macros:

- TYPE=INITIAL
- TYPE=ENTRY
- TYPE=FINAL

This coding can be done without using the DSLGEN process, and preferably using a name other than DSLFNTT.

```
FNTT     TITLE 'MERVA ESA SAMPLE FUNCTION TABLE'
DSLFNTT  DSLFNT TYPE=INITIAL                                        [1]
         COPY  DSLFNTTC            MERVA ESA FUNCTION TABLE       [2]
         COPY  DWSFNTTC            MERVA ESA SWIFT LINK           [3]
         COPY  ENLFNTTC            MERVA ESA TELEX LINK           [4]
         COPY  EKAFNTTC            MERVA ESA MERVA LINK           [5]
         COPY  EKAFNTSC            MERVA ESA FMT                  [6]
******** ****                      RESERVED 06                    [7]
******** ****                      RESERVED 07
******** ****                      RESERVED 08
******** ****                      RESERVED 09
******** ****                      RESERVED 10
******** ****                      RESERVED FOR USER
         DSLFNT TYPE=FINAL                                          [8]
         END
```

Figure 1. Coding Example of a MERVA ESA Function Table

Notes:

[1] This must be the first macro. The label DSLFNTT is the name of the function table. If no name is specified, the default name DSLFNTT is used.

However, the name of the table must be specified as the value of the parameter FNT for the macro DSLPARM in the module DSLPRM.

It is possible to work with different function tables by changing the table name specified in this module.

[2] This statement copies the function definitions required by MERVA ESA into the function table, for example, USR for the user file maintenance, or CMD for the operator command processing.

[3] This statement copies the function definitions required by the SWIFT Link into the function table. These are functions for:

- The entry, verification, and authorization of SWIFT input messages
- The display and authorization of SWIFT output messages
- The other functions used to process SWIFT messages
- The online maintenance of the Authenticator-Key File

[4] This statement copies the functions required by the Telex Link into the function table. These functions are for the:

- Entry, verification, and authorization of outgoing telex messages
- Display and distribution of incoming telex messages
- Test-key calculation and verification of telex messages
- Telex processing queues

[5] This statement copies the functions required by the MERVA Link into the function table.

[6] This statement copies the functions required by FMT/ESA with MERVA Link into the function table.

[7] Depending on the installation, the copy statements of other components are shown here.

[8] This must be the last macro and is followed by the assembler END statement.

The functions can be defined in any sequence, but the name used for each function must be unique. To make any reply to a **dq** (display queue) command more readable, you can build function groups by giving each function name within that group a significant prefix or other sequence of characters anywhere in the name.

All function table entries presented in this book are supplied with the machine readable material in the copy codes DSLFNTTC, DWSFNTTC, EKAFNTTC, EKAFNTSC, and ENLFNTTC. The function table entries of DSLFNTTC for general MERVA ESA functions are presented first. These are followed by the entries for the examples of three SWIFT Link master logical terminals with different paths for routing of messages. The prefix L1 is used for all functions in the DWSFNTTC copy code related to the example of the first master logical terminal.

General MERVA ESA Functions

Figure 2 on page 5 shows the table entries for the general MERVA ESA functions for:

- Printing
- Sequential file processing
- Special user functions

```
DSLFNT NAME=DMPR0,QUEUE=YES,PRFORM=(E,0),THRESH=20,           *  [1]
      TRAN=DSLH,LTERM=PRT1,STATUS=HOLD,                       *
      DESCR='MERVA Print Queue'                                  [2]
DSLFNT NAME=DMPR1,QUEUE=YES,PRFORM=(E,1),THRESH=20,           *
      TRAN=DSLH,LTERM=PRT1,STATUS=HOLD,                       *
      DESCR='MERVA Print Queue'
DSLFNT NAME=DMPR2,QUEUE=YES,PRFORM=(E,2),THRESH=20,           *
      TRAN=DSLH,LTERM=PRT1,STATUS=HOLD,                       *
      DESCR='MERVA Print Queue'
DSLFNT NAME=DMPR3,QUEUE=YES,PRFORM=(E,3),THRESH=20,           *
      TRAN=DSLH,LTERM=PRT1,STATUS=HOLD,                       *
      DESCR='MERVA Print Queue'
DSLFNT NAME=DMPR4,QUEUE=YES,PRFORM=(E,4),THRESH=20,           *
      TRAN=DSLH,LTERM=PRT1,STATUS=HOLD,                       *
      DESCR='MERVA Print Queue'
DSLFNT NAME=DMPR89,QUEUE=YES,PRFORM=(E,0),THRESH=20,          *
      TRAN=DSLH,LTERM=PRT1,STATUS=HOLD,                       *
      DESCR='MERVA Print Queue'
DSLFNT NAME=DMSDI,QUEUE=YES,NEXT=DMSO0,                       *  [3]
      DESCR='MERVA Batch Input Queue'
DSLFNT NAME=DMSO0,NEXT=DMERR,QUEUE=YES,                       *
      DESCR='MERVA Batch Output Queue'
DSLFNT NAME=DMSO1,NEXT=DMERR,QUEUE=YES,                       *
      DESCR='MERVA Batch Output Queue'
DSLFNT NAME=DMERR,NEXT=DMERR,QUEUE=YES,SPCMND=(DEL,ROU),      *
      NOPR=YES,                                               *
      DESCR='MERVA Error Queue'
DSLFNT NAME=DMSY0,PRFORM=(E,0),QUEUE=YES,                     *  [4]
      DESCR='MERVA Batch Print Queue'
DSLFNT NAME=DMSY1,PRFORM=(E,1),QUEUE=YES,                     *
      DESCR='MERVA Batch Print Queue'
DSLFNT NAME=DMSY2,PRFORM=(E,2),QUEUE=YES,                     *
      DESCR='MERVA Batch Print Queue'
DSLFNT NAME=DMSY3,PRFORM=(E,3),QUEUE=YES,                     *
      DESCR='MERVA Batch Print Queue'
DSLFNT NAME=DMSY4,PRFORM=(E,4),QUEUE=YES,                     *
      DESCR='MERVA Batch Print Queue'
DSLFNT NAME=DUMMY,QUEUE=DUMMY,                                *  [5]
      DESCR='MERVA Dummy Queue'
DSLFNT NAME=DMTST,QUEUE=YES,THRESH=30,NEXT=DMERR,PROT=YES,    *  [6]
      KEY1=(SW20,16),KEY2=(MSGTRUID,8),SPCMND=(ROU,DEL,OK),   *
      DESCR='MERVA Test Queue'
DSLFNT NAME=CMD,QUEUE=NO,PROGRAM=DSLECMD,                     *  [7]
      FRAME=(0CMD,0BOT),NOPR=NO,PRINT=DMPR0,                  *
      DESCR='Operator Command Processing'
DSLFNT NAME=FLM,QUEUE=NO,PROGRAM=DSLEFLM,                     *  [8]
```

Figure 2. Example of Function Table Entries for MERVA ESA Functions (Part 1 of 2)

```
      FRAME=(0FLM,0FLM),NOPR=NO,PRINT=DMPR0,                  *
      DESCR='General File Maintenance'
DSLFNT NAME=USR,QUEUE=NO,PROGRAM=DSLEUSR,PRFORM=(E,1),        *  [9]
      FRAME=(0USR,0USR),NOPR=NO,PRINT=DMPR0,SPCMND=(OK),      *
      DESCR='User File Maintenance'
DSLFNT NAME=USR0,QUEUE=NO,PROGRAM=DSLEUSR,PRFORM=(E,1),       *
      FRAME=(0USR,0USR),NOPR=NO,PRINT=DMPR0,                  *
      PROT=YES,                                               *
      DESCR='User File Maintenance / Display only'
DSLFNT NAME=USR1,QUEUE=NO,PROGRAM=DSLEUSR,PRFORM=(E,1),       *
      FRAME=(0USR,0USR),NOPR=NO,PRINT=DMPR0,                  *
      DESCR='User File Maintenance / Update'
DSLFNT NAME=USR2,QUEUE=NO,PROGRAM=DSLEUSR,PRFORM=(E,1),       *
      FRAME=(0USR,0USR),NOPR=NO,PRINT=DMPR0,                  *
      PROT=YES,SPCMND=(OK),                                   *
      DESCR='User File Maintenance / Authorization'
```

Figure 2. Example of Function Table Entries for MERVA ESA Functions (Part 2 of 2)

Notes:

[1] Six hard-copy print functions DMPRx (transaction TRAN=DSLH) are defined for all the different compression formats (PRFORM=(E,n); n = 0 to 4).

The print functions are assigned to logical terminal names (LTERM). LTERM specifies the name of a logical terminal as defined in DSLTFDT. Queues are assigned (QUEUE=YES) with a threshold of 20 elements (THRESH=20). The functions are set initially into the hold status (STATUS=HOLD).

[2] The descriptive name is coded for all functions (DESCR parameter). This descriptive name is used in the Function Selection panel when the menu of the functions available to a user is shown.

[3] Four functions (queues) are defined. The first three are used for MERVA ESA sequential file processing of messages, DMSDI for input, DMSO0 and DMSO1 for output. The NEXT parameter specifies where the messages are routed to after processing.

The fourth is the error queue DMERR. It shows that the two panel-commands **delete** (delete the message) and **route** (route the message to a specified function) are allowed. The parameter NOPR=YES specifies that message processing in NOPROMPT mode is allowed.

The DMERR function specifies the NEXT=DMERR parameter, which causes the **eom** command to result in a **requeue** command. This means that the message is written to the end of the DMERR queue.

[4] Five functions DMSYx are defined for system printer functions.

[5] The function DUMMY is defined with the parameter QUEUE=DUMMY to allow for the routing of messages without creating a queue element. Routing to this queue has the same effect as a **delete** command.

[6] The function DMTST defines a test queue with a threshold of 30 queue elements. All fields of the message are protected on the user panel (PROT=YES). Two keys are defined for direct retrieval of the messages from this queue: KEY1 specifies the TOF field SW20 (transaction reference number field of SWIFT messages) with length 16. KEY2 specifies the TOF field MSGTRUID with length 8. MSGTRUID is a subfield of the MSGTRACE field. It contains either of the following:

- The user ID of the user who created the message the first time
- The name of the program that wrote the message to a MERVA ESA queue the first time

The restricted user panel commands **delete**, **ok**, and **route** are allowed. The function DMERR is defined as the NEXT= function.

Note: If the user detects an error in the message during execution of DMTST, this error can be removed only in the NEXT function DMERR because the PROT=YES parameter prevents the change (or correction) of the message on the screen terminal.

[7] The function CMD is defined for entering operator commands on the screen. It is assigned to the MERVA ESA processing program DSLECMD. The top frame 0CMD and bottom frame 0BOT are used to display the screen or print pages. NOPROMPT processing is not allowed as it does not make sense for the MERVA ESA operator command panel.

[8] The function FLM is defined for MERVA ESA general file maintenance and assigned to the MERVA ESA processing program DSLEFLM. The top frame 0FLM and bottom frame 0FLM are used to display the screen or print pages. NOPROMPT processing is not allowed.

[9] Four functions are defined for MERVA ESA user file maintenance and assigned to the MERVA ESA processing program DSLEUSR. The top frame 0USR and bottom frame 0USR are used to display the screen. If the HCO command is used to print the current page, the top and bottom frames that are defined for the print function are used. NOPROMPT processing is not allowed and UNIT compression (PRFORM=(E,1)) is assigned.

The four functions show the four different types of user file maintenance:

USR Has the parameters PROT=NO (default) and SPCMND=OK, allowing the addition, replacement, and deletion of user file records, and authorizing them in the same step.

USR0 Has the parameter PROT=YES but *not* the parameter SPCMND=OK. This only allows for the display of user file records.

USR1 Has the parameter PROT=NO (default) but *not* the parameter SPCMND=OK. This allows for the addition, replacement, and deletion of user file records, but *not* for the authorization of the changes. The user who authorizes these changes must process a function like USR2 or USR as a second step.

USR2 Has the parameter PROT=YES and the parameter SPCMND=OK, allowing the authorization of changes to the file that are made by another user.

Examples of Function Table Entries for the SWIFT Link

The SWIFT Link supplies, in the copy code DWSFNTTC, examples of function table entries for the message processing functions for the SWIFT network, SWIFT USE, and the functions necessary for the online maintenance of the Authenticator-Key File. The message functions and the functions for online maintenance are described in the following section; the functions for SWIFT USE are described in *MERVA Workstation Based Functions*.

Examples of Function Table Entries for Maintaining the Authenticator-Key File

The following example for Authenticator-Key File maintenance apply to any bank regardless of how complicated its organizational structure for processing SWIFT messages is.

```
DSLFNT NAME=AUT,QUEUE=NO,PROGRAM=DWSEAUT,PRFORM=(E,1),        *
      FRAME=(SAUT,SAUT),NOPR=NO,PRINT=L1PR0,SPCMND=(OK),      *
      PFGROUP=24,EXPAND=(UNCOND,UNCOND),                      *
      DESCR='Authenticator-Key File Maintenance'
DSLFNT NAME=AUT0,QUEUE=NO,PROGRAM=DWSEAUT,PRFORM=(E,1),       *
      FRAME=(SAUT,SAUT),NOPR=NO,PRINT=L1PR0,PROT=YES,         *
      PFGROUP=28,EXPAND=(UNCOND,UNCOND),                      *
      DESCR='Authenticator-Key File Maintenance / Display only*
      '
DSLFNT NAME=AUT1,QUEUE=NO,PROGRAM=DWSEAUT,PRFORM=(E,1),       *
      FRAME=(SAUT,SAUT),NOPR=NO,PRINT=L1PR0,                  *
      PFGROUP=32,EXPAND=(UNCOND,UNCOND),                      *
      DESCR='Authenticator-Key File Maintenance / Update'
DSLFNT NAME=AUT2,QUEUE=NO,PROGRAM=DWSEAUT,PRFORM=(E,1),       *
      FRAME=(SAUT,SAUT),NOPR=NO,PRINT=L1PR0,PROT=YES,         *
      SPCMND=(OK),PFGROUP=36,EXPAND=(UNCOND,UNCOND),          *
      DESCR='Authenticator-Key File Maintenance / Authorizatio*
      n'
```

Figure 3. Definition of the Functions for Authenticator-Key File Maintenance

These functions are defined for the Authenticator-Key File maintenance, and are assigned to the SWIFT Link user-processing program DWSEAUT. The top frame SAUT and bottom frame SAUT are used to display the screen or print pages. NOPROMPT processing is not allowed and UNIT compression (PRFORM=(E,1)) is assigned.

The descriptive name is coded with the DESCR parameter. This descriptive name is used in the function selection panel when the menu of the functions available to a user is shown. The PFGROUP parameter assigns the correct program function keys according to the allowed functions. The EXPAND parameter expands the correspondent's SWIFT address to the full address, this requires the SWIFT Correspondents File being available in MERVA ESA, and then gives an indication if this SWIFT address is an existing one.

The four functions show the four different types of Authenticator-Key File maintenance:

- AUT has the parameters PROT=NO (default) and SPCMND=OK allowing the addition, replacement, and deletion of Authenticator-Key File records, and authorization of the changes in the same step. It also allows for the **exchange** command.
- AUT0 has the parameter PROT=YES and *not* the parameter SPCMND=OK. This allows display of Authenticator-Key File records only.
- AUT1 has the parameter PROT=NO (default) and *not* the parameter SPCMND=OK. This allows the addition, replacement, and deletion of Authenticator-Key File records, but *not* for the authorization of these changes. The **exchange** command is *not* allowed. The user who authorizes the changes must process a function like AUT or AUT2.

- AUT2 has the parameters PROT=YES and SPCMND=OK allowing for the authorization of changes made to the file by another user. The **exchange** command is also allowed by the SPCMND=OK parameter.

Examples of Function Table Entries for Processing Messages for the SWIFT Network

The following examples show the functions necessary for the creation and processing of SWIFT messages for the SWIFT network:

- Example 1 is the simplest example.
- Example 2 uses retype verification and more sophisticated routing than Example 1. It is similar to Example 1, but with the following modifications:
  - The functions for the SWIFT-EDIFACT conversion are not available.
  - A forms queue L2FORMS is added to ease the data entry of messages that contain similar texts.
  - A retyping function L2RE0 for the SWIFT field 32 is added.
  - A second authorization L2AI2 for amounts greater than 10000 is added.
- Example 3 has the same functions for processing of SWIFT input messages as Example 2, except that it uses the checking and expansion transaction DSLCXT to receive messages from SWIFT, and it evaluates branch codes and synonym logical terminals during the routing process. The message flow is identical to that of Example 2.

Each example corresponds to a different bank with its own organizational structure. Each structure has a different level of complexity. The examples are also related to the SWIFT Link example of the logical terminal table DWSLTT.

When using the SWIFT User Security Enhancements (USE), these examples all use a group of routing tables that is described in more detail in *MERVA Workstation Based Functions*. The following routing tables give an example of the flow of messages between MERVA ESA, the SWIFT network, and the USE functions of MERVA ESA (sometimes referred to as *USE workstation*):

EKARTTXU Routing of:

- The control queue (TX2USECQ) of the MERVA Link connection between MERVA ESA and the USE workstation
- The send process control queue (USECQS), and receive process control queue (USECQR) of the MERVA-MQI Attachment connection between MERVA ESA and the USE workstation

This includes routing of the messages sent to the USE workstation and confirmed and routing of the messages received from the USE workstation.

DWSRTSK Routing of session key requests for login and select to the USE workstation when the SWIFT Secure Login/Select (SLS) is used, but session key preload is not used

Example 1: This example uses function names that start with the characters L1, and refers to the first master logical terminal (LT) in DWSLTT. It uses the following routing tables:

DWSL1AI0 Routing after authorization input (L1AI0)

DWSL1AO0 Routing after authentication output (L1AO0)

DWSL1DO0 Routing after distribution output (L1DO0)

DWSL1ES Routing after SWIFT-EDIFACT conversion input (L1SDIES and L1CES)

DWSL1SE Routing after EDIFACT-SWIFT conversion output (L1SDOSE and L1CSE)

DWSL1IN Routing of input messages in DWSDGPA (from ready queues)

DWSL1OUT Routing of output messages in DWSDGPA (from SWIFT)

Note: The routing tables DWSL1IN, DWSL1OUT, DWSL2IN, DWSL2OUT, DWSL3GPI, DWSL3GPO, DWSL3FII, DWSL3FIO and DWSRTSK are referred to in the logical terminal table DWSLTT. All other routing tables are referred to in the function table entries enclosed in parentheses.

Function Table Entries for Example 1: Figure 4 shows the SWIFT Link function table entries for the first example. This example contains functions for the SWIFT-EDIFACT conversion for both directions using DSLSDI/DSLSDO or user-written application programs.

```
DSLFNT NAME=L1DE0,QUEUE=YES,DE=YES,NEXT=L1AI0,KEY1=(SW20,16), *   [1]
      THRESH=50,EXPAND=(COND,UNCOND),EXPNAM=(PRIVATE,COMMON), *
      PRINT=L1PR0,
      DESCR='Data Entry'                                          [2]
DSLFNT NAME=L1AI0,QUEUE=YES,DE=NO,NEXT=L1ERROR,KEY1=(SW20,16),*   [3]
      ROUTE=DWSL1AI0,SPCMND=(OK),THRESH=50,PROT=YES,          *
      EXPAND=(COND,UNCOND),EXPNAM=(PRIVATE,COMMON),           *
      PRINT=L1PR0,                                            *
      DESCR='Authorization of Input Messages'
DSLFNT NAME=L1VE0,QUEUE=YES,NEXT=L1AI0,PROT=NO,THRESH=20,     *   [4]
      KEY1=(SW20,16),KEY2=(SWBHSN,6),PRINT=L1PR0,NOPR=YES,    *
      EXPAND=(COND,UNCOND),EXPNAM=(PRIVATE,COMMON),           *
      DESCR='Visual Verification and Correction'
DSLFNT NAME=L1PR0,QUEUE=YES,LTERM=PRT1,TRAN=DSLH,             *   [5]
      PRFORM=(E,3),STATUS=NOHOLD,THRESH=100,                  *
      MSGID=SACOV,                                            *   [6]
      DESCR='Print Function (Printer PRT1)'
DSLFNT NAME=L1RGPAU,QUEUE=YES,THRESH=30,NEXT=L1ERROR,         *   [7]
      DESCR='Urgent Ready Queue of GPA for Sending to SWIFT'
DSLFNT NAME=L1RFINU,QUEUE=YES,THRESH=30,NEXT=L1ERROR,         *   [8]
      DESCR='Urgent Ready Queue of FIN for Sending to SWIFT'
DSLFNT NAME=L1RFINN,QUEUE=YES,THRESH=30,NEXT=L1ERROR,         *
      DESCR='Normal Ready Queue of FIN for Sending to SWIFT'
DSLFNT NAME=L1ACK,QUEUE=YES,THRESH=100,MSGID=SACOV,           *   [9]
      KEY1=(SW20,16),KEY2=(SWBHSN,6),PRINT=L1PR0,             *
      DESCR='Positive Acknowledgment Queue'
DSLFNT NAME=L1ERROR,QUEUE=YES,DE=NO,THRESH=100,               *   [10]
      NEXT=L1PR0,NOPR=YES,                                    *
      DESCR='Routing Errors from Ready Queues'
DSLFNT NAME=L1FREE,QUEUE=YES,THRESH=100,PROT=YES,NEXT=L1PR1,  *   [11]
      NOPR=YES,PRFORM=(,4),PRINT=L1PR1,                       *
      DESCR='Free Format Queue'
DSLFNT NAME=L1PR1,QUEUE=YES,LTERM=PRT1,TRAN=DSLH,             *   [12]
      PRFORM=(E,3),STATUS=NOHOLD,THRESH=100,                  *
      DESCR='Print Function (System Msg, Free Format Msg)'
DSLFNT NAME=L1AO0,QUEUE=YES,THRESH=100,SPCMND=(AUT),          *   [13]
      KEY1=(SW20,16),KEY2=(SWBHSN,6),ROUTE=DWSL1AO0,          *
      PROT=YES,PRINT=L1PR1,                                   *
      DESCR='Authentication Output Queue'
DSLFNT NAME=L1DO0,QUEUE=YES,THRESH=100,PROT=YES,              *   [14]
      KEY1=(SWBHSN,6),NEXT=L1DO0,SPCMND=ROU,PRINT=L1PR1,      *
      ROUTE=DWSL1DO0,                                         *
      DESCR='Distribution of Output Messages'
DSLFNT NAME=L1SDI,QUEUE=YES,THRESH=100,NEXT=L1AI0,            *   [15]
      DESCR='Sequential Data Input'
DSLFNT NAME=L1SDIES,QUEUE=YES,THRESH=100,NEXT=L1AI0,          *   [16]
      ROUTE=DWSL1ES,                                          *
      DESCR='SDI EDIFACT-SWIFT Conversion'
DSLFNT NAME=L1SDO,QUEUE=YES,THRESH=100,NEXT=L1PR1,            *   [17]
      DESCR='Sequential Data Output'
DSLFNT NAME=L1SDOSE,QUEUE=YES,THRESH=100,NEXT=L1DO0,          *   [18]
      ROUTE=DWSL1SE,                                          *
      KEY1=(SW21,16),KEY2=(SW27,3),                           *
      DESCR='SDO SWIFT-EDIFACT Conversion'
```

Figure 4. Function Table Entries for SWIFT Link First Example (Part 1 of 2)

```
DSLFNT NAME=L1SDY,QUEUE=YES,THRESH=100,                        *   [19]
      DESCR='Output to a System Printer'
DSLFNT NAME=L1CESI,QUEUE=YES,THRESH=100,NEXT=L1VE0,            *   [20]
      TRAN=DCES,STATUS=NOHOLD,INTQUE=L1CES,                    *
      DESCR='Transaction EDIFACT-SWIFT Conversion Input Queue'
DSLFNT NAME=L1CES,QUEUE=YES,THRESH=100,NEXT=L1VE0,             *   [21]
      ROUTE=DWSL1ES,                                           *
      DESCR='Trans. EDIFACT-SWIFT Conversion Intermediate Queu*
      e'
DSLFNT NAME=L1CSE,QUEUE=YES,THRESH=100,NEXT=L1DO0,             *   [22]
      ROUTE=DWSL1SE,                                           *
      KEY1=(SW21,16),KEY2=(SW27,3),                            *
      TRAN=DCSE,STATUS=NOHOLD,                                 *
      DESCR='Transaction SWIFT-EDIFACT Conversion Input Queue'
DSLFNT NAME=L1CSEO,QUEUE=YES,THRESH=100,NEXT=L1DO0,            *   [23]
      DESCR='Transaction SWIFT-EDIFACT Conversion Output Queue*
      '
```

Figure 4. Function Table Entries for SWIFT Link First Example (Part 2 of 2)

Notes:

[1] The function L1DE0 is defined for creating input messages by data entry (DE=YES). It has a queue with a threshold of 50 elements (THRESH=50), which means that an operator message is issued when the 50th message is put into the queue. Messages can be retrieved from the queue of this function using the transaction reference number as a key with a length of up to 16 (KEY1=(SW20,16)). Conditional address expansion is specified and both private and common nicknames are accepted. Completed messages are routed to the authorization function L1AI0 (NEXT=L1AI0). The print queue L1PR0 is assigned to allow for hardcopy printing in this function (PRINT=L1PR0).

Note: Data entry in NOPROMPT mode is not allowed; only the display of data is supported (default of the NOPR parameter).

[2] The descriptive name is coded for all functions (DESCR parameter). This descriptive name is used in the functions selection panel when the menu of the functions available to a user is shown.

[3] The function L1AI0 is defined for visual check and authorization. It has a queue with a threshold of 50 elements (THRESH=50). All fields are protected (PROT=YES) and the creating of new messages by data entry is not allowed (DE=NO). Expansion conditions are as for L1DE0 and messages can be retrieved using the same key. The user command **ok** (SPCMND=OK) is available to show whether the message is correct. The routing table DWSL1AI0 determines where the message is to be routed (ROUTE=DWSL1AI0). The function L1ERROR is specified as next function (NEXT=L1ERROR) if the evaluation of the routing table resulted in an error.

[4] The function L1VE0 is defined to visually verify or correct messages that have not been:

- Authorized (**ok no**)
- Sent to SWIFT because they contain an error
- Acknowledged by SWIFT

The fields are not protected (PROT=NO), and full NOPROMPT mode for correction is specified (NOPR=YES). Messages can be retrieved using one of two keys, the transaction reference number (SW20) and the input

sequence number from the message header (SWBHSN). The mode of the address expansion is as in L1DE0. The function L1PR0 is specified for hardcopy printing (PRINT=L1PR0). Completed messages are routed to function L1AI0 (NEXT=L1AI0).

Note: Unconditional address expansion is specified for NOPROMPT mode to ensure that address fields possibly altered in NOPROMPT mode are automatically corrected by address file expansion.

[5] The hardcopy printer function L1PR0 (TRAN=DSLH) is assigned to the printer terminal PRT1. The print queue is initially set into the NOHOLD status (printed automatically when the message is written to the print queue) and printed in the "blank line and empty field compression" format (PRFORM=(E,3). The queue has a threshold of 100 elements (THRESH=100). Acknowledgments for General Purpose Application messages are routed into this queue by routing module DWSL1IN.

Note: The logical terminal printer PRT1 is defined in the MERVA ESA terminal feature definition table DSLTFDT.

[6] The parameter MSGID=SACOV specifies to print the SWIFT acknowledgment information in a readable form on a separate print page if the MSGACK field in the MERVA ESA TOF contains the system acknowledgment (ISN ACK, APDU 21).

[7] One ready queue, L1RGPAU, is specified for sending General Purpose Application messages to the SWIFT network.

[8] Two functions (ready queues) are specified for sending financial messages to the SWIFT network, L1RFINU for urgent priority messages and L1RFINN for normal priority messages. Both have a threshold of 30 elements. The NEXT= parameter specifies that the messages are routed to L1ERROR when the routing table DWSL1IN used by DWSDGPA results in an error, or when a user processes the messages of L1RFINN or L1RFINU and uses the **eom** command. This should not be done under normal circumstances so that it does not interfere with the processing of DWSDGPA.

Note: The names of the ready queues must also be defined in the SWIFT Link logical terminal table DWSLTT for the appropriate master logical terminal for processing by DWSDGPA. See also Figure 44 on page 135.

[9] The function L1ACK is specified for financial messages positively acknowledged by SWIFT.

Messages can be retrieved using the same keys as in L1VE0, and are printed using the hardcopy print function L1PR0. The parameter MSGID=SACOV specifies, as for the function L1PR0, to display the SWIFT acknowledgment information in a readable form on a separate page.

[10] The function L1ERROR is specified for all kinds of errors arising from message routing. The messages can be printed using function L1PR0.

[11] The function L1FREE is specified for output messages with severe format errors, such as unknown message type. All fields are protected (PROT=YES). The message is initially displayed in NOPROMPT mode (PRFORM=(,4)). Completed messages are routed to the print function L1PR1.

Note: Full NOPROMPT mode is required, although editing of data is not available in this function. PROT=NO and NOPR=YES can be used to correct the message, but then the NEXT parameter must specify not a hardcopy function but a function for further processing of the corrected message.

[12] The function L1PR1 is specified for a hardcopy printer with terminal name PRT1, and the print format is as for function L1PR0.

[13] The function L1AO0 is defined for manual authentication of output messages with authentication errors. Messages can be retrieved using one of two keys, the transaction reference number (SW20) or the output sequence number from the message header (SWBHSN). The user command **authent** (SPCMND=(AUT)) is allowed to authenticate the message. If the **eom** command is used, the completed messages are routed by the routing table DWSL1AO0.

[14] The function L1DO0 is defined for distributing output financial messages with formal errors. Messages can be retrieved using the output sequence number from the message header (KEY2=(SWBHSN,6). The user command **route** is allowed to route the message. The routing table DWSL1DO0 is used to restrict the functions that can be entered with the route command. The queue names L1SDO and L1PR1 are allowed as routing target functions. The NEXT parameter is used in case of routing errors. If the **eom** command is used the completed messages are routed back to the same function L1DO0.

[15] The function L1SDI is defined for sequential input of messages (batch). The messages are routed to the authorization input function L1AI0 (NEXT parameter).

[16] The function L1SDIES is defined for sequential input of EDIFACT messages for conversion to the SWIFT message types 105 and 106. The routing table DWSL1ES is used to route correct messages to the ready queues L1RFINU or L1RFINN and incorrect messages to the correction function L1VE0.

[17] The function L1SDO is defined for sequential output of messages (batch). Messages that cannot be formatted for the sequential output file are routed to the print function L1PR1 (NEXT parameter).

[18] The function L1SDOSE is defined for sequential output of SWIFT message types 105 and 106 for conversion to the EDIFACT messages. The keys definitions allow for checking the availability of all message types 105 or 106 that are needed to create the complete EDIFACT message using the MERVA ESA queue list. The routing table DWSL1SE is used to route incorrect messages to the function L1DO0.

[19] The function L1SDY is defined for printing messages on a system printer (batch). No routing is possible during processing of DSLSDY.

[20] The function L1CESI is defined for transaction input of EDIFACT messages for conversion to the SWIFT message types 105 and 106. The transaction name DCES is defined. The intermediate queue L1CES (INTQUE parameter) is used to ensure always for the complete set of message types 105 or 106 resulting from one EDIFACT message. The message types 105 and 106 are written to the intermediate queue, incorrect EDIFACT messages are routed to the correction function L1VE0 (NEXT parameter).

[21] The function L1CES is used as intermediate queue for transaction input of EDIFACT messages for conversion to the SWIFT message types 105 and

106. The intermediate queue is specified with the parameter INTQUE in the definition of the input queue L1CESI. The routing table DWSL1ES is used as in the function L1SDIES.

[22] The function L1CSE is defined for conversion of the SWIFT message types 105 and 106 into the EDIFACT messages using a transaction program. The transaction name DCSE is defined. The keys definitions allow for checking the availability of all message types 105 or 106 that are needed to create the complete EDIFACT message using the MERVA ESA queue list. The routing table DWSL1SE is used to route the EDIFACT messages to the function L1CSEO, and the incorrect messages to the function L1DO0.

[23] The function L1CSEO is defined for storing the EDIFACT messages that result from the conversion of the SWIFT message types 105 and 106 in the function L1CSE.

Routing Logic of Example 1: Figure 5 shows the data flow of SWIFT input messages.

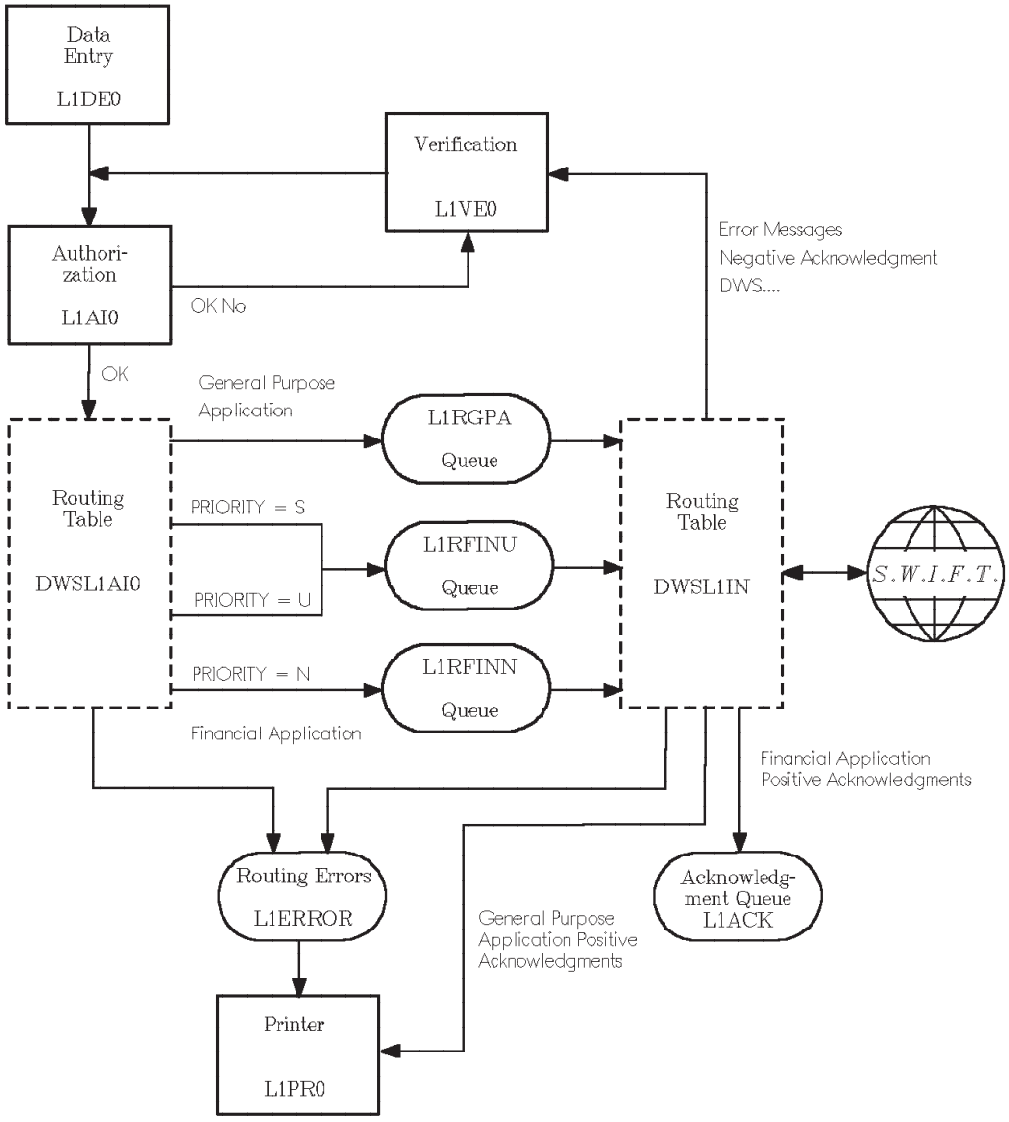

Figure 5. Data Flow of SWIFT Input Messages, Example 1

Data Flow:

1. In the data entry function L1DE0 the data entry clerk types in the data (from a form sheet). In this function all fields are unprotected. When the message is complete, the **eom** command is used to route the message to the Authorization queue L1AI0 following the specification of the NEXT parameter.
2. In the Authorization function L1AI0 the message is protected and can only be checked visually. In this function the special command **ok** is used to show whether the result of this check is positive or not.

   If the Authorization is not given (**ok no**), the message is routed to the verification queue L1VE0 (logic of the routing table DWSL1AI0).
3. In the verification function L1VE0 the message is not protected. Here the message can be corrected and is routed to the authorization queue L1AI0 (NEXT parameter of L1VE0).
4. If the authorization is given (**ok yes**), the message is routed to L1RFINN or L1RFINU depending on the message priority (logic of the routing table DWSL1AI0):

   N Normal priority for L1RFINN

   U Urgent priority for L1RFINU

   S System priority for L1RFINU
5. The message is selected for sending to the SWIFT network. Before sending, DWSDGPA checks the following:
   - The message is a SWIFT input message.
   - The sending logical terminal in the message header is the same as the master logical terminal that owns the ready queue from which the message was read.

   If an error is found, an appropriate error indication is added to the message, and the routing table DWSL1IN routes the message to the verification queue L1VE0. If everything is OK, the message is sent to the SWIFT network. SWIFT returns an APDU Id 21. The text block contains the field 451 which is either '0' (positive acknowledgment) or '1' (negative acknowledgment).
6. The routing table DWSL1IN routes the message either to the acknowledgment queue L1ACK or if in error (NAK) to the verification queue L1VE0. The ACK or NAK information (APDU Id 21) will be added to the message in field MSGACK. Positive acknowledgment messages for the General Purpose Application are routed directly to the print function L1PR0.

   If routing is not successful, the message is routed to the function L1ERROR. This is caused by either the NEXT parameter of L1RFINN and L1RFINU or in the routing table DWSL1IN by the TARGET parameter of the DSLROUTE TYPE=FINAL statement.

   Note: The NEXT parameter in the function table overrides the specification of the DSLROUTE TYPE=FINAL statement.
7. The acknowledgment queue L1ACK can be processed by:
   - A MERVA ESA user who can use the **hco** command to print the message online via the printer function L1PR0 (printer terminal PRT1)
   - DSLSDO, which creates a sequential data set of the messages
   - DSLSDY, which prints the messages on a system printer
   - A user-written application program

Figure 6 shows the data flow of SWIFT output messages.

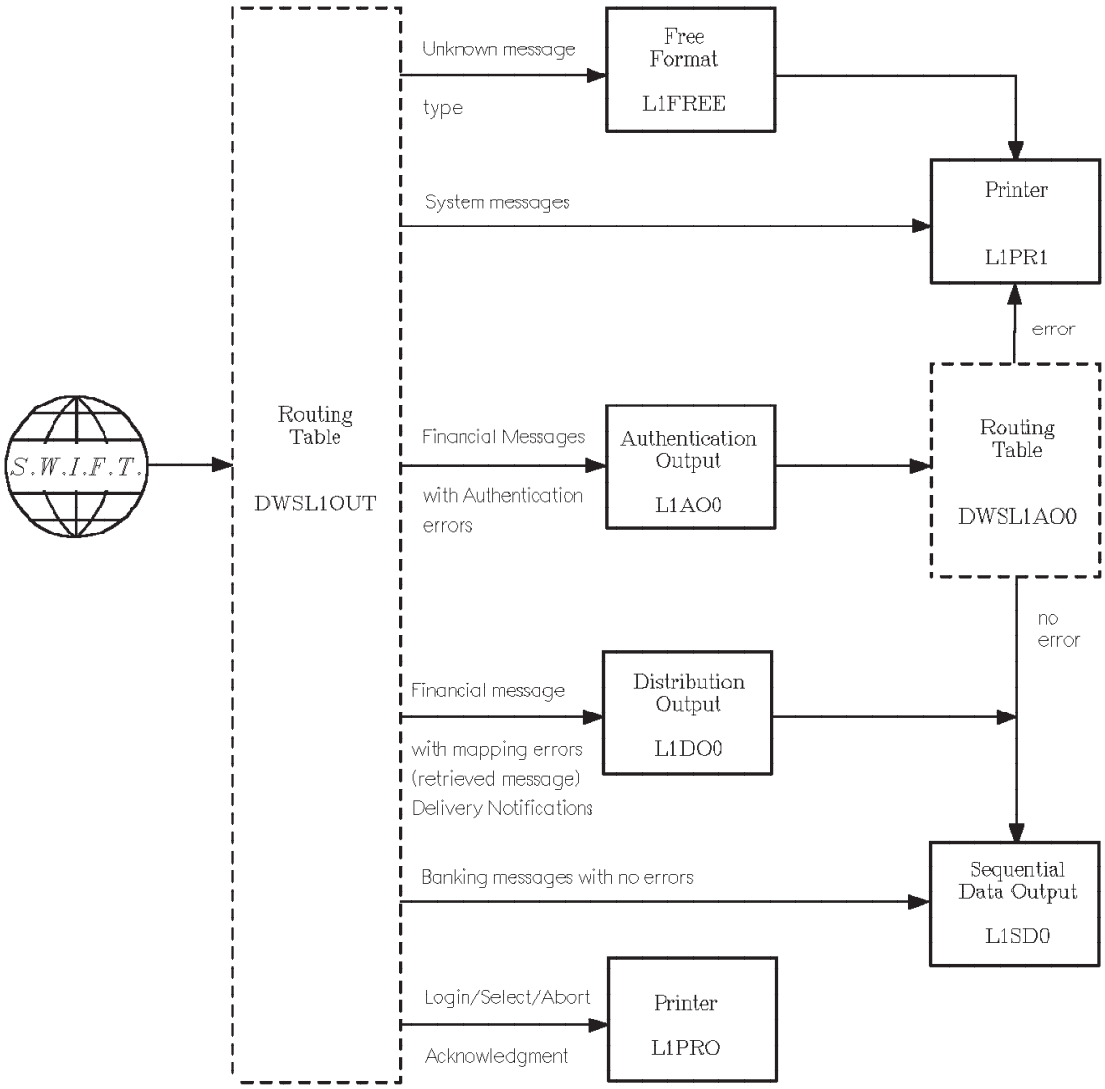

Figure 6. Data Flow of SWIFT Output Messages, Example 1

Data Flow:

1. The routing table DWSL1OUT is used to determine the target queues for messages arriving from SWIFT.

   The MERVA ESA TOF field DSLLFBUF holds a message when the message cannot be transformed to the internal TOF format. In this example the message is sent to the free format queue L1FREE. There it can be printed by L1PR1 (printer terminal PRT1).
2. System messages are routed to the hardcopy printer function L1PR1 with the associated printer terminal address PRT1.
3. The result of the authentication must be checked.

   In this example only those messages are accepted where either no authentication is needed or the authentication was successful with the primary key. These are indicated by the diagnostic messages DWS766 and DWS765. Messages with authentication errors are routed to the authentication output queue L1AO0.

In this function the command **authent** is allowed for manual authentication.

The routing table DWSL1AO0 checks the new status of the message. If the message still contains an authentication error it is routed to the printer function L1PR1. Otherwise it is routed to the sequential data-set-output function L1SDO.

Authentication errors can appear only in financial messages.

4. If the message has no authentication error the message is checked for formal correctness as defined by SWIFT.

   The field MSGTRERR containing a value of 0000 means that the message is formally correct. Incorrect messages are routed to the distribution/output function L1DO0 from where it can be routed by the special command **route**. Delivery notifications and delivery reports from SWIFT are also routed to function L1DO0 for distribution.

5. Correct messages are routed to the sequential data output function L1SDO from where they are processed by the MERVA ESA program DSLSDO. When during the processing of DSLSDO a message cannot be formatted for the sequential output file, it is routed to the hardcopy function L1PR1 (printer terminal PRT1).

6. The positive or negative acknowledgments for LOGIN, SELECT, ABORT, and QUIT are routed directly to the hardcopy function L1PR0 where they are printed in sequence with the originating messages.

Example 2: The second example uses retype verification and more sophisticated routing than Example 1. It is similar to the first example, but with the following modifications:

- The functions for the SWIFT-EDIFACT conversion are not available.
- A forms queue L2FORMS is added to ease the data entry of messages that contain similar texts.
- A retyping function L2RE0 for the SWIFT field 32 is added.
- A second authorization L2AI2 for amounts greater than 10000 is added.

All function names in this example start with the characters L2, and refer to the second master logical terminal in DWSLTT. It uses the following routing tables:

DWSL2DE0 Routing after data entry (L2DE0)

DWSL2RE0 Routing after retype verification (L2RE0)

DWSL2VE0 Routing after visual verification (L2VE0)

DWSL2AI0 Routing after authorization input (L2AI0)

DWSL2AO0 Routing after authentication output (L2AO0)

DWSL2DO0 Routing after distribution output (L2DO0)

DWSL2IN Routing of input messages in DWSDGPA (from ready queues)

DWSL2OUT Routing of output messages in DWSDGPA (from SWIFT)

This example uses the second master logical terminal defined in the SWIFT Link logical terminal table (DWSLTT). A synonym logical terminal is defined for this master logical terminal, and SWIFT output messages are routed accordingly. Figure 7 shows the SWIFT Link function table entries for the second example.

Function Table Entries for Example 2:

```
DSLFNT NAME=L2DE0,QUEUE=YES,DE=YES,NEXT=L2VE0,KEY1=(SW20,16), *   [1]
      THRESH=50,EXPAND=(COND,COND),NOPR=YES,COPY=L2FORMS,     *
      ROUTE=DWSL2DE0,EXPNAM=(PRIVATE,COMMON),PRINT=L2PR0,     *
      DESCR='Data Entry'
DSLFNT NAME=L2FORMS,QUEUE=YES,DE=YES,KEY1=(SW20,16),          *   [2]
      NEXT=L2FORMS,NOPR=YES,PRINT=L2PR0,SPCMND=DEL,CHECK=NO,  *
      DESCR='Data Entry of Forms'
DSLFNT NAME=L2RE0,QUEUE=YES,DE=NO,NEXT=L2VE0,KEY1=(SW20,16),  *   [3]
      THRESH=50,ROUTE=DWSL2RE0,RETYPE=YES,                    *
      EXPAND=(COND,COND),SPCMND=(OK),NOPR=NO,                 *
      EXPNAM=(PRIVATE,COMMON),PRINT=L2PR0,                    *
      DESCR='Retype Verification'
DSLFNT NAME=L2VE0,QUEUE=YES,NEXT=L2VE0,PROT=NO,THRESH=20,     *   [4]
      KEY1=(SW20,16),KEY2=(SWBHSN,6),PRINT=L2PR0,             *
      EXPAND=(COND,UNCOND),NOPR=YES,                          *
      ROUTE=DWSL2VE0,EXPNAM=(PRIVATE,COMMON),                 *
      DESCR='Visual Verification and Correction'
DSLFNT NAME=L2AI0,QUEUE=YES,DE=NO,NEXT=L2ERROR,KEY1=(SW20,16),*   [5]
      ROUTE=DWSL2AI0,SPCMND=(OK),THRESH=50,PROT=YES,          *
      PRINT=L2PR0,                                            *
      DESCR='First Authorization of Input Messages'
DSLFNT NAME=L2AI2,QUEUE=YES,DE=NO,NEXT=L2ERROR,KEY1=(SW20,16),*   [6]
      ROUTE=DWSL2AI0,SPCMND=(OK),THRESH=50,PROT=YES,          *
      PRINT=L2PR0,                                            *
      DESCR='Second Authorization of Input Messages'
DSLFNT NAME=L2PR0,QUEUE=YES,LTERM=PRT1,TRAN=DSLH,             *
      PRFORM=(E,3),STATUS=NOHOLD,THRESH=100,                  *
      DESCR='Print Function (Printer PRT1)'
DSLFNT NAME=L2RGPAU,QUEUE=YES,THRESH=30,NEXT=L2ERROR,         *   [7]
      DESCR='Urgent Ready Queue of GPA for Sending to SWIFT'
DSLFNT NAME=L2RFINU,QUEUE=YES,THRESH=30,NEXT=L2ERROR,         *
      DESCR='Urgent Ready Queue of FIN for Sending to SWIFT'
DSLFNT NAME=L2RFINN,QUEUE=YES,THRESH=30,NEXT=L2ERROR,         *
      DESCR='Normal Ready Queue of FIN for Sending to SWIFT'
DSLFNT NAME=L2ACK,QUEUE=YES,THRESH=100,
      KEY1=(SW20,16),KEY2=(SWBHSN,6),PRINT=L2PR0,             *
      MSGID=SACOV,                                            *
      DESCR='Positive Acknowledgment Queue'
DSLFNT NAME=L2ERROR,QUEUE=YES,DE=NO,THRESH=100,               *
      NEXT=L2PR0,NOPR=YES,PRINT=L2PR0,                        *
      DESCR='Routing Errors from Ready Queues'
DSLFNT NAME=L2FREE,QUEUE=YES,THRESH=100,PROT=YES,NEXT=L2PR1,  *   [8]
      NOPR=YES,PRFORM=(,4),PRINT=L2PR1,                       *
      DESCR='Free Format Queue'
DSLFNT NAME=L2PR1,QUEUE=YES,LTERM=PRT1,TRAN=DSLH,             *   [9]
      PRFORM=(E,3),STATUS=NOHOLD,THRESH=100,                  *
      MSGID=SACOV,                                            *
      DESCR='Print Function (System Msg, Free Format Msg)'
```

Figure 7. Function Table Entries for SWIFT Link Second Example (Part 1 of 2)

```
DSLFNT NAME=L2AO0,QUEUE=YES,THRESH=100,SPCMND=AUT,            *
      KEY1=(SW20,16),KEY2=(SWBHSN,6),ROUTE=DWSL2AO0,          *
      PROT=YES,PRINT=L2PR1,                                   *
      DESCR='Authentication Output Queue'
DSLFNT NAME=L2DO0,QUEUE=YES,THRESH=100,PROT=YES,              *   [10]
       KEY1=(SW20,16),KEY2=(SWBHSN,6),NEXT=L2DO0,SPCMND=ROU,   *
      PRINT=L2PR1,ROUTE=DWSL2DO0,                             *
      DESCR='Distribution of Output Messages'
DSLFNT NAME=L2PR1S,QUEUE=YES,LTERM=PRT1,TRAN=DSLH,            *   [11]
      PRFORM=(E,3),STATUS=NOHOLD,THRESH=100,                  *
      DESCR='Print Function (System Msg, Free Format Msg)'
DSLFNT NAME=L2AO0S,QUEUE=YES,THRESH=100,SPCMND=AUT,           *
      KEY1=(SW20,16),KEY2=(SWBHSN,6),ROUTE=DWSL2AO0,          *
      PROT=YES,PRINT=L2PR1S,                                  *
      DESCR='Authentication Output Queue'
DSLFNT NAME=L2DO0S,QUEUE=YES,THRESH=100,PROT=YES,             *
      KEY1=(SW20,16),KEY2=(SWBHSN,6),NEXT=L2DO0S,SPCMND=ROU,  *
      PRINT=L2PR1S,ROUTE=DWSL2DO0,                            *
      DESCR='Distribution Output'
DSLFNT NAME=L2SDI,QUEUE=YES,                                  *   [12]
      NEXT=L2AI2,                                             *
      DESCR='Sequential Data Input'
DSLFNT NAME=L2SDO,QUEUE=YES,THRESH=100,                       *
      NEXT=L2PR1,                   FOR MASTER                *
      DESCR='Sequential Data Output'
DSLFNT NAME=L2SDOS,QUEUE=YES,THRESH=100,                      *   [13]
      NEXT=L2PR1S,                  FOR SYNONYM               *
      DESCR='Sequential Data Output'
DSLFNT NAME=L2SDY,QUEUE=YES,THRESH=100,  MASTER               *
      DESCR='Output to a System Printer'
DSLFNT NAME=L2SDYS,QUEUE=YES,THRESH=100, SYNONYM              *
      DESCR='Output to a System Printer'
```

Figure 7. Function Table Entries for SWIFT Link Second Example (Part 2 of 2)

Notes:

[1] The function L2DE0 is similar to the function L1DE0 of the first example except that the ROUTE parameter is specified. The routing module DWSL2DE0 checks if a message has the retype field 32. The COPY parameter allows you to use the **copy** command to copy forms from the forms queue L2FORMS.

Note: All functions specify the DESCR parameter for the function selection panel.

[2] The function L2FORMS is used to create partly filled messages that are used like forms (see function L2DE0). The KEY1 parameter allows to get the forms with a key. The SPCMND parameter allows to delete forms that are no longer needed. CHECK=NO prevents checking during the **eom** command, as the form is most likely not a complete message.

[3] The function L2RE0 is specified to allow for retype verification (RETYPE=YES). NOPROMPT mode is not available. The PROT parameter is not specified as RETYPE=YES implies that all non-retype fields are protected, and the fields that must be retyped (the SWIFT field 32 in the example) is shown empty and must be filled in. The special command **ok** is specified (SPCMND=(OK)). The routing table specified with ROUTE=DWSL2RE0 checks for retype errors and the **ok** command.

[4] The function L2VE0 is similar to the function L1VE0 of the first example except that the ROUTE parameter is specified. The routing module DWSL2VE0 checks if a message has the retype field 32. As well as the

messages indicated in the first example for L1VE0, L2VE0 also receives the messages that after processing in L2RE0 have retype errors or the **ok no** indication.

[5] The function L2AI0 is similar to the function L1AI0 of the first example. The routing table DWSL2AI0 not only checks for the message priority but also for the currency and amount subfields in the SWIFT field 32. If the currency is 'USD' and the amount is found and is greater than 10000, the message is routed to a second authorization function L2AI2.

[6] The function L2AI2 authorizes the messages that contain a financial transaction with currency 'USD' and an amount greater than 10000. The same routing module DWSL2AI0 is used as in L2AI0 (ROUTE=DWSL2AI0) which shows how to find out the last processing function to decide if the message can now be routed to one of the ready queues L2RFINU and L2RFINN.

[7] The functions L2RGPAU, L2RFINU, L2RFINN, L2ACK, and L2ERROR are similar to the functions L1RGPAU, L1RFINU, L1RFINN, L1ACK, and L1ERROR of the first example. All the queues used for processing SWIFT input messages, especially the ready queues L2RFINU and L2RFINN, are used by both the master logical terminal and the synonym logical terminal.

[8] The function L2FREE is similar to the function L1FREE of the first example. It is not possible to separate messages in the free format for the master or synonym logical terminal.

[9] The functions L2PR1, L2AO0, and L2DO0 are similar to the functions L1PR1, L1AO0, and L1DO0 of the first example. These three functions are used for SWIFT output messages of the master logical terminal.

[10] The function L2DO0 specifies the routing table ROUTE=DWSL2DO0 that shows how to **not** check the function parameter of the special command route (SPCMND=ROU). Any function specified as target in the command route is accepted.

[11] The functions L2PR1S, L2AO0S, and L2DO0S are used for SWIFT output messages of the synonym logical terminal, and they are similar to the functions L2PR1, L2AO0, and L2DO0 of the master logical terminal.

[12] The function L2SDI is specified for sequential input data processing (Batch). The messages are routed to the second authorization function L2AI2 (NEXT parameter). The THRESH parameter is not specified and defaulted to zero, that is the threshold notification is not done.

[13] The functions L2SDO and L2SDOS are defined for sequential output of messages (batch), and the functions L2SDY and L2SDYS are defined for output to the system printer. L2SDO and L2SDY are for the master logical terminal, L2SDOS and L2SDYS are for the synonym logical terminal.

Routing Logic of Example 2: Figure 8 shows the data flow of SWIFT input messages.

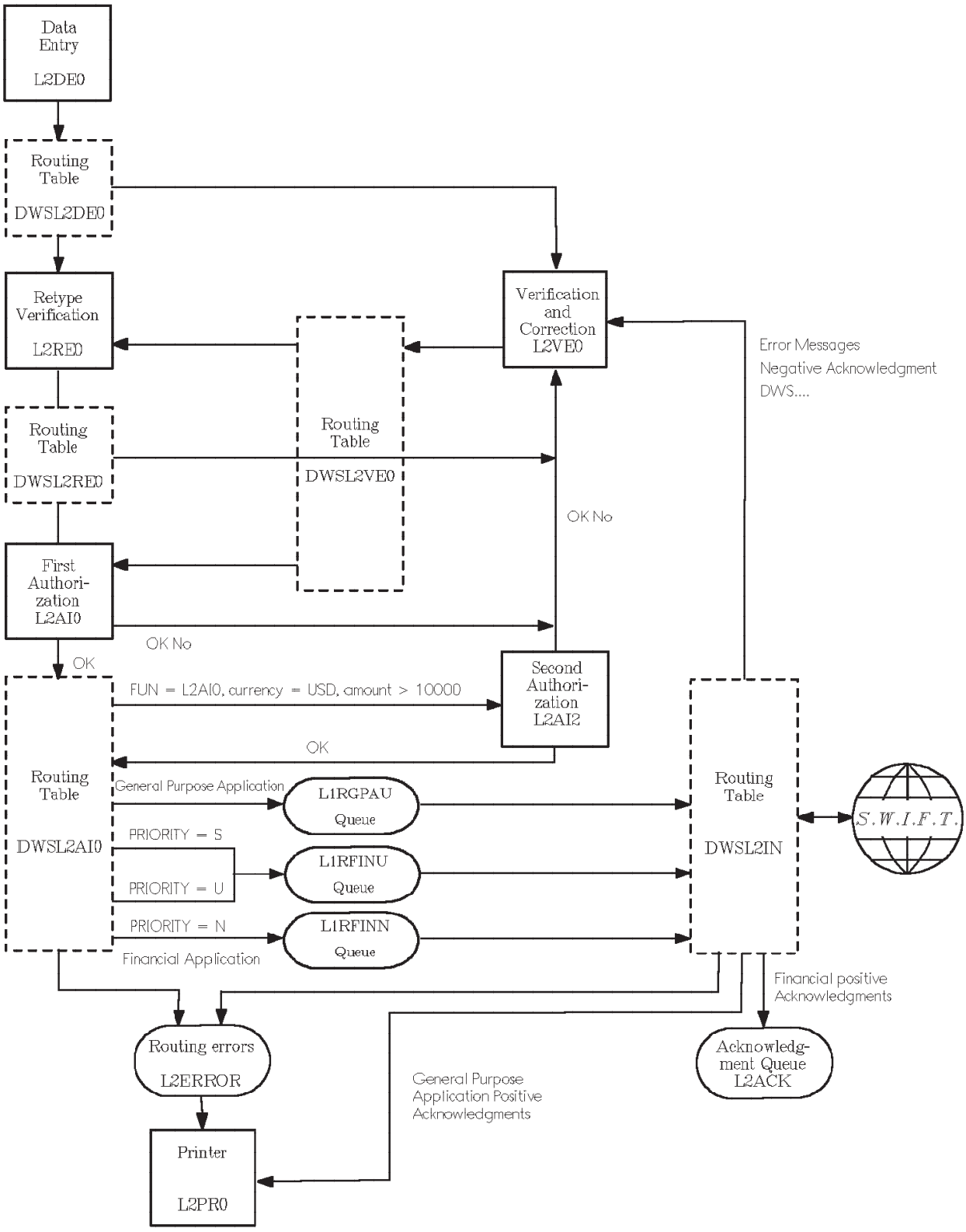

Figure 8. Data Flow of SWIFT Input Messages, Example 2

Data Flow:

1. The data entry clerk types in the message (from a form sheet). The routing table DWSL2DE0 controls the message flow. If the message contains the retype

field SW32 the message is routed to the retype verification function L2RE0, otherwise the message is routed to the verification function L2VE0.

2. In the retype verification all fields are protected except the following subfields of the SWIFT field 32 that have to be verified:
   - Value date
   - Currency code
   - Amount

   These fields are the three parts (subfields) of field 32. The contents of these fields entered in L2DE0 are not shown on the screen but must be entered again. MERVA ESA compares the retype input with the input from L2DE0. If both inputs match then the user must use the **ok** command to show if he detected other errors in the message (**ok no**) or not (**ok yes**). If everything is OK, the message is routed to the first authorization queue L2AI0. If retyping fails or contains other errors (**ok no**), the message is routed to the verification function L2VE0. The routing table DWSL2RE0 controls the message flow.
3. In the verification function L2VE0 the message is not protected. The message can be corrected in this function, for example, when field 32 was incorrectly entered in the L2DE0 function. The routing table DWSL2VE0 controls the message flow. If the message contains the retype field SW32 the message is again routed to the retype verification function L2RE0, otherwise the message is routed to the first authorization function L2AI0.
4. In the first authorization function the message is protected and can only be checked visually. In this function the special command **ok** is used to show whether the result of this check is positive or not.

   The routing table DWSL2AI0 controls the data flow to the ready queues. If the authorization is not given (**ok no**), the message is routed back to the verification queue L2VE0. If the authorization is given (**ok yes**), the message is checked to see whether it came from L2AI0 or L2AI2, as the routing table DWSL2AI0 is used in both L2AI0 and L2AI2.

   From L2AI0: The SWIFT field 32 is checked in the first occurrence regardless of the message type. If the currency is 'USD' and the amount is greater than 10 000, the message is routed to the second authorization queue. Otherwise the message is routed to one of the ready queues L2RFINU or L2RFINN depending on the message priority.

   Note: This is just one example of how to make such tests and should be adapted to meet the requirements of your organization.

   From L2AI2: The message is routed to one of the ready queues L2RGPAU, L2RFINU, or L2RFINN depending on the application and message priority.

   The message priority is evaluated as follows:

   N Normal priority for L2RFINN

   U Urgent priority for L2RFINU

   S System priority for L2RFINU
5. The message is selected for sending to the SWIFT network. Before sending, DWSDGPA checks if the message is a SWIFT input message, and if the sending logical terminal in the message header is the same as the master logical terminal that owns the ready queue from which the message was read, or if it is the same as the synonym of this master logical terminal as defined in the logical terminal table (DWSLTT). If an error is found, an appropriate error indication is added to the message, and the routing table DWSL2IN routes the

message to the verification queue L2VE0. If everything is OK, the message is sent to the SWIFT network. SWIFT sends back an APDU Id 21. The text block contains the field 451 which is either '0' (positive acknowledgment) or '1' (negative acknowledgment).

6. The routing table DWSL2IN routes the message either to the acknowledgment queue L2ACK or if in error (NAK) to the verification queue L2VE0. The ACK or NAK information (APDU Id 21) is added to the message in field MSGACK. Positive acknowledgment messages for the General Purpose Application are routed directly to the print function L2PR0.

   If routing is not successful, the message is routed to the function L2ERROR. This is caused by either the NEXT parameter of L2RFINN and L2RFINU or in the routing table DWSL2IN by the TARGET parameter of the DSLROUTE TYPE=FINAL statement.

   Note: The NEXT parameter in the function table overrides the specification of the DSLROUTE TYPE=FINAL statement.

7. The acknowledgment queue L2ACK can be processed by:
   - A MERVA ESA user, who can use the **hco** command to print the message online via the printer function L2PR0 (printer terminal PRT1)
   - DSLSDO, which creates a sequential data set of the messages
   - DSLSDY, which prints the messages on a system printer
   - A user-written application program

Figure 9 shows the data flow of SWIFT output messages.

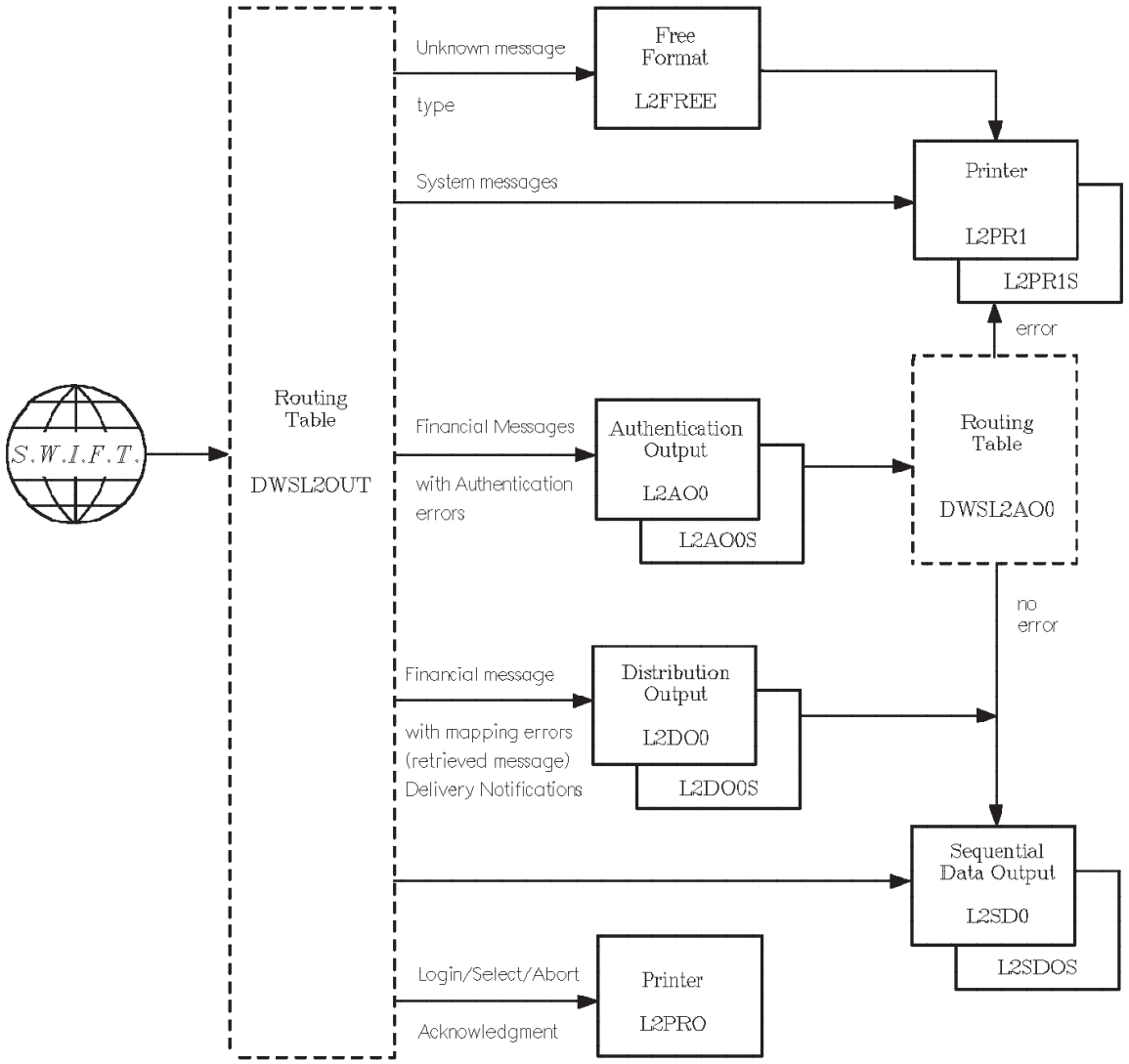

Figure 9. Data Flow of SWIFT Output Messages, Example 2

Data Flow:

Note: The function names with the suffix 'S' refer to the synonym logical terminal (see “Function Table Entries for Example 2” on page 19).

1. The routing table DWSL2OUT is used to determine the target queues for messages arriving from SWIFT.

   The MERVA ESA TOF field DSLLFBUF holds a message when the message cannot be transformed to the internal TOF format. In this example the message is routed to the free format queue L2FREE. There it can be printed by L2PR1 (printer terminal PRT1).
2. System messages of the master logical terminal are routed to the hardcopy printer function L2PR1 with the associated printer terminal address PRT1. System messages of the synonym logical terminal are routed to the hardcopy printer function L2PR1S with the associated printer terminal address PRT1.
3. The result of the authentication must be checked at this point.

In this example only those messages are accepted where either no authentication is needed or this authentication was successful with the primary key. This is indicated by the diagnostic messages DWS766 and DWS765.

Master logical terminal: Messages with authentication errors are routed to the authorization output L2AO0.

Synonym logical terminal: Messages with authentication errors are routed to the authorization output L2AO0S.

In these functions the command **authent** is allowed for manual authentication.

The routing table DWSL2AO0 checks the new status of the message. If the message still contains an authentication error it is routed to the printer function L2PR1 of the master or L2PR1S of the synonym logical terminal, respectively. Otherwise it is routed to the sequential data set output function L2SDO or L2SDOS, respectively.

Authentication errors can appear only in financial messages.

4. If the message has no authentication error the message is checked for formal correctness as defined by SWIFT.

   The field MSGTRERR containing a value of 0000 means that the message is formally correct. Incorrect messages are routed to the distribution/output functions L2DO0 or L2DO0S from which it can be routed by the special command **route**. Delivery notifications and delivery reports from SWIFT are also routed to functions L1DO0 or L1DO0S for distribution.
5. Correct messages are routed to the sequential data output functions L2SDO or L2SDOS from where they are processed by the MERVA ESA program DSLSDO. When during the processing of DSLSDO a message cannot be formatted for the sequential output file, it is routed to the hardcopy function L2PR1 (printer terminal PRT1) from the master logical terminal function L2SDO, or to the hardcopy function L2PR1S (printer terminal PRT1) from the synonym logical terminal function L2SDOS.
6. The positive or negative acknowledgments for LOGIN, SELECT, ABORT, and QUIT are routed directly to the hardcopy function L2PR0 where they are printed in sequence with the originating messages.

Example 3: This example has the same functions for processing of SWIFT input messages as Example 2, except that it uses the checking and expansion transaction DSLCXT to receive messages from SWIFT, and it evaluates branch codes and synonym logical terminals during the routing process. The message flow is identical to that of Example 2.

For processing of SWIFT output messages, the third example uses the MERVA ESA checking and expansion transaction DSLCXT for checking of messages and expansion of SWIFT addresses.

This example uses the third master logical terminal defined in the SWIFT Link logical terminal table (DWSLTT). Two synonym logical terminals are defined for this master logical terminal. Besides these definitions, three branch codes are used with the master logical terminal. The SWIFT output messages are routed depending on their error status and the receiving logical terminals and branch codes.

Figure 10 shows the SWIFT Link function table entries for the third example.

All function names in this example start with the characters L3, and refer to the third master logical terminal and its synonyms in DWSLTT. The following routing tables are used in this example:

DWSL3DE0 Routing after data entry (L3DE0)

DWSL3RE0 Routing after retype verification (L3RE0)

DWSL3VE0 Routing after visual verification (L3VE0)

DWSL3AI0 Routing after authorization input (L3AI0)

DWSL3CXT Routing after DSLCXT processing (L3CXT)

DWSL3DO0 Routing after distribution output (L3DO0)

DWSL3AO0 Routing after authentication output (L3AO0)

DWSL3GPI Routing of input messages for the general application in DWSDGPA (from ready queues)

DWSL3GPO Routing of output messages for the general application in DWSDGPA (from SWIFT network)

DWSL3FII Routing of input messages for the financial application in DWSDGPA (from ready queues)

DWSL3FIO Routing of output messages for the financial application in DWSDGPA (from SWIFT network)

Function Table Entries for Example 3:

```
DSLFNT NAME=L3DE0,QUEUE=YES,DE=YES,NEXT=L3PR0,KEY1=(SW20,16), *   [1]
      THRESH=50,EXPAND=(COND,UNCOND),ROUTE=DWSL3DE0,          *
      EXPNAM=(PRIVATE,COMMON),PRINT=L3PR0,                    *
      DESCR='Data Entry with Checking and Expansion'
DSLFNT NAME=L3RE0,QUEUE=YES,DE=NO,NEXT=L3ERROR,KEY1=(SW20,16),*
      THRESH=50,ROUTE=DWSL3RE0,RETYPE=YES,SPCMND=(OK),NOPR=NO,*
      EXPAND=(COND,UNCOND),EXPNAM=(PRIVATE,COMMON),           *
      PRINT=L3PR0,                                            *
      DESCR='Retype Verification'
DSLFNT NAME=L3AI0,QUEUE=YES,DE=NO,NEXT=L3ERROR,KEY1=(SW20,16),*
      ROUTE=DWSL3AI0,SPCMND=(OK),THRESH=50,PROT=YES,          *
      EXPAND=(COND,UNCOND),EXPNAM=(PRIVATE,COMMON),           *
      PRINT=L3PR0,                                            *
      DESCR='First Authorization of Input Messages'
DSLFNT NAME=L3AI2,QUEUE=YES,DE=NO,NEXT=L3ERROR,KEY1=(SW20,16),*
      ROUTE=DWSL3AI0,SPCMND=(OK),THRESH=50,PROT=YES,          *
      EXPAND=(COND,UNCOND),EXPNAM=(PRIVATE,COMMON),           *
      PRINT=L3PR0,                                            *
      DESCR='Second Authorization of Input Messages'
DSLFNT NAME=L3VE0,QUEUE=YES,NEXT=L3PR0,PROT=NO,THRESH=20,     *
      KEY1=(SW20,16),KEY2=(SWBHSN,6),PRINT=L3PR0,             *
      CHECK=YES,EXPAND=(COND,UNCOND),ROUTE=DWSL3VE0,NOPR=YES, *
      EXPNAM=(PRIVATE,COMMON),                                *
      DESCR='Visual Verification and Correction'
DSLFNT NAME=L3PR0,QUEUE=YES,LTERM=PRT1,TRAN=DSLH,             *
      PRFORM=(E,3),STATUS=NOHOLD,THRESH=100,                  *
      DESCR='Print Function (Printer PRT1)'
DSLFNT NAME=L3RGPAU,QUEUE=YES,THRESH=30,NEXT=L3ERROR,         *
      DESCR='Urgent Ready Queue of GPA for Sending to SWIFT'
DSLFNT NAME=L3RFINU,QUEUE=YES,THRESH=30,NEXT=L3ERROR,         *
      DESCR='Urgent Ready Queue of FIN for Sending to SWIFT'
DSLFNT NAME=L3RFINN,QUEUE=YES,THRESH=30,NEXT=L3ERROR,         *
      DESCR='Normal Ready Queue of FIN for Sending to SWIFT'
DSLFNT NAME=L3ACKF,QUEUE=YES,THRESH=100,                      *   [2]
      KEY1=(SW20,16),KEY2=(SWBHSN,6),PRINT=L3PR0,             *
      DESCR='Positive Acknowledgment Queue of FIN'
DSLFNT NAME=L3ACKG,QUEUE=YES,THRESH=100,                      *   [3]
      KEY2=(SWBHSN,6),PRINT=L3PR0,                            *
      DESCR='Positive Acknowledgment Queue of GPA'
DSLFNT NAME=L3ERROR,QUEUE=YES,DE=NO,THRESH=100,               *
      NEXT=L3PR0,NOPR=YES,PRINT=L3PR0,                        *
      DESCR='Routing Errors from Ready Queues'
DSLFNT NAME=L3FREE,QUEUE=YES,THRESH=100,PROT=YES,NEXT=L3PR1,  *   [4]
      NOPR=YES,PRFORM=(,4),PRINT=L3PR1,                       *
      DESCR='Free Format Queue'
DSLFNT NAME=L3PR1,QUEUE=YES,LTERM=PRT1,TRAN=DSLH,             *
      PRFORM=(E,3),STATUS=NOHOLD,THRESH=100,                  *
      DESCR='Print Function (Free Format Messages)'
```

Figure 10. Function Table Entries for SWIFT Link Functions of the Third Routing Example (Part 1 of 2)

```
DSLFNT NAME=L3CXT,QUEUE=YES,THRESH=100,PROT=YES,NEXT=L3PR1,   *   [5]
      ROUTE=DWSL3CXT,TRAN=DSLX,STATUS=NOHOLD,EXPAND=UNCOND,   *
      CHECK=YES,                                              *
      DESCR='Expand Banking Transaction Messages'
DSLFNT NAME=L3DO0,QUEUE=YES,THRESH=100,PROT=YES,              *   [6]
      KEY2=(SWBHSN,6),NEXT=L3DO0,SPCMND=(ROU,OK),             *
      ROUTE=DWSL3DO0,PRINT=L3PR1,                             *
      KEY1=(SW20,16),                                         *
      DESCR='Distribution of Output Messages'
 DSLFNT NAME=L3PR1AA,QUEUE=YES,LTERM=PRT1,TRAN=DSLH,          *   [7]
      PRFORM=(E,3),STATUS=NOHOLD,THRESH=100,                  *
      DESCR='Print Function for System Messages'
DSLFNT NAME=L3AO0AA,QUEUE=YES,THRESH=100,SPCMND=AUT,          *
      KEY2=(SWBHSN,6),ROUTE=DWSL3AO0,PROT=YES,PRINT=L3PR1AA,  *
      DESCR='Authentication Output Queue'
DSLFNT NAME=L3PR1BB,QUEUE=YES,LTERM=PRT1,TRAN=DSLH,           *
      PRFORM=(E,3),STATUS=NOHOLD,THRESH=100,                  *
      DESCR='Print Function for System Messages'
DSLFNT NAME=L3AO0BB,QUEUE=YES,THRESH=100,SPCMND=AUT,          *
      KEY2=(SWBHSN,6),ROUTE=DWSL3AO0,PROT=YES,PRINT=L3PR1BB,  *
      DESCR='Authentication Output Queue'
DSLFNT NAME=L3PR1CC,QUEUE=YES,LTERM=PRT1,TRAN=DSLH,           *
      PRFORM=(E,3),STATUS=NOHOLD,THRESH=100,                  *
      DESCR='Print Function for System Messages'
DSLFNT NAME=L3AO0CC,QUEUE=YES,THRESH=100,SPCMND=AUT,          *
      KEY2=(SWBHSN,6),ROUTE=DWSL3AO0,PROT=YES,PRINT=L3PR1CC,  *
      DESCR='Authentication Output Queue'
DSLFNT NAME=L3PR1S1,QUEUE=YES,LTERM=PRT1,TRAN=DSLH,           *   [8]
      PRFORM=(E,3),STATUS=NOHOLD,THRESH=100,                  *
      DESCR='Print Function for System Messages'
DSLFNT NAME=L3AO0S1,QUEUE=YES,THRESH=100,SPCMND=AUT,          *
      KEY2=(SWBHSN,6),ROUTE=DWSL3AO0,PROT=YES,PRINT=L3PR1S1,  *
      DESCR='Authentication Output Queue'
DSLFNT NAME=L3PR1S2,QUEUE=YES,LTERM=PRT1,TRAN=DSLH,           *
      PRFORM=(E,3),STATUS=NOHOLD,THRESH=100,                  *
      DESCR='Print Function for System Messages'
DSLFNT NAME=L3AO0S2,QUEUE=YES,THRESH=100,SPCMND=AUT,          *
      KEY2=(SWBHSN,6),ROUTE=DWSL3AO0,PROT=YES,PRINT=L3PR1S2,  *
      DESCR='Authentication Output Queue'
DSLFNT NAME=L3SDI,QUEUE=YES,NEXT=L3AI2,                       *   [9]
      DESCR='Sequential Data Input'
DSLFNT NAME=L3SDO,QUEUE=YES,THRESH=100,NEXT=L3PR1,            *  [10]
      DESCR='Sequential Data Output'
DSLFNT NAME=L3SDOS1,QUEUE=YES,THRESH=100,NEXT=L3PR1S1,        *
      DESCR='Sequential Data Output'
DSLFNT NAME=L3SDOS2,QUEUE=YES,THRESH=100,NEXT=L3PR1S2,        *
      DESCR='Sequential Data Output'
DSLFNT NAME=L3SDY,QUEUE=YES,THRESH=100,                       *
      DESCR='Output to a System Printer'
DSLFNT NAME=L3SDYS1,QUEUE=YES,THRESH=100,                     *
      DESCR='Output to a System Printer'
DSLFNT NAME=L3SDYS2,QUEUE=YES,THRESH=100,                     *
      DESCR='Output to a System Printer'
```

Figure 10. Function Table Entries for SWIFT Link Functions of the Third Routing Example (Part 2 of 2)

Notes:

[1] The functions L3DE0, L3AI0, L3AI2, L3VE0, L3RE0, L3PR0, L1RGPAU L3RFINU, L3RFINN, and L3ERROR are similar to the corresponding functions of the second example for processing of SWIFT input messages (in the second example, the function names all start with L2).

[2] The function L3ACKF is specified for financial messages positively acknowledged by SWIFT.

[3] The function L3ACKG is specified for General Purpose Application messages positively acknowledged by SWIFT.

[4] The functions L3FREE and L3PR1 are used for the same purposes with SWIFT output messages as the functions L2FREE and L2PR1 of the second example.

[5] The function L3CXT receives all SWIFT output messages that could be transformed to the MERVA ESA internal format (TOF). L3CXT is processed by the MERVA ESA checking and expansion transaction that carries out message checking and SWIFT address expansion according to the parameters CHECK=YES and EXPAND=UNCOND, and routes the messages afterwards using the routing table DWSL3CXT (ROUTE parameter). When specifying CHECK=NO and EXPAND=NO, DSLCXT only performs routing.

[6] The function L3DO0 is defined for distributing output financial messages with formal errors. This function exists only once, that is, there are no similar functions for the synonym logical terminals and the branches of the master logical terminal. A user processing this function causes distribution of the messages to the various logical terminals with appropriate parameters of the **ok** or **route** command that is evaluated by the routing table DWSL3DO0 (ROUTE parameter).

[7] The functions L3PR1AA, L3PR1BB, L3PR1CC, L3AO0AA, L3AO0BB, and L3AO0CC are specified for the branches AAA, BBB, and CCC of the master logical terminal. They are used for the same purposes with SWIFT output messages as the functions L2PR1 and L2AO0 of the second example.

[8] The functions L3PR1S1, L3PR1S2, L3AO0S1, and L3AO0S2 are specified for the synonym logical terminals and are used for the same purposes as the functions L3PR1xx and L3AO0xx of the branches of the master logical terminal.

[9] The function L3SDI is defined for sequential input of messages (batch). The messages are routed to the authorization input function L3AI2 (NEXT parameter).

[10] The batch functions L3SDO, L3SDOS1, and L3SDOS2 for sequential data output, and L3SDY, L3SDYS1, and L3SDYS2 for the system printer queues are specified for the master and the synonym logical terminals and are similar to the corresponding functions of the second example.

Routing Logic of Example 3: Figure 11 shows the data flow of SWIFT input messages.

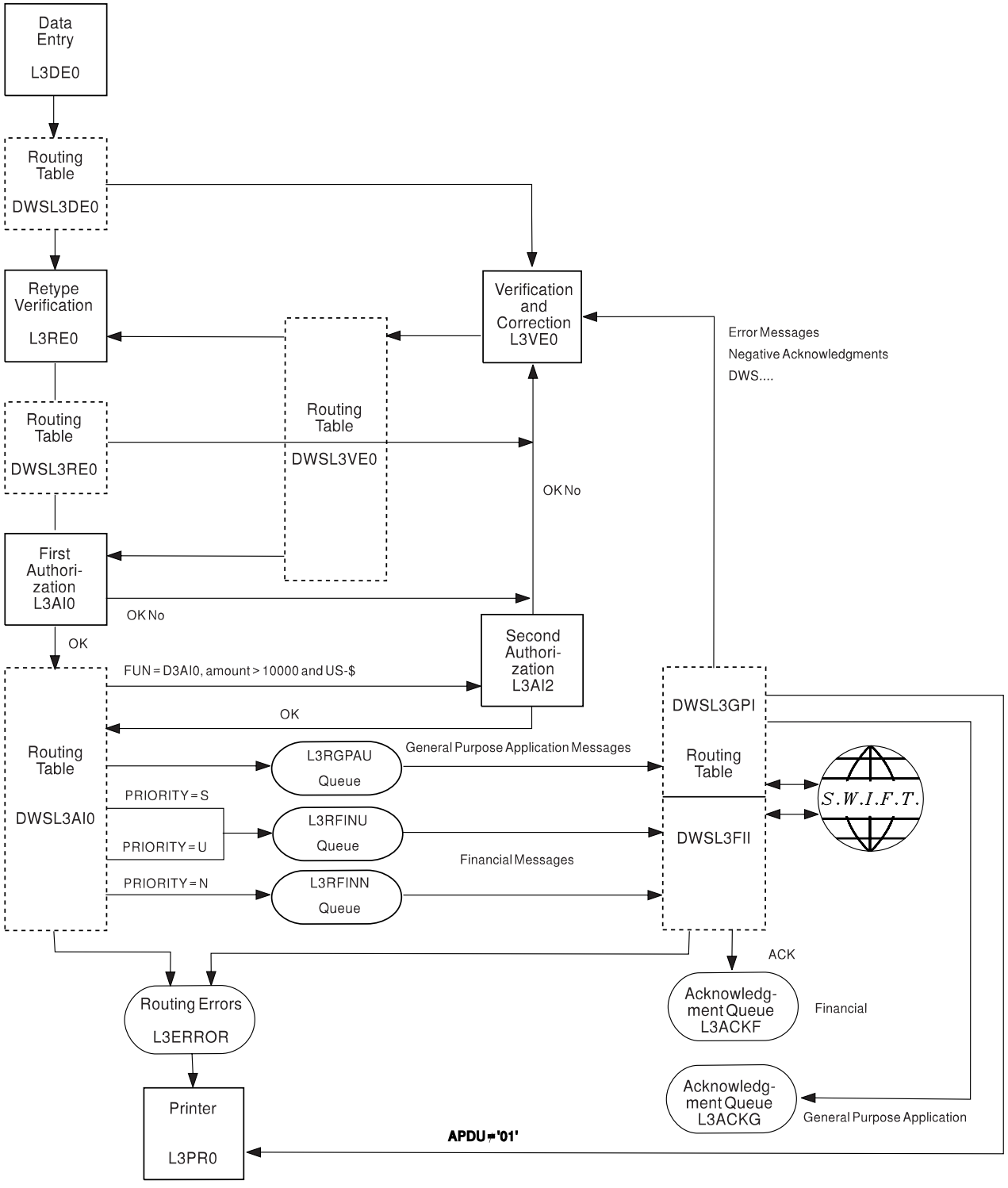

Figure 11. Data Flow of SWIFT Input Messages, Example 3

Data Flow:

1. The data entry clerk types in the message (from a form sheet). The routing table DWSL3DE0 controls the message flow. If the message contains the retype field SW32 the message is routed to the retype verification function L3RE0, otherwise the message is routed to the verification function L3VE0.
2. In the retype verification all fields are protected except the following subfields of field 32:
   - Value date
   - Currency code
   - Amount

   These fields are the three parts (subfields) of the field 32. The contents of these fields entered in L3DE0 are not shown on the screen but must be entered again. MERVA ESA compares the retype input with the input from L3DE0. If both inputs match then the user must use the **ok** command to show if he detected other errors in the message (**ok no**) or not (**ok yes**). If everything is OK, the message is routed to the first authorization queue L3AI0. If retyping fails or contains other errors (**ok no**), the message is routed to the verification function L3VE0. The routing table DWSL3RE0 controls the message flow.
3. In the verification function L3VE0 the message is not protected. Here the message can be corrected. For example, in the case when in L3DE0 field 32 was incorrectly entered. The routing table DWSL3VE0 controls the message flow. If the message contains the retype field SW32 the message is again routed to the retype verification function L3RE0, otherwise the message is routed to the first authorization function L3AI0.
4. In the first authorization function the message is protected and can only be checked visually. In this function the special command **ok** is used to show whether the result of this check is positive or not.

   The routing table DWSL3AI0 controls the data flow to the ready queues. If the authorization is not given (**ok no**), the message is routed back to the verification queue L3VE0. If the authorization is given (**ok yes**), the message is checked whether it came from L3AI0 or L3AI2, as the routing table DWSL3AI0 is used in both L3AI0 and L3AI2.

   From L3AI0: The SWIFT field 32 is checked in the first occurrence independent of the message type. If the currency code is USD (United States Dollars) and the amount is greater than 10 000, the message is routed to the second authorization queue. Otherwise the message is routed to one of the ready queues L3RFINU or L3RFINN depending on the message priority.

   Note: This currency code and amount is just one example of how to make such tests and should be adapted to the requirements of the customer's organization for other currency codes (more than one) and other amounts.

   General Purpose Application messages are routed directly to the ready queue L3RGPAU.

   From L3AI2: The message is routed to one of the ready queues L3RFINU or L3RFINN depending on the message priority.

   The message priority is evaluated as follows:

   N Normal priority for L3RFINN

   U Urgent priority for L3RFINU

S System priority for L3RFINU

5. The message is selected for sending to the SWIFT network. Before sending, DWSDGPA checks if the message is a SWIFT input message, and if the sending logical terminal in the message header is the same as the master logical terminal that owns the ready queue from which the message was read, or if it is one of the synonyms of this master logical terminal as defined in the logical terminal table (DWSLTT).

   General Purpose Application messages from the ready queue L3RGPAU are routed with the routing table DWSL3GPI. Financial messages are routed with the table DWSL3FII.

   If an error is found, an appropriate error indication is added to the message, and the message is routed to the verification queue L3VE0. If everything is OK, the message is sent to the SWIFT network. SWIFT sends back an APDU Id 21. The text block contains the field 451 which is either 0 (positive acknowledgment) or 1 (negative acknowledgment).

6. For financial messages, the routing table DWSL3FII routes the message either to the acknowledgment queue L3ACKF or if in error (NAK) to the verification queue L3VE0. For General Purpose Application messages, the routing table DWSL3GPI routes the message either to the acknowledgment queue L3ACKG or if in error (NAK) to the verification queue L3VE0.

   The ACK or NAK information is added to the message in field MSGACK. Positive acknowledgment messages for the General Purpose Application with an APDU identification (Id) other than 01, for example LOGIN acknowledgments, are routed directly to the print function L1PR0.

   If routing is not successful, the message is routed to the function L3ERROR. This is caused by the NEXT parameter of L3RGPAU, L3RFINN, and L3RFINU function. The TARGET parameter of the DSLROUTE TYPE=FINAL statement in the routing tables DWSL3GPI and DWSL3FII is specified as well, but the NEXT parameter in the function table overrides the specification of the DSLROUTE TYPE=FINAL statement.

7. The acknowledgment queues L3ACKF and L3ACKG can be processed by:
   - A MERVA ESA user, who can use the **hco** command to print the message online via the printer function L3PR0 (printer terminal PRT1)
   - DSLSDO, which creates a sequential data set of the messages
   - DSLSDY, which prints the messages on a system printer
   - A user-written application program

   Note: L3ACKG has no KEY1 specification because the field SW20 exists in financial messages only.

Figure 12 shows the data flow of SWIFT output messages.

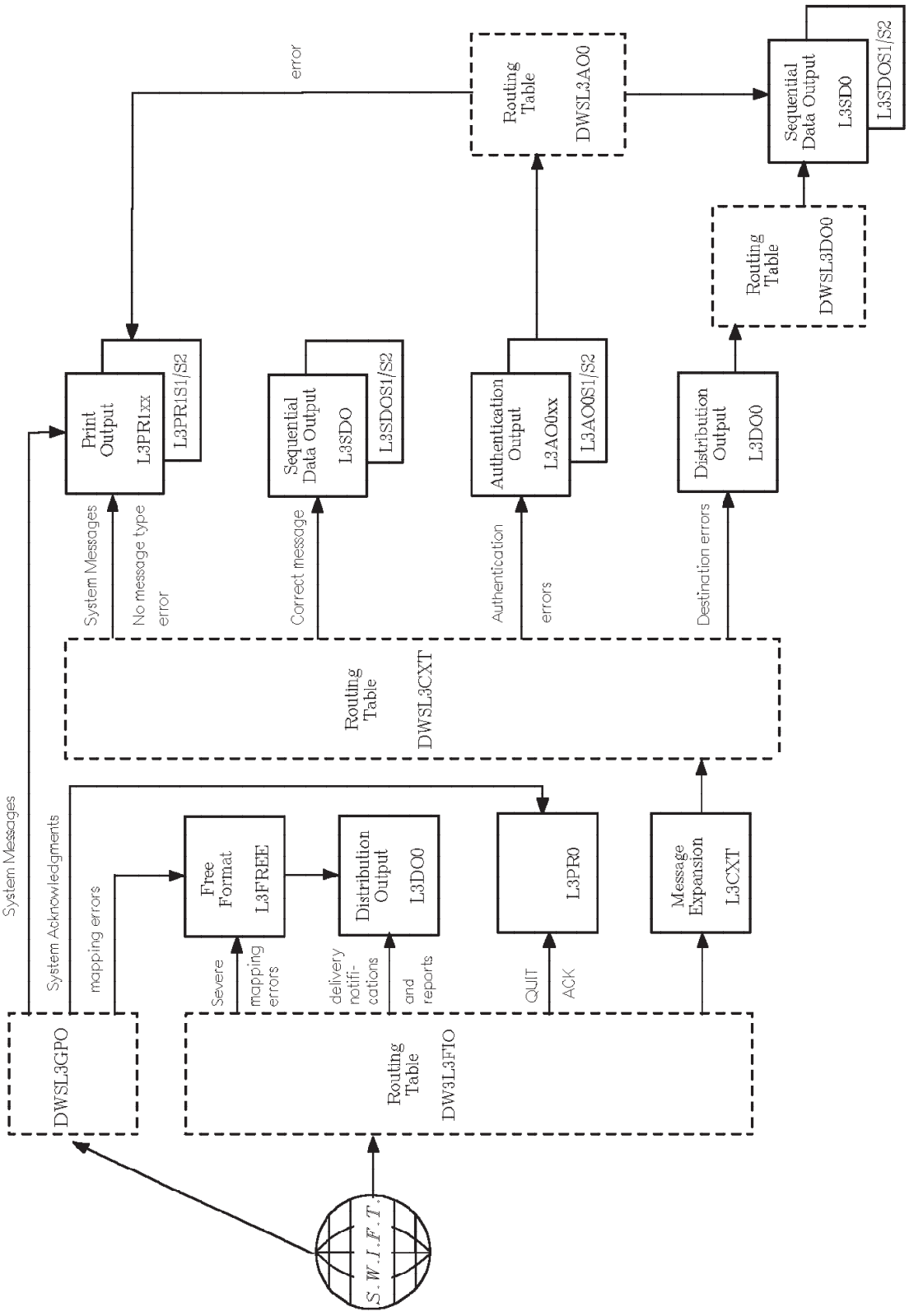

Figure 12. Data Flow of SWIFT Output Messages, Example 3

Data Flow:

Note: The function names with the suffixes AA, BB and CC refer to the branches AAA, BBB, and CCC of the master logical terminal. The function names with the suffixes S1 and S2 refer to the first and second synonym logical terminal (see "Function Table Entries for Example 3" on page 28).

1. The routing tables DWSL3FIO and DWSL3GPO are used to determine the target queues for messages arriving from SWIFT. DWSL3FIO is used for financial messages and DWSL3GPO is used for General Purpose Application messages.

   The MERVA ESA TOF field DSLLFBUF holds a message when the message cannot be transformed to the internal TOF format. In this example the message is routed to the free format queue L3FREE. There it can be printed by L3PR1 (printer terminal PRT1).

   System acknowledgments, for example Login ACKs and Quit ACKs, are routed directly to the hardcopy function L3PR0.

   General Purpose Application system messages are routed to the hardcopy function L3PR1S1 for the first synonym logical terminal, to L3PR1S2 for the second synonym logical terminal, or to one of the hardcopy functions L3PR1AA, L3PR1BB, or L3PR1CC according to the branch code.

   If a financial message can be mapped into the TOF it is routed to the function L3CXT. This function is processed by the MERVA ESA checking and expansion transaction DSLCXT that is started with the transaction code DSLX. The parameters CHECK and EXPAND specify what DSLCXT does.

2. The routing table DWSL3CXT controls the data flow after expansion. DWSL3CXT first determines which branch of the master logical terminal or which synonym logical terminal is to receive the message. If the branch code is wrong, the message is routed to the function L3DO0 for manual distribution.

   The result of the authentication must be checked at this stage.

   In this example only those messages are accepted where either no authentication is needed or this authentication was successful with the primary key. This is indicated by the diagnostic messages DWS766 and DWS765. Messages with authentication errors are routed to the functions L3AO0AA, L3AO0BB, L3AO0CC, L3AO0S1, or L3AO0S1.

3. If the message has no authentication error the message is checked for formal correctness as defined by SWIFT.

   The field MSGTRERR containing a value of 0000 means that the message is formally correct. Incorrect messages are routed to the distribution/output function L2DO0 from which it can be routed by the special command **route**.

   Correct system messages are printed by the functions L3PR1AA, L3PR1BB, L3PR1CC, L3PR1S1, or L3PR1S1, except the message type 021 (retrieved message) that is routed to L3DO0.

   Correct financial messages are routed to the sequential data set output function L3SDO for all branches of the master logical terminal, or to L3SDOS1 and L3SDOS2 for the two synonym logical terminals.

4. In the functions L3AO0AA, L3AO0BB, L3AO0CC, L3AO0S1, and L3AO0S1 the command **authent** is allowed for manual authentication.

   The routing table DWSL3AO0 checks the new status of the message. If the message still contains an authentication error it is routed to one of the functions L3PR1AA, L3PR1BB, L3PR1CC, L3PR1S1, or L3PR1S1, according to the branch code of the master or name of the synonym logical terminal. Otherwise it is

routed to the sequential data set output function L3SDO for all branches of the master logical terminal or to L3SDOS1 and L3SDOS2 for the two synonym logical terminals.

Authentication errors can appear only in financial messages.

5. In the function L3DO0 the **route** and **ok** commands are allowed for manual distribution. The routing table DWSL3DO0 checks the parameter of the **ok** or **route** command that must be one of the branch codes of the master logical terminal (AAA or BBB or CCC) or the suffix of the functions of the two synonym logical terminals (S1 or S2), or one of the functions L3SDOxx or L3PR1xx. If the parameter is incorrect, the message is routed to L3DO0, if the parameter is correct, the messages are routed to the sequential data set output function L3SDO for all branches of the master logical terminal or to L3SDOS1 and L3SDOS2 for the two synonym logical terminals, or to the print functions L3PR1xx.

Examples of Function Table Entries for the Telex Link

The Telex Link supplies a sample of message processing functions in the copy code ENLFNTTC that you can use to create and process telex messages. Some of the sample functions you can modify; for example, you can define more than one function for creating telex messages. Some of the functions, however, cannot be modified (except for the name of the functions), because they are required for communication on the line between the Telex Link and the Headoffice Telex on a fault-tolerant system.

In some cases, having more than one function of a processing step is only possible if there is a field in a telex message that allows you to distinguish between the telex message, for example, of the sender of an outgoing telex message and the receiver of an incoming telex message, which may identify different branches within your organization.

When distributing telex messages using the MERVA ESA command **route**, it is the responsibility of the terminal user to distribute telex messages to functions of the same processing step.

The following functions can be defined more than once for the same processing step:

TXDE0 Telex Data Entry

TXVE0 Telex Verification SWIFT Messages

TXAI0 Telex Authorization SWIFT Messages

TXTKC Telex Test-Key Calculation

TXTKCERR Telex Test-Key Calculation Error

TXERROR Telex Error Queue

TXACK Telex Positively Acknowledged

TXNAK Telex Negatively Acknowledged

TXNOTX No Telex

TXPR0 Telex Print Outgoing Telexes

TXRCV Telex Received

TXPDR Telex Possible Duplicate Received

TXTKV Telex Test-Key Verification

TXTKVERR Telex Test-Key Verification Errors

TXDISTR Telex Distribute Received

TXINVR Telex Invalid Received

TXPR1 Telex Print Received Telexes

TXSDI Telex Sequential Data Input

TXSDO Telex Sequential Data Output

TXSDY Telex Output to a System Printer

The following Telex Link functions can be defined only once, as they are required for communication with the Headoffice Telex on a fault-tolerant system. You must not modify these functions, however, you can modify their names:

TXWAIT Telex Wait for Acknowledgments

TXURG Telex Urgent Ready Queue

TXNRM Telex Normal Ready Queue

TXSTPPDE Telex Possible Duplicate Emitted Queue

TXHCFSND Telex Send to Headoffice Telex on a fault-tolerant system Queue

TXHCFRCV Telex Receive from Headoffice Telex on a fault-tolerant system Queue

TXSTPLR Telex Last Received Queue

Figure 13 shows the function table definitions of the Telex Link. These functions are described on the following pages.

```
DSLFNT NAME=TXDE0,DESCR='Telex Data Entry',                    *   [1]
      DE=YES,FRAME=(0TOP,0BOT),                                *
      KEY1=(ENLTXREF,16,1),                                    *
      NEXT=TXDE0,PRINT=TXPR0,                                  *
      QUEUE=YES,ROUTE=ENLRTDE0,THRESH=100,                     *
      NOPR=YES,MSGID=TCOV,PFKSET=ENLMPF00
DSLFNT NAME=TXVE0,DESCR='Telex Verification SWIFT Messages',   *   [2]
      DE=NO,FRAME=(0TOP,0BOT),                                 *
      KEY1=(ENLTXREF,16,1),                                    *
      NEXT=TXERROR,ROUTE=ENLRTVE0,PRINT=TXPR0,                 *
      QUEUE=YES,THRESH=20,MSGID=TCOV,NOPR=YES
DSLFNT NAME=TXAI0,DESCR='Telex Authorization SWIFT Messages', *
      DE=NO,PROT=YES,FRAME=(0TOP,0BOT),                        *
      KEY1=(ENLTXREF,16,1),                                    *
      NEXT=TXVE0,PRINT=TXPR0,                                  *
      QUEUE=YES,ROUTE=ENLRTAI0,THRESH=50,                      *
      SPCMND=OK,MSGID=TCOV,NOPR=DISPLAY
DSLFNT NAME=TXTKCA,                                            *   [3]
      DESCR='Telex Test-Key Calculation Automatic',            *
      DE=NO,FRAME=(0TOP,ENLTKBOT),                             *
      KEY1=(ENLTXREF,16,1),                                    *
      NEXT=TXTKCA,EXPAND=UNCOND,                               *
      TRAN=DSLX,PRFORM=(E,0),PRINT=TXPR0,PROT=YES,             *
      QUEUE=YES,ROUTE=ENLRTTKC,MSGID=TCOV,THRESH=20
DSLFNT NAME=TXTKC,DESCR='Telex Test-Key Calculation',          *
      DE=NO,FRAME=(0TOP,ENLTKBOT),                             *
      KEY1=(ENLTXREF,16,1),                                    *
      NEXT=TXTKC,PRFORM=(E,0),                                 *
      PRINT=TXPR0,PROT=YES,                                    *
      QUEUE=YES,ROUTE=ENLRTTKC,MSGID=TCOV,THRESH=20
DSLFNT NAME=TXTKCERR,DESCR='Telex Test-Key Calculation Errors'*   [4]
      DE=NO,FRAME=(0TOP,ENLTKBOT),                             *
      KEY1=(ENLTXREF,16,1),                                    *
      NEXT=TXTKCERR,PRFORM=(E,0),PRINT=TXPR0,PROT=YES,         *
      QUEUE=YES,ROUTE=ENLRTTKC,MSGID=TCOV,THRESH=10
DSLFNT NAME=TXERROR,DESCR='Telex Error Queue',                 *   [5]
      DE=NO,FRAME=(0TOP,0BOT),                                 *
      KEY1=(ENLTXREF,16,1),                                    *
      KEY2=(ENLXMHDR,8,23),  ORIGINAL SESSION/SEQUENCE NUMBER *
      NEXT=TXERROR,PRFORM=(E,0),PRINT=TXPR0,PROT=NO,           *
      QUEUE=YES,SPCMND=(ROU,DEL),ROUTE=ENLRTXXX,               *
      NOPR=YES,MSGID=TCOV,THRESH=10
DSLFNT NAME=TXWAIT,DESCR='Telex Wait for Acknowledgments',     *   [6]
      DE=NO,FRAME=(0TOP,0BOT),                                 *
      KEY1=(ENLTXREF,16,1),                                    *
      KEY2=(ENLXMHDR,8,23),  ORIGINAL SESSION/SEQUENCE NUMBER *
      NEXT=TXWAIT,PRFORM=(E,0),PRINT=TXPR0,PROT=YES,           *
      QUEUE=YES,NOPR=NO,MSGID=TCOV,THRESH=50
DSLFNT NAME=TXACK,DESCR='Telex Positively Acknowledged',       *   [7]
      DE=NO,FRAME=(0TOP,0BOT),                                 *
      KEY1=(ENLTXREF,16,1),                                    *
      KEY2=(ENLXMHDR,8,23),  ORIGINAL SESSION/SEQUENCE NUMBER *
      PRFORM=(E,0),PRINT=TXPR0,PROT=YES,                       *
      QUEUE=YES,NOPR=DISPLAY,MSGID=TCOV,THRESH=100
```

Figure 13. Example of Function Table Entries for the Telex Link (Part 1 of 3)

```
DSLFNT NAME=TXNAK,DESCR='Telex Negatively Acknowledged',      *   [8]
      DE=NO,FRAME=(0TOP,0BOT),                               *
      KEY1=(ENLTXREF,16,1),                                  *
      KEY2=(ENLXMHDR,8,23),  ORIGINAL SESSION/SEQUENCE NUMBER *
      PRFORM=(E,0),PRINT=TXPR0,                              *
      SPCMND=(ROU,DEL),ROUTE=ENLRTXXX,PROT=NO,               *
      QUEUE=YES,NOPR=YES,MSGID=TCOV,THRESH=100
DSLFNT NAME=TXNOTX,DESCR='No Telex',                         *   [9]
      DE=NO,FRAME=(0TOP,0BOT),                               *
      NEXT=TXNOTX,PRFORM=(E,0),PRINT=TXPR0,                  *
      QUEUE=YES,SPCMND=(ROU,DEL),ROUTE=ENLRTXXX,MSGID=TCOV,  *
      THRESH=150
DSLFNT NAME=TXPR0,DESCR='Telex Print Outgoing Telexes',      *  [10]
      DE=NO,FRAME=(0TOP,0BOT),                               *
      TRAN=DSLH,LTERM=PRT1,PRFORM=(E,0),                     *
      QUEUE=YES,STATUS=HOLD,MSGID=TCOV,THRESH=150
DSLFNT NAME=TXRCV,DESCR='Telex Received',                    *  [11]
      DE=NO,FRAME=(0TOP,0BOT),                               *
      KEY1=(ENLTXREF,16,1),                                  *
      NEXT=TXRCV,PRFORM=(E,0),PRINT=TXPR1,PROT=YES,NOPR=DISPLA*
      QUEUE=YES,SPCMND=ROU,ROUTE=ENLRTXXX,MSGID=TCOV,        *
      THRESH=100
DSLFNT NAME=TXPDR,DESCR='Telex Possible Duplicate Received', *  [12]
      DE=NO,FRAME=(0TOP,0BOT),                               *
      NEXT=TXPDR,PRFORM=(E,0),PRINT=TXPR1,PROT=YES,NOPR=DISPLA*
      QUEUE=YES,SPCMND=(ROU,DEL),                            *
      ROUTE=ENLRTXXX,MSGID=TCOV,THRESH=20
DSLFNT NAME=TXTKV,DESCR='Telex Test-Key Verification',       *  [13]
      DE=NO,FRAME=(0TOP,ENLTKBOT),                           *
      KEY1=(ENLTXREF,16,1),                                  *
      NEXT=TXTKV,PRFORM=(E,0),PRINT=TXPR1,PROT=YES,QUEUE=YES, *
      ROUTE=ENLRTTKV,MSGID=TCOV,THRESH=100
DSLFNT NAME=TXTKVERR,                                        *  [14]
      DESCR='Telex Test-Key Verification Errors',            *
      DE=NO,FRAME=(0TOP,ENLTKBOT),NEXT=TXTKVERR,             *
      PRFORM=(E,0),PRINT=TXPR1,PROT=YES,                     *
      QUEUE=YES,SPCMND=(ROU,DEL),                            *
      ROUTE=ENLRTTKV,MSGID=TCOV,THRESH=10
DSLFNT NAME=TXDISTR,DESCR='Telex Distribute Received',       *  [15]
      DE=NO,FRAME=(0TOP,0BOT),                               *
      KEY1=(ENLTXREF,16,1),                                  *
      NEXT=TXDISTR,PRFORM=(E,0),PRINT=TXPR1,PROT=YES,        *
      QUEUE=YES,SPCMND=ROU,                                  *
      ROUTE=ENLRTXXX,MSGID=TCOV,THRESH=100
DSLFNT NAME=TXINVR,DESCR='Telex Invalid Received',           *  [16]
      DE=NO,FRAME=(0TOP,0BOT),NEXT=TXINVR,                   *
      PRFORM=(E,0),PRINT=TXPR1,PROT=YES,NOPR=NO,             *
      QUEUE=YES,SPCMND=(ROU,DEL),                            *
      ROUTE=ENLRTXXX,MSGID=TCOV,THRESH=10
DSLFNT NAME=TXPR1,DESCR='Telex Print Received Telexes',      *  [17]
      DE=NO,FRAME=(0TOP,0BOT),TRAN=DSLH,LTERM=PRT1,          *
      PRFORM=(E,0),                                          *
      QUEUE=YES,STATUS=HOLD,MSGID=TCOV,THRESH=150
DSLFNT NAME=TXSDI,DESCR='Telex Sequential Data Input',       *  [18]
      NEXT=TXDE0,QUEUE=YES,ROUTE=ENLRTDE0,                   *
      THRESH=100,MSGID=TCOV
DSLFNT NAME=TXSDO,DESCR='Telex Sequential Data Output',      *
      QUEUE=YES,NEXT=TXPR1,MSGID=TCOV,THRESH=100
DSLFNT NAME=TXSDY,DESCR='Telex Output to a System Printer',  *
      QUEUE=YES,MSGID=TCOV,THRESH=100
```

Figure 13. Example of Function Table Entries for the Telex Link (Part 2 of 3)

```
DSLFNT NAME=TXURG,QUEUE=YES,THRESH=10,STORE=(SMALL,31900),    *  [19]
      PROT=YES,NOPR=NO,NEXT=TXURG,PRINT=TXPR0,MSGID=TCOV
 DSLFNT NAME=TXNRM,QUEUE=YES,THRESH=100,STORE=(SMALL,31900), *  [20]
      PROT=YES,NOPR=NO,NEXT=TXNRM,PRINT=TXPR0,MSGID=TCOV
DSLFNT NAME=TXSTPPDE,QUEUE=YES,THRESH=2,                      *  [21]
      KEY1=(ENLXMHD,8,23),PROT=YES,NOPR=NO,                   *
      NEXT=TXSTPPDE,PRINT=TXPR0,MSGID=TCOV
DSLFNT NAME=TXHCFSND,TRAN=ENLS,LTERM=XXXX,                    *  [22]
      QUEUE=YES,THRESH=5,PRINT=TXPR0,                         *
      PROT=YES,NOPR=NO,MSGID=TCOV
DSLFNT NAME=TXHCFRCV,QUEUE=YES,THRESH=2,PRINT=TXPR1,          *  [23]
      PROT=YES,NOPR=NO,MSGID=TCOV
DSLFNT NAME=TXSTPLR,QUEUE=YES,THRESH=2,PRINT=TXPR1,           *  [24]
      NEXT=TXSTPLR,PROT=YES,NOPR=NO,MSGID=TCOV
DSLFNT NAME=TXCLEAN,QUEUE=DUMMY,MSGID=TCOV                    *  [25]
      COPY  ENLFNTT2                                          *  [26]
```

Figure 13. Example of Function Table Entries for the Telex Link (Part 3 of 3)

Notes:

[1] The function TXDE0 is defined for creating free-format or formatted telex messages (for example, SWIFT messages). The parameter DE=YES permits messages to be created. The program-function key table ENLMPF00 (PFKSET parameter) defines PF keys for the Telex Link commands **txinsert**, **txsplit**, and **txjoin**. If a telex message is saved in the TXDE0 queue, it can be retrieved directly using the telex reference identification (KEY1 parameter).

After you complete a message with the TXDE0 function, the message is routed according to the specifications of the ENLRTDE0 routing table:

- If the message is a telex message and requires test-key calculation, it is routed to the TXTKC function.
- If the message is a telex message and does not require test-key calculation, it is routed to the TXURG or TXNRM function, depending on the telex type.
- If the message is not a telex message, but a SWIFT message, it is routed to the TXAI0 function for authorization.
- If the message is not a telex message, and not a SWIFT message, it is routed to the TXNOTX function.

The parameter MSGID=TCOV specifies that messages are to be displayed and formatted using the MCB ENLTCOV, which is connected to the message identification TCOV. This MCB contains a DSLEXIT statement which includes the telex header fields (parameter IMBED=TX). See “Chapter 11. Cover MCBs” on page 369 for more information on cover MCBs.

The parameter MSGID=TCOV also specifies that the telex command **telex on** can be used. This command extends a SWIFT message by a telex header so that the message can be transmitted via the telex network.

[2] The functions TXVE0 and TXAI0 show how telex and SWIFT messages are processed in an installation. The routing tables ENLRTDE0, ENLRTVE0, and ENLRTAI0 determine whether a message is a formatted telex message (that is, the telex header information is contained in the message, and the text contains a SWIFT message), or a SWIFT message without telex information.

With the TXVE0 function, you verify and correct SWIFT messages that failed authorization in the TXAI0 function, or you can decide to send the SWIFT message via the telex network by adding the telex information with the command **telex on**.

After you complete a message in the TXVE0 function, the message is routed according to the specifications of the routing table ENLRTVE0 if the message:

- Is a telex message and requires test-key calculation, it is routed to the TXTKC function.
- Is a telex message and does not require test-key calculation, it is routed to the TXURG or TXNRM function, depending on the telex type.
- Is not a telex message, but a SWIFT message, it is routed to the TXAI0 function for authorization.
- Is not a telex message, and not a SWIFT message, it is routed to the TXNOTX function.

With the TXAI0 function, you authorize SWIFT messages for sending to the SWIFT network. You use the MERVA ESA command **ok** to give the authorization:

- With **ok yes**, you give the authorization for sending the message to the SWIFT network.
- With **ok no**, the message is routed to the TXVE0 function, where it is either corrected or the telex information for sending the message via the telex network is added.

After you have entered the MERVA ESA command **ok**, the message is routed according to the specifications of the routing table ENLRTAI0:

- If you have entered **ok yes**, the SWIFT message is routed to the **L1RFINU** or **L1RFINN** function for sending to the SWIFT network.

  Note: These functions are only available if the SWIFT Link is installed.
- If you have entered **ok no**, the SWIFT message is routed to the TXVE0 function.
- If you have entered the MERVA ESA command **eom** instead of the **ok** command, the SWIFT message is routed back to the TXAI0 function. You must use the **ok** command.

[3] The functions TXTKCA and TXTKC are used for test-key calculation. With the bottom frame ENLTKBOT (FRAME parameter), the fields of the test-key input area are displayed to provide data for the test-key calculation. The ENLRTTKC routing table uses the test-key flag field to decide about the further processing of the telex message.

When you have completed a message with the TXTKC function, the message is routed according to the specifications of the routing table ENLRTTKC:

- If the message is a telex message and contains the test-key flag OK-C or OK-M for successful test-key calculation, it is routed to the TXURG or TXNRM function, depending on the telex type.
- If the message is a telex message and does not contain a test-key flag (that is, the test-key processing failed), it is routed to the TXTKCERR function.

- If the message is not a telex message (that is, you have used the command **telex off** to remove the telex information), but the message is a SWIFT message, it is routed to the TXAI0 function for authorization.
- If the message is not a telex message (that is, you have used the command **telex off** to remove the telex information), and the message is also not a SWIFT message, it is routed to the TXNOTX function.

The function TXTKCA is defined with TRAN=DSLX. This specifies that the telex messages are processed using the MERVA ESA checking and expansion transaction with routing. The expansion of a message can be extended by an automatic test-key calculation done by calling the Telex Link provided MFS user exit 397 within the standard user exit 23. The MFS user exit 23 is called for each message processed by the MERVA ESA checking and expansion transaction. The automatic test-key calculation works only for message types that have installed a network description for extracting test-key fields in their MCB description.

[4] The function TXTKCERR is used to process telex messages for which test-key calculation failed. The same parameters are used as for the TXTKC function, to allow for the repetition of the test-key calculation. You can either repeat the test-key calculation, or you can use the command **telex off** to remove the telex information when not sending the message via the telex network. The attributes of the TXTKCERR function are the same as the attributes of the TXTKC function, and the same routing table, ENLRTTKC, is used.

[5] The function TXERROR is used to store all telex messages for which routing failed. The SPCMND parameter allows an authorized user to route or delete these messages depending on the error. The routing module ENLRTXXX determines the target queue from the parameter given in the ROUTE command (MSGOK field).

With the TXERROR function, you can either correct the message, using the MERVA ESA command **route** to route the message to another function for correction or further processing, or use the MERVA ESA command **delete** to delete the message if it cannot be corrected or processed correctly by the normal functions of the Telex Link and MERVA ESA.

[6] The function TXWAIT is used to store all telex messages that have been sent to the Headoffice Telex on a fault-tolerant system and the logical acknowledgment received, but for which the transmission acknowledgment has not yet been received. The KEY2 parameter is necessary to allow the Telex Link to find the telex message in this queue when the transmission acknowledgment arrives.

With the TXWAIT function, you can display the telex status fields of a telex message using the command **txinfo on**. You must use the MERVA ESA command **escape** to stop displaying the message in order to not interfere with the Telex Link's processing.

TXWAIT requires a KEY2=(ENLXMHDR,8,23) specification to find a telex message when the transmission acknowledgment arrives. This specification must not be changed.

After the transmission acknowledgment has arrived from the Headoffice Telex on a fault-tolerant system, the Telex Link uses the routing table ENLRTHCF (specified with the RTSND parameter of the ENLPARM macro during the Telex Link customization) to route the telex message. The Telex

Link sets an indicator in the field ENLSTAMP, which is used by the routing table ENLRTHCF to determine the target queue.

[7] The function TXACK is used to store all telex messages that have been sent to the Headoffice Telex on a fault-tolerant system, and for which both a positive logical acknowledgment and a positive transmission acknowledgment have been received. This is the end of the processing cycle for an outgoing telex message. It is up to your installation to decide what to do with the telex messages in this queue.

With the TXACK function, you can use the command **txinfo on** to display the telex status fields of a telex message. You must use the MERVA ESA command **escape** to stop displaying the message, as there is no routing provided for this function. You can use the MERVA ESA program DSLSDO to write these telex messages onto a sequential file, or the MERVA ESA program DSLSDY to print them on a system printer.

[8] The function TXNAK is used to store all telex messages that have been sent to the Headoffice Telex on a fault-tolerant system, and for which either a negative logical acknowledgment was received, or a positive logical acknowledgment and a negative transmission acknowledgment were received. The SPCMND parameter allows an authorized user to route or delete these messages, depending on the error.

With the TXNAK function, you can display the telex status fields of a telex message using the command **txinfo on**. You can correct the message if necessary, and use the MERVA ESA command **route** to route the telex message to the TXTKC, TXURG, or TXNRM function for retransmission, or the MERVA ESA command **delete** to delete the message, should you no longer wish to send it.

[9] The function TXNOTX is used to store messages that contained neither the telex nor SWIFT message indicators during routing. The SPCMND parameter allows an authorized user to route or delete these messages, depending on the error. You can correct the message if necessary, and then use the MERVA ESA command **route** to route the message to an appropriate function for further processing, or you can use the MERVA ESA command **delete** to delete the message if it is no longer required.

[10] The function TXPR0 is used for hardcopy printing of outgoing free-format or formatted telex messages. The transaction code DSLH (TRAN parameter) and the logical terminal name PRT1 (LTERM parameter) are used. In particular, the LTERM parameter may be changed in your installation. The STATUS=HOLD parameter prevents printing until the MERVA ESA command **sf txpr0** sets the function into the NOHOLD status. Telex messages that are routed from the TXWAIT function to the TXACK function are at the same time also routed to the TXPR0 function. The TXPR0 function is processed by the MERVA ESA program DSLHCP.

[11] The Telex Link routes a received telex message to the TXRCV function only if it is sure that it is not a duplicate. The received telex message is also routed to the TXSTPLR function to enable the Telex Link to detect duplicate received telex messages automatically. Telex Link uses the routing table ENLRTHCF, specified with the RTRCV parameter of the ENLPARM macro during the Telex Link customization. The SPCMND parameter allows the MERVA ESA command **route** to process further the received telex messages, for example, to route it to the TXTKV function for test-key verification.

With the TXRCV function, you use the MERVA ESA command **route** to route the received telex message, depending on its contents if the received telex message:

- Contains a test key, you route it to the TXTKV function for test-key verification.
- Does not contain a test key, you can route it to:
  - The TXDISTR function for further distribution, which depends on your installation.
  - The TXSDO function for creating a sequential file of received telex messages.
  - The TXSDY function for printing the received telex messages on a system printer.

[12] If the Telex Link cannot find a telex message in the TXSTPLR function, it cannot decide whether a received telex message is a duplicate. The Telex Link therefore stores it in the TXPDR queue. The Telex Link uses the routing table ENLRTHCF specified with the RTRCV parameter of the ENLPARM macro during the Telex Link customization. The received telex message is also routed to the TXSTPLR function to enable the Telex Link to detect duplicate received telex messages automatically. The SPCMND parameter allows the MERVA ESA commands **route** and **delete** to be used.

With the TXPDR function you decide about the further processing of the received telex message if you discover that the received telex message:

- Is a duplicate, you use the MERVA ESA command **delete** to delete the message as it was processed already earlier.
- Is not a duplicate, you use the MERVA ESA command **route** to route the received telex message, as described for the function TXRCV. For example, to the TXTKV function for test-key verification.

[13] The function TXTKV is used for test-key verification of received telex messages. With the bottom frame ENLTKBOT (FRAME parameter), the fields of the test-key input area are displayed to provide the data for the test-key verification. The routing table ENLRTTKV uses the test-key flag field to determine how the telex message is further processed.

After you complete a message in the TXTKV function, the message is routed according to the specifications of the routing table ENLRTTKV:

- If the message contains the test-key flag OK-V, OK-G or OK-M for successful test-key verification, it is routed to the TXDISTR function for further distribution.
- If the message does not contain a test-key flag, that is, the test-key processing failed, it is routed to the TXTKVERR function.

[14] With the TXTKVERR function, you decide what to do with a telex message whose test-key verification failed. You can either repeat the test-key verification, or you can use the MERVA ESA commands **route** or **delete** to further process the message or to delete it. The attributes of the TXTKVERR function are the same as those of the TXTKV function, and the same routing table, ENLRTTKV, is used.

[15] The TXDISTR function is the last function in the processing cycle of a received telex message. A received telex message is routed to the TXDISTR function from the TXRCV, TXPDR, TXTKV, or TXTKVERR function. Your installation can decide what to do with these telex messages using the MERVA ESA command **route**, which is authorized by the SPCMND parameter.

With the TXDISTR function, you use the MERVA ESA command **route** to route the received telex message, depending on its contents, to:

- Any function of received telex messages that is defined in your installation
- The TXSDO function for creating a sequential file of received telex messages
- The TXSDY function for printing the received telex messages on a system printer

[16] A received message is routed to the TXINVR function using the routing table ENLRTHCF (specified with the RTSND or RTRCV parameter of the ENLPARM macro during the Telex Link customization), if it is either:

- An unidentified acknowledgment message from the Headoffice Telex on a fault-tolerant system, that is, no telex message was found in the TXWAIT or TXNAK queue for this acknowledgment), or
- A completely unidentified message from the Headoffice Telex on a fault-tolerant system

The SPCMND parameter authorizes the use of the MERVA ESA commands **route** and **delete** to determine how these messages are further processed.

[17] The function TXPR1 is used for hardcopy printing of received telex messages. The transaction code DSLH (TRAN parameter) and the logical terminal name PRT1 (LTERM parameter) are used. The LTERM parameter in particular may be changed to suit your installation's requirements. The STATUS=HOLD parameter prevents printing until the MERVA ESA command **sf txpr1** sets the function to the NOHOLD status. The TXPR1 function is processed by the MERVA ESA program DSLHCP.

[18] The functions TXSDI, TXSDO and TXSDY are used for sequential data set input, sequential data set output, or the printing of telex messages on a system printer, as defined by the MERVA ESA programs DSLSDI, DSLSDO and DSLSDY, respectively.

- **Telex Sequential Data Input.** The TXSDI function is used by the MERVA ESA program DSLSDI to store the messages loaded from a sequential file before routing them to the final queue. You must never use the TXSDI function for another purpose. The TXSDI function uses the same routing table as the TXDE0 function (ROUTE parameter).
- **Telex Sequential Data Output.** The TXSDO function is used by the MERVA ESA program DSLSDO to create a sequential file of telex messages. Messages that cannot be formatted for the sequential file are routed to the hardcopy function TXPR1.
- **Telex Output to a System Printer.** The TXSDY function is used by the MERVA ESA program DSLSDY to print telex messages on a system printer.

The Telex Link Internal Functions

[19] The functions TXURG and TXNRM are used for the telex messages ready for sending to the Headoffice Telex on a fault-tolerant system. TXURG is for urgent telex messages with the telex type U, TXNRM is for all other telex messages with a type of N for normal, T for timed, or P for telexes to be printed on the TXIP S/1 printer. The telex messages in TXURG are sent before the telex messages in TXNRM.

[20] The functions TXURG and TXNRM contain the parameter

STORE=(SMALL,31900) which indicates that telex messages up to a size of 31900 bytes can be stored in the ready queue.

[21] The function TXSTPPDE is used to store the last telex message sent to the Headoffice Telex on a fault-tolerant system until the logical acknowledgment is received from the Headoffice Telex on a fault-tolerant system. If the logical acknowledgment does not arrive within the time specified in the RTIM parameter of the ENLPARM macro, an authorized user can resend the message with the command **txdisp recover**. The KEY1 parameter is required by the Telex Link as shown. This specification must not be changed.

[22] The function TXHCFSND is used for sending one telex message to the Headoffice Telex on a fault-tolerant system. ENLSTP stores the telex message in the TXHCFSND queue, and the program ENLHCF1 is started with the transaction code ENLS (TRAN parameter). The LTERM parameter must specify the logical terminal name for the line between the Telex Link and Headoffice Telex on a fault-tolerant system. ENLHCF1 gets the telex message from this queue and gives it to CICS or IMS for sending, and then ENLHCF1 deletes the telex message from the TXHCFSND queue.

[23] The function TXHCFRCV is used for receiving one telex message from the Headoffice Telex on a fault-tolerant system. ENLHCF1 stores the telex message in the TXHCFRCV queue, and the program ENLSTP gets control from the MERVA ESA program DSLNUC. ENLSTP gets the telex message from this queue and processes it depending on its contents, and then deletes the telex message from the TXHCFRCV queue.

[24] The function TXSTPLR is used by the program ENLSTP to detect duplicate received telex messages. If a telex message is received via the TXHCFRCV queue, it is compared with the message in the TXSTPLR queue:

- If the TXSTPLR queue is empty and Headoffice Telex on a fault-tolerant system indicated a possible duplicate emission, ENLSTP cannot determine whether the received telex message is a duplicate. The telex message is therefore routed to the possible duplicate received queue TXPDR.
- If a telex message is found in the TXSTPLR queue, and it is the same message as the one just received, the one just received is discarded, as it is a duplicate.
- If a telex message is found in the TXSTPLR queue, and it is not the same message as the one just received, the one just received is routed to the TXRCV queue.

[25] The function TXCLEAN is used as a dummy queue in the routing table ENLRTHCF to dispose of messages found in the queues TXHCFSND and TXHCFRCV during the start-up of the Telex Link (ENLSTP). These messages are given an ENLSTAMP field with the contents 'CLEANUP'. The deletion is caused by the queue type DUMMY. If you want to keep the messages found in these queues, you can use a real queue instead, for example, a hardcopy print queue.

[26] This COPY statement imbeds the necessary functions for Telex Link via workstation. The Telex Link via workstation is described in *MERVA Workstation Based Functions.*

Note: For all Telex Link functions MSGID=TCOV is defined. This means that the display or printout of a message in this function is formatted according to the definition in the ENLTCOV MCB.

Data Flow of Outgoing Telex Messages: Figure 14 shows an example of the paths taken by outgoing telex messages.

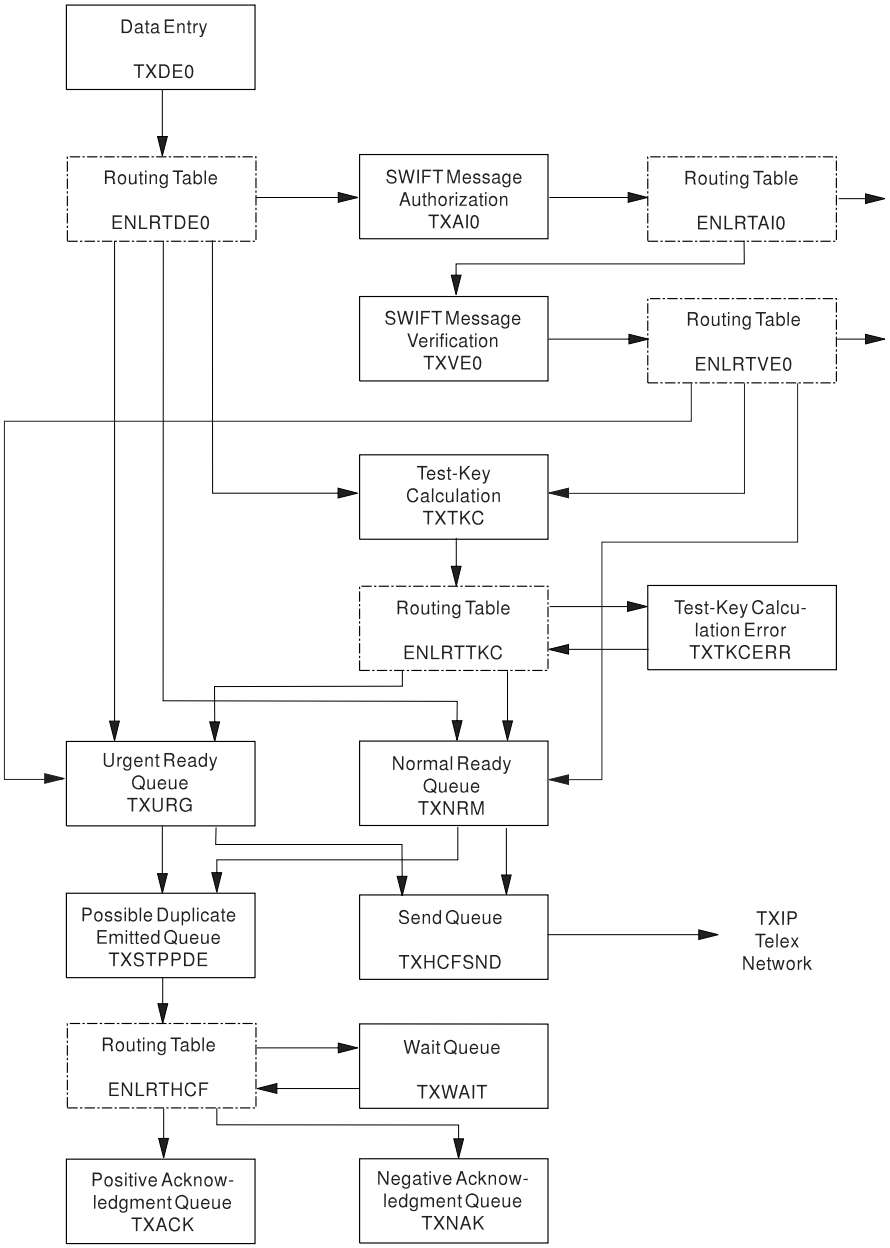

Figure 14. Sample Data Flow of Outgoing Telex Messages

Data Flow:

1. The data for the telex message is entered. The routing table ENLRTDE0 controls the message path. If the message contains the value "Y" in the field ENLTXIND, the message is a telex message:
   - If the message also contains the value "YES" in the field ENLTKIND, test-key calculation is required, and the message is routed to the test-key verification function TXTKC.
   - If the message does not contain the value "YES" in the field ENLTKIND, test-key calculation is not required, and the message is routed according to the contents of the telex type field ENLTXPRI:

- If the telex type is “U” (urgent), the telex message is routed to the urgent ready queue function TXURG.
- If the telex type is “N” (normal), “T” (timed), or “P” (print only), the telex message is routed to the normal ready queue function TXNRM.

If the message is not a telex message but a SWIFT message, it is routed to the function TXAI0 for authorization. If the message is neither a telex message nor a SWIFT message, it is routed to the function TXNOTX (no telex message).

2. In the test-key calculation function TXTKC, the test-key input area is displayed for entering the data necessary for the test-key calculation. After this calculation, the routing table ENLRTTKC first tests the field ENLTXIND for the value “Y” to verify that the message is still a telex message. If not, the message is routed as after the TXDE0 function (SWIFT message or neither SWIFT nor telex message).

   In a telex message, the test-key flag in the field ENLTKFLG is tested:
   - If the test-key flag starts with “OK”, that is, the test-key calculation was successful, the message is routed to the TXURG or TXNRM function depending on the telex type, as after TXDE0.
   - If the test-key flag is not found or does not start with “OK”, that is, the test-key calculation was not successful or was not made, the message is routed to the test-key calculation error function TXTKCERR.
3. The telex message is selected for sending to the Headoffice Telex on a fault-tolerant system. For this purpose, the program ENLSTP stores the telex message in the two queues TXSTPPDE and TXHCFSND without using a routing table. The names of these queues are defined in ENLPRM.
4. When the logical acknowledgment is received from the Headoffice Telex on a fault-tolerant system, the routing table ENLRTHCF routes the message. The Telex Link provides a stamp in the field ENLSTAMP that indicates if the telex message was acknowledged:
   - Positively by the Headoffice Telex on a fault-tolerant system, then it is routed to the TXWAIT function
   - Negatively by the Headoffice Telex on a fault-tolerant system, then it is routed to the TXNAK function
5. When the transmission acknowledgment is received from the Headoffice Telex on a fault-tolerant system, the routing table ENLRTHCF routes the message. The Telex Link provides a stamp in the field ENLSTAMP that indicates if the telex message was acknowledged:
   - Positively by the Headoffice Telex on a fault-tolerant system, then it is routed to the TXACK function
   - Negatively by the Headoffice Telex on a fault-tolerant system, then it is routed to the TXNAK function

Data Flow of Incoming Telex Messages: Figure 15 shows the paths taken by incoming telex messages.

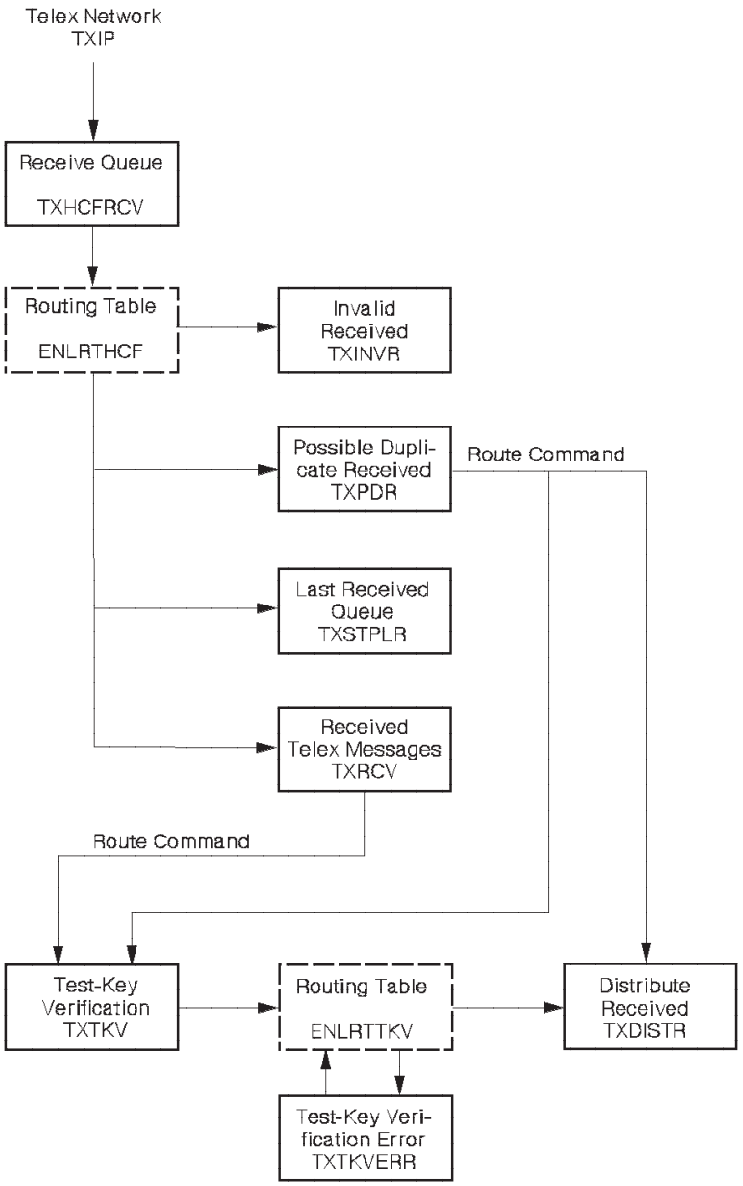

Figure 15. Incoming Telex Message Paths

Data Flow:

1. The telex message is received by the Headoffice Telex on a fault-tolerant system and sent to the Telex Link. The receive transaction ENLHCF1 stores the telex message in the TXHCFRCV queue as defined in ENLPRM.
2. The Telex Link program ENLSTP is invoked to process the received telex message. The routing table ENLRTHCF routes the message, and the Telex Link provides a stamp in the field ENLSTAMP for the routing decision:
   - If the received message cannot be identified, it is routed to the TXINVR function.

- If the Telex Link cannot decide whether the received telex message is a duplicate because the TXSTPLR queue is empty, the received telex message is routed to the possible duplicate received function TXPDR and the last received queue TXSTPLR.
- If the Telex Link can decide that the received telex message is not a duplicate because it is different from the message in the TXSTPLR queue, the received telex message is routed to the received telex message function TXRCV and the last received queue TXSTPLR.

  Note: If a received telex message is definitely a duplicate, then it is discarded.

3. In the TXRCV and TXPDR queues, you must decide how to continue the processing of a received telex message:
   - If a message in the TXPDR function is a duplicate, use the MERVA ESA command **delete** to delete the telex message.
   - If a message in the TXPDR function (that is not a duplicate), or a message in the TXRCV function requires test-key verification, use the MERVA ESA command **route** to route the telex message to the test-key verification function TXTKV.
   - If the telex message does not require test-key verification, you can route it to the TXDISTR function for further distribution or directly to the function where it is finally processed.
4. In the test-key verification function TXTKV, the test-key input area is displayed for entering the data necessary for the test-key verification. After this verification, the routing table ENLRTTKV tests the test-key flag in the field ENLTKFLG if the test-key flag:
   - Starts with OK, that is, the test-key verification was successful, the message is routed to the TXDISTR function for further distribution.
   - Is not found or does not start with OK, that is, the test-key verification was not made or was not successful, the message is routed to the test-key verification error function TXTKVERR.
5. In the function TXDISTR you can distribute the received telex messages as required by your organization using the MERVA ESA command **route**.

Data Flow of a SWIFT Message: The paths taken by SWIFT messages in the TXVE0 and TXAI0 functions are similar to the flow of SWIFT messages as described in "Function Table Entries for Example 1" on page 10. However, the routing table samples ENLRTVE0 (after TXVE0) and ENLRTAI0 (after TXAI0) test whether the message is a SWIFT message without telex information, or whether the telex information was added:

- If the telex information was added, the message is routed to the functions for sending to the telex network (TXTKC for test-key calculation or one of the ready queues TXURG or TXNRM).
- If the telex information was not added, the message is routed to the SWIFT Link ready queues L1RFINU or L1RFINN that are only available in the MERVA ESA function table if the SWIFT Link is also installed.

Function Table Example for the MERVA Link

All MERVA ESA queues used by the MERVA Link must be specified in the MERVA ESA Function Table DSLFNTT.

The Function Table entry for the MERVA System Control Facility and the DSLFNTT entries used in the MERVA Link installation verification sample are

shown in Figure 16. The copy book EKAFNTTC of the MERVA ESA macro library contains the code to generate these DSLFNTT entries.

```
*-------------------------------------------------------------------
*        MERVA SYSTEM CONTROL FACILITY (EUD APPLICATION)
*-------------------------------------------------------------------
         DSLFNT NAME=MSC,QUEUE=NO,PROGRAM=EKAEMSC,NOPR=NO,             *    [1]
               FRAME=(AC01,0BOT),PFGROUP=40,                           *
               DESCR='MERVA System Control'
*-------------------------------------------------------------------
*        MERVA LINK SAMPLE UNIQUE REQUEST QUEUES
*-------------------------------------------------------------------
         DSLFNT NAME=EKAEOM,DE=YES,ROUTE=EKARTS,                       *    [2]
               QUEUE=YES,THRESH=100,SPCMND=(ROU,DEL),                  *
               DESCR='MERVA Link Sample Edit Outgoing Message Queue'
         DSLFNT NAME=EKAEAM,DE=NO,ROUTE=EKARTS,                        *    [3]
               QUEUE=YES,THRESH=50,SPCMND=(ROU,DEL),KEY1=(EKAAMSUB,4), *
               DESCR='MERVA Link Sample Edit Acknowledgment Msg Queue'
         DSLFNT NAME=EKAAWQ,KEY1=(EKAAMSID,16,1),ROUTE=EKARTS,         *    [4]
               QUEUE=YES,THRESH=50,SPCMND=(ROU,DEL),                   *
               DESCR='MERVA Link Sample Ack Wait Queue'
         DSLFNT NAME=EKACMQ,ROUTE=EKARTS,                              *    [5]
               QUEUE=YES,THRESH=50,SPCMND=(ROU,DEL),KEY1=(EKAAMSUB,4), *
               DESCR='MERVA Link Sample Completed Message Queue'
         DSLFNT NAME=EKARMQ,ROUTE=EKARTS,                              *    [6]
               QUEUE=YES,THRESH=50,SPCMND=(ROU,DEL),KEY1=(EKAAMSUB,4), *
               DESCR='MERVA Link Sample Received Message Queue'
         DSLFNT NAME=EKAEMQ,ROUTE=EKARTS,                              *    [7]
               QUEUE=YES,THRESH=50,SPCMND=(ROU,DEL),KEY1=(EKAAMSUB,4), *
               DESCR='MERVA Link Sample Received Erroneous Msg Queue'
         DSLFNT NAME=EKADMY,QUEUE=DUMMY,                               *    [8]
                DESCR='MERVA Link Sample Dummy Queue'
*-------------------------------------------------------------------
*        MERVA LINK SAMPLE QUEUES FOR ASP A1I
*-------------------------------------------------------------------
         DSLFNT NAME=EKA1ICQ,KEY1=(EKACLASS,2,1),ROUTE=EKARTSCQ,       *    [9]
               QUEUE=YES,THRESH=50,SPCMND=(ROU,DEL),                   *
               DESCR='Application Control Queue of ASP A1I'
         DSLFNT NAME=EKA1IS1,TRAN=EKAS,QUEUE=YES A1I SEND QUEUE             [10]
```

Figure 16. Function Table Example for MERVA Link

Notes:

[1] The function MSC for the MERVA System Control Facility must be defined in the MERVA ESA Function Table. It is not represented by a queue in the MERVA ESA queue data set. The MERVA System Control Facility program EKAEMSC is linked to the MERVA ESA End User Driver DSLEUD. The top frame of the MSC screens is defined in the MCB AC01. The PF key sets 41 and 42 of the PF key group 40 are used by MSC.

[2] The Edit Outgoing Message Queue (EKAEOM in this example) represents the application that generates the message to be sent to its partner application. The partner application is determined by the routing table EKARTS that is associated with this queue. As an alternative, a message can be directly routed to the MERVA Link send queue of a specific ASP (DSLFNT NEXT parameter).

[3] The Edit Acknowledgment Message Queue (EKAEAM in this example) represents the application that generates an acknowledgment message to be sent to its partner application. The partner application is, in this example, again determined by the routing table EKARTS that is associated with this queue.

[4] The ACK Wait Queue (EKAAWQ in this example) is defined only if confirmed outgoing messages must be correlated with an acknowledgment message. An ACK Wait Queue must be defined with the 16-character IAM Message Identifier as the message key. The IAM Message Identifier, contained in the MERVA Link control field EKAAMSID, is used to correlate an acknowledgment message with its reported application message. An ACK Wait Queue can be shared between MERVA Link applications.

[5] The Completed Message Queue (EKACMQ in this example) represents the application that processes messages that have been completely handled by the MERVA Link. A Completed Message Queue can be shared between MERVA Link applications.

[6] The Received Message Queue (EKARMQ in this example) represents the application that processes incoming messages. A Received Message Queue can be shared between MERVA Link applications.

[7] The Received Erroneous Message Queue (EKAEMQ in this example) represents the application that processes incoming messages that have been delivered with a nonzero delivery return code in the MERVA Link control field EKADELRC. A Received Erroneous Message Queue can be shared between MERVA Link applications.

[8] The dummy queue EKADMY is used to discard confirmed acknowledgment messages. You do not need to define a proprietary dummy queue for each application. A dummy queue can be shared between MERVA Link applications.

[9] The MERVA Link application control queue (EKA1ICQ in this example) is a resource owned by a MERVA Link application. You must define an application control queue for every ASP defined in the partner table. application control queues cannot be shared between ASPs.

Any MERVA Link application control queue must be defined with a 2-character key. The MERVA Link message class, which may be LC (last confirmed), IP (in process), or LR (last received) for a message in this queue, is used as the message key. The MERVA Link message class is contained in the MERVA Link control field EKACLASS.

A routing table must be associated with any application control queue. This routing table is used for MERVA Link internal purposes and to support specific application requirements.

[10] The Send Queue Cluster of a MERVA Link application can consist of one, two, or three MERVA ESA queues. All members of this cluster are owned by a specific MERVA Link ASP. They cannot be shared between ASPs.

The parameter RELATED=(...,...) should be used in the definitions of the members of a MERVA Link send queue cluster if it consists of two or three queues. This specification means that those two or three queues must be treated by MERVA ESA in several aspects as a single resource. Those aspects are, for example, that all three queues are set to HOLD status if one of them is set to HOLD status, and that the transaction associated with these queues (EKAS in this example) is not started if it is active for any of the three queues.

If the RELATED=(....,....) parameter is not specified for all members of a send queue cluster, the MERVA System Control Facility commands **hold** and **astart** have an effect only on the first send queue specified in the SENDQC parameter of the EKAPT TYPE=ASP macro.

The parameter TRAN= specifies the transaction code for the MERVA Link sending ASP. In the CICS environment it is a unique transaction code (EKAS, in the MERVA Link sample).

In the IMS environment, all members of a send queue cluster share one transaction name for the sending ASP. Two send queue clusters, however, must not share one transaction name for the sending ASP. Different transaction names for the sending ASP must be defined in IMS for every send queue cluster, this means for every MERVA Link ASP. All these transaction names, however, call for the same program, the sending Application Support Program EKAAS10.

Starting a Send Task with a CICS APPC Session

All MERVA Link send queues are defined in the MERVA ESA Function Table (DSLFNTT) with a transaction identifier of the MERVA Link send task in the TRAN parameter of the DSLFNT macro (sample transaction identifier is EKAS). The LTERM parameter of this macro provides an option to start the MERVA Link send task with an APPC session to the partner system as its principal facility.

If a MERVA Link send queue is defined in the DSLFNTT with **LTERM=sysid**, where **sysid** is the identifier of a remote system described in a CICS CONNECTION definition, the task started by CICS on behalf of that queue is already associated with an APPC session to that partner system. A task start request issued by MERVA ESA is honored by CICS as soon as any session is available for the task to be started.

A MERVA Link send task uses its principal facility APPC session (if available) instead of allocating its own session to the partner system.

A MERVA Link send task, which could not be started because of unavailable sessions to the partner system, is automatically started by CICS as soon as the partner system is active and at least one session is bound between the partner systems.

Starting a MERVA Link send task with an APPC session to its partner system may avoid an ASP in the CICS environment to become inoperable because of a missing link to the partner system.

In the IMS APPC environment, a MERVA Link send task cannot be started with an already allocated APPC session. In this environment, task management and data communication is performed by different subsystems (IMS and APPC/MVS) which cannot cooperate in this way.

Processing a New or Changed Function Table

A function table is a sequence of the following types of DSLFNT macros:

- TYPE=INITIAL
- TYPE=ENTRY
- TYPE=FINAL

When you modify the sample function table (DSLFNTT), or when you create your own, you can do this without using the DSLGEN process. If you create your own function table, to avoid confusion, give it a name other than DSLFNTT.

Do not change the copy members DSLFNTTC, DWSFNTTC, ENLFNTTC, EKAFNTTC, and EKAFNTSC, because these are maintained by IBM. When a

program temporary fix (PTF) is applied to any of these copy members, any changes made by you will be overwritten.

When installing a new or changed function table, do the following:

1. If the name of the function table is not DSLFNTT, specify its name (it is the label of the first macro of the function table) as the value of the parameter FNT in the macro DSLPARM of the module DSLPRM.
2. Assemble and link-edit the new or changed function table.

Defining Message Paths within MERVA ESA

MERVA ESA uses routing tables to define the paths used by messages within a MERVA ESA installation. They define the path of a message from data entry through transmission to a network, or from reception of a message from a network to the execution of the transaction requested by the message.

The routing tables are tables of nonexecutable code. When the routing service is called by the MERVA ESA Queue Management program (DSLQMGT) in a DSLQMG TYPE=ROUTE request, the routing table is scanned, appropriate action taken, and a target list of up to 12 functions is established.

Examples of routing tables are supplied with the SWIFT Link, the Telex Link and the MERVA Link distributed material, a few of which are presented later in this book. They are intended only as examples, and should be changed to reflect the requirements of your installation.

Types of DSLROUTE Macro Calls

A routing table consists of a sequence of DSLROUTE macros. Details of the DSLROUTE macro are given in the *MERVA for ESA Macro Reference*.

The purpose of a DSLROUTE macro is determined by its type. The following is a list of the different purposes of the DSLROUTE macro:

- Defines variable fields for further use from a literal or a TOF field
- Compares the contents of a variable field with the contents of another variable field or literal
- Sets a target name in the target list from variable fields or literals or both
- Drops one or more variable fields if they are no longer needed
- Sets a target name in the last DSLROUTE macro, if an error occurs during the processing of the routing table

The first DSLROUTE macro generates the initial statements, and the last instruction generates the indication for the end of the routing table.

The first DSLROUTE macro must have a symbolic name according to assembler language conventions. The table is cataloged under this name and referred to by the ROUTE parameter of the DSLFNT macro in the MERVA ESA function table, and by the ROUTIN, ROUTOUT and ROUTSK parameters of the DWSLT macro in the SWIFT Link logical terminal table.

If MERVA ESA operates under MVS™, the last macro must be followed by the assembler statement END, which must be the last physical source statement.

If a macro refers to a field defined by another macro, you must ensure that the current definition is correct.

Routing criteria defined from TOF fields either relate to a message component or to special TOF fields that are provided for routing purposes. Refer to *MERVA for ESA Concepts and Components* for details.

Coding Considerations

The following points should be taken into account when coding routing tables:

- If a TOF field is accessed in a DSLROUTE TYPE=DEFINE macro, the EMPTY label is used to process the routing table when the field is empty, or considering the DISP parameter, no data is available at the specified displacement.
- If a TOF field is accessed in a DSLROUTE TYPE=DEFINE macro, and the data of the field is not long enough for the specified or default LENGTH parameter, the available data is taken and the FOUND label is used in the processing of the routing table.
- If a TOF field is accessed in a DSLROUTE TYPE=DEFINE macro, and the data of the field is longer than for the specified or default LENGTH parameter, only the requested length is taken.

  Note: The maximum length of a variable field is 32 bytes.
- If a TOF field is accessed in a DSLROUTE TYPE=DEFINE macro, and the access to the TOF field is not successful, the NOTFND label is taken. The processing of the routing table is not stopped.
- All variable field names used in a DSLROUTE TYPE=TEST or DSLROUTE TYPE=SET macro must be defined successfully with a DSLROUTE TYPE=DEFINE macro before its use. If this is not done, processing of the routing table continues with the NOTFND label.
- In the DSLROUTE TYPE=TEST macro, the FALSE exit is used if a field to be compared has a data length of zero, as no condition can be true.
- For AMOUNT test, the fields (field and literal) are adjusted with the decimal comma position and padded with zeros. If an overflow occurs during this adjustment, the FALSE label is taken for further processing of the routing table.
- For SHORT test, the fields (field and literal) are compared in the length of the shorter operand, the rest of the longer operand is ignored.
- For LONG test, the fields (field and literal) are compared in the length of the longer operand, the shorter operand is padded with binary zeros.
- If the DSLROUTE TYPE=SET macro only refers to fields with a data length of zero, the target field remains empty and is used by a following DSLROUTE TYPE=SET macro if any. When the target name is filled in, eight characters are used. If less than eight characters are used, the target name is padded with blanks. If more than eight characters are used, the target name is truncated. The target is verified in the function table. If the target is not found in the function table, the NOTFND label is used to continue processing. The target field is then cleared and reused in a subsequent DSLROUTE TYPE=SET macro.
- If all twelve target function fields are filled and another DSLROUTE TYPE=SET macro is found, it is ignored; that is, the already existing target functions are finally used.
- If an error is detected that makes further processing of the routing table impossible, the NEXT function of the function table entry is taken. If the NEXT function is not available, the target function of the DSLROUTE TYPE=FINAL is taken. If one of these is found, the return code of DSLRTNSC is 4 (successful with a warning). If neither of them is found, the return code of DSLRTNSC is 8 (not successful).

A complete list of the SWIFT message fields and header fields is contained in the *MERVA for ESA Messages and Codes*.

Note: When coding a routing table:

- SWIFT output messages should not be routed to functions used to process SWIFT input messages, nor to the SWIFT Link ready queues.
- Input messages should not be routed to output functions.
- Messages can be routed from one message-processing function to up to 12 queues for other message-processing functions.
- If you detect errors during the processing of routing tables or results that you do not understand, use the MERVA ESA routing trace facility to help find the error. The entries of the routing trace are described in *MERVA for ESA Concepts and Components*.

The routing table that routes messages to a SWIFT Link ready queue can include testing of the data supplied with the **ok** or **route** command. The user enters a character string (up to eight alphameric characters where any characters are permitted) with the **ok** or **route** command. The character string shows the routing table that the message is correct and can be routed to the ready queue. The TOF field MSGOK contains the data entered with this command. The contents of the TOF field MSGOK are erased after the message is reread from a queue by DSLEMSG.

As in any assembler language module, comments can be included in the table. With the exception of SPACE, EJECT, TITLE, and comments, only DSLROUTE macro calls can be included in the routing table.

Using Indices in Field Definitions

When defining fields in SWIFT messages you should be aware of the indices used in the FIELD parameter of a DSLROUTE TYPE=DEFINE statement (see *MERVA for ESA Macro Reference*).

If a field occurs only once in a message or the first occurrence in the message is requested, the modifier VFIRST can be used to access the required field. In this case the indices in the FIELD parameter can be left empty.

If you want to address a field explicitly, you must follow these rules:

- The nesting identifier of a field in a SWIFT message is usually 1. The index is 2, or higher depending on the nesting depth, when fields in embedded messages are referenced. For the SWIFT system message types 021 and 023 the header of the embedded message (basic header and application header) is stored in the field SWS21H on nesting identifier 1.
- The field group index of the basic header or of a subfield of the basic header (SWBH) is always 1. The field group index of the application header or of a subfield of an application header (SWAH) is always 2. The subfields of the basic and application headers are listed in *MERVA for ESA Messages and Codes*.

  SWIFT I header fields and message formats are still available in MERVA ESA but the SWIFT I network link support has been dropped. Header fields used in SWIFT I messages are handled as subfields of the basic or application header.

  The field group index of a field for the user header is always 3.

  The field group index of a trailer field (SWTRAIL or SWTRL) is always 255.

  The field group index for other message fields depends on the message type and is defined in the message device description of the MCB for the message type.

The numbering starts with field group index 5 and is incremented by one for each DSLLGRP statement in the message device description.

- The repeatable sequence index must be specified when a field within a repeatable sequence in one of the repetitions should be accessed. Fields in nested repeatable sequences with an index higher than one cannot be accessed.
- The data area index must be specified for fields that have more than one line and a line other than the first one should be accessed.

The *MERVA for ESA Macro Reference* contains a detailed description of the FIELD parameter.

Using SWIFT Fields in Routing Modules

The naming convention for SWIFT header and trailer fields in messages in tokenized format is:

SWBHxxxx Fields in the basic header

SWAHxxxx Fields in the application header

SWnnn System fields identified by a number (nnn)

SWTRAIL The trailer containing multiple data areas

A complete list of the SWIFT message fields and header fields is contained in the *MERVA for ESA Messages and Codes*. The copy book DWSFDTTC for the field definition table contains the definitions for these fields; this copy book can be found in the macro library.

If a message is stored in an external line format, the fields of the message are not tokenized. Therefore, they cannot be accessed by their name.

MERVA ESA provides special subfields to allow the routing of SWIFT messages in external line format depending on the contents of the header or trailer fields. The data of a header subfield with the name SWxxxxxx can be extracted by using the field name EFxxxxxx. The data areas of the trailer can be accessed using the field name EFTRAIL. When the external line format is used, the fields of the user header (block 3) and of the message text cannot be accessed by name.

The routing table in Figure 17 on page 58 shows how to identify whether a message is in external line format and how to extract the relevant field data. The message type for the external line format is ELF, with a prefix for the network. The prefix is either ‘S’ for SWIFT Link or ‘0’ for the MERVA ESA base function.

```
*--------------------------------------------------------------------
* Determine whether external format or TOF format is used
*--------------------------------------------------------------------
DWSL1IN  DSLROUTE TYPE=DEFINE,FIELD=(MSGID,DSLEXIT,,,,,VFIRST)
         DSLROUTE TYPE=TEST,COND=(MSGID,'SELF    ',EQ),TRUE=EF
         DSLROUTE TYPE=TEST,COND=(MSGID,'0ELF    ',EQ),FALSE=SWI
*--------------------------------------------------------------------
* External Line Format Message Processing
*  Check Buffer for valid SWIFT Message Format
*  Define the SWIFT fields used in the routing module
EF       DSLROUTE TYPE=DEFINE,FIELD=(BASHEAD,EFBH,,,,,VFIRST)
         DSLROUTE TYPE=DEFINE,FIELD=(BLOCK1,DSLELF,,,,,VFIRST),        *
               DISP=0,LENGTH=3,EMPTY=VE0,NOTFND=VE0
         DSLROUTE TYPE=TEST,COND=(BLOCK1,'{1:',EQ),FALSE=VE0
*        DEFINE THE APDU IDENTIFIER
         DSLROUTE TYPE=DEFINE,FIELD=(APDUID,EFBHAPDU,,,,,VFIRST)
* Check for Block 2; if not there, it is system message
         DSLROUTE TYPE=DEFINE,FIELD=(BLOCK2,DSLELF,,,,,VFIRST),        *
               DISP=29,LENGTH=3,EMPTY=PR0,NOTFND=PR0
         DSLROUTE TYPE=TEST,COND=(BLOCK2,'{2:',EQ),FALSE=PR0
         DSLROUTE TYPE=DEFINE,FIELD=(MSGTYPE,EFAHMT,,,,,VFIRST),       *
               EMPTY=PR0,NOTFND=PR0
         DSLROUTE TYPE=DEFINE,FIELD=(APPL,EFBHAPI,,,,,VFIRST),         *
               FOUND=ALL,NOTFND=ALL,EMPTY=ALL
*--------------------------------------------------------------------
* SWIFT Tokenized Message Processing
*        DEFINE THE BASIC HEADER FOR THE ROUTING TRACE
SWI      DSLROUTE TYPE=DEFINE,FIELD=(BASHEAD,SWBH,,,,,VFIRST)
*
*        DEFINE THE APDU IDENTIFIER
         DSLROUTE TYPE=DEFINE,FIELD=(APDUID,SWBHAPDU,,,,,VFIRST),      *
               EMPTY=VE0
         DSLROUTE TYPE=DEFINE,FIELD=(MSGTYPE,SWAHMT,,,,,VFIRST),       *
               EMPTY=PR0,NOTFND=PR0
         DSLROUTE TYPE=DEFINE,FIELD=(APPL,SWBHAPI,,,,,VFIRST)
*--------------------------------------------------------------------
*        APDU ID 01 contains an application header with a message type
*        all other APDU IDs are generated by DWSDGPA and are routed
*        to L1PR0 where they are printed in sequence with their ACKs.
ALL      DSLROUTE TYPE=TEST,COND=(APDUID,'01',EQ),FALSE=PR0
           .
           .
           .
```

Figure 17. Routing Sample for Messages in External Line Format

Examples of Routing Tables for the SWIFT Link

This section shows three examples of routing tables used for the SWIFT Link to the SWIFT network. These examples are used in the second example of a bank's organizational structure, which is described in detail in "Function Table Entries for Example 2" on page 19 and "Routing Logic of Example 2" on page 21. This example uses the following routing tables:

- For input messages: DWSL2IN, DWSL2DE0, DWSL2RE0, DWSL2VE0, DWSL2AI0
- For output messages: DWSL2OUT, DWSL2AO0, DWSL2DO0

Routing Table DWSL2AI0

The following shows routing table DWSL2AI0 after message authorization:

```
DWSL2AI0 DSLROUTE TYPE=DEFINE,FIELD=(FUNC,MSGTRFUN,,,,,VFIRST,LASTDA), * [1]
               EMPTY=ERROR,NOTFND=ERROR
         DSLROUTE TYPE=DEFINE,FIELD=(OK,MSGOK,,,,,VFIRST),             *
               EMPTY=AI0,NOTFND=AI0
         DSLROUTE TYPE=TEST,COND=(OK,'NO',EQ),TRUE=VE0
         DSLROUTE TYPE=TEST,COND=(OK,'YES',EQ),FALSE=AI0
         DSLROUTE TYPE=TEST,COND=(FUNC,'L2AI0',EQ,SHORT),TRUE=CUR        [2]
         DSLROUTE TYPE=TEST,COND=(FUNC,'L2AI2',EQ,SHORT),TRUE=APPLIC,  *
               FALSE=ERROR
CUR      DSLROUTE TYPE=DEFINE,FIELD=(CURRY,SW32CUR,,,,,VFIRST),        * [3]
               EMPTY=APPLIC,NOTFND=APPLIC
         DSLROUTE TYPE=TEST,COND=(CURRY,'USD',EQ),FALSE=APPLIC
         DSLROUTE TYPE=DEFINE,FIELD=(AMNT,SW32AMNT,,,,,VFIRST),        *
               EMPTY=APPLIC,NOTFND=APPLIC
         DSLROUTE TYPE=TEST,COND=(AMNT,'10000,',GT,AMOUNT),            *
               TRUE=AI2
APPLIC   DSLROUTE TYPE=DEFINE,FIELD=(APPL,SWBHAPI,,,,,VFIRST),         * [4]
               EMPTY=VE0,NOTFND=VE0
         DSLROUTE TYPE=TEST,COND=(APPL,'F',EQ),TRUE=FIN
         DSLROUTE TYPE=TEST,COND=(APPL,'A',EQ),TRUE=GPA
         DSLROUTE TYPE=TEST,COND=(APPL,'L',EQ),TRUE=GPA,FALSE=VE0
GPA      DSLROUTE TYPE=SET,TARGET=('L2RGPAU'),GOTO=END
FIN      DSLROUTE TYPE=DEFINE,FIELD=(PRTY,SWAHIPY,,,,,VFIRST),         * [5]
               EMPTY=VE0,NOTFND=VE0
         DSLROUTE TYPE=TEST,COND=(PRTY,'N',EQ),TRUE=FINN                 [6]
         DSLROUTE TYPE=TEST,COND=(PRTY,'U',EQ),TRUE=FINU                 [7]
         DSLROUTE TYPE=TEST,COND=(PRTY,'S',EQ),TRUE=FINU,FALSE=VE0       [8]
FINN     DSLROUTE TYPE=SET,TARGET=('L2RFINN'),GOTO=END
FINU     DSLROUTE TYPE=SET,TARGET=('L2RFINU'),GOTO=END
AI0      DSLROUTE TYPE=SET,TARGET=(FUNC),GOTO=END
AI2      DSLROUTE TYPE=SET,TARGET=('L2AI2'),GOTO=END
VE0      DSLROUTE TYPE=SET,TARGET=('L2VE0'),GOTO=END
ERROR    DSLROUTE TYPE=SET,TARGET=('L2ERROR')
         DSLROUTE TYPE=SET,TARGET=('L2PR0'),GOTO=END
END      DSLROUTE TYPE=FINAL,TARGET='L2VE0'
         END
```

Notes:

[1] This statement names the routing table with the label DWSL2AI0. The variable field FUNC is defined from the TOF field MSGTRFUN with the DSLTSV modifiers VFIRST and LASTDA. The last data area of this field contains the last function processed.

If the last function is not found empty processing is continued with the label ERROR and the message is routed to the error function L2ERROR.

[2] The following statements define the variable field OK from the TOF field MSGOK (this field contains the parameter of the **ok** command, or the command word of the **yes** or **no** commands), and test it. If the MSGOK field is not found or contains an incorrect value, the message is routed back to the same function at label AI0. If the MSGOK field contains NO, the message must be corrected in the function L2VE0, if it contains YES, further processing depends on the input function (L2AI0 or L2AI2).

[3] The following two statements define the variable field CURRY from the TOF field SW32CUR and test it whether it is the currency for US dollar (USD) or not. Only when the currency is US dollar, then the amount field is checked whether it is greater than 10000. If this is the case processing continues at label AI2 and the message is routed to the second authorization function L2AI2.

[4] The following statements define the variable field APPL from the application identifier in the SWIFT basic header. This one-character

identifier is 'F' for a financial application, 'A' for a General Purpose Application APC message, and 'L' for a General Purpose Application LTC message. For General Purpose Application messages processing is continued with label GPA, where the message is routed to the ready queue L2RGPAU.

[5] The following statements define the variable field PRTY from the priority in the SWIFT application header. The priority for SWIFT messages can be one of the following:

N Normal priority messages

U Urgent priority messages

S System messages

[6] When the priority of the message is N, processing is continued with label FINN, where the message is routed to the ready queue L2RFINN.

[7] When the priority of the message is U, processing is continued with label FINU, where the message is routed to the ready queue L2RFINU.

[8] When the priority of the message is S, processing is also continued with label FINU. When the priority is neither N, U, or S, the priority field in the SWIFT application header is incorrect and the processing continues with label VE0, where the message is routed to the verification function L2VE0.

Routing Table DWSL2IN

The routing table DWSL2IN is used for routing of SWIFT input messages:

```
DWSL2IN  DSLROUTE TYPE=DEFINE,FIELD=(BASHEAD,SWBH,,,,,VFIRST)              [1]
         DSLROUTE TYPE=DEFINE,FIELD=(APDUID,SWBHAPDU,,,,,VFIRST),      *  [2]
               EMPTY=VE0
         DSLROUTE TYPE=TEST,COND=(APDUID,'01',EQ),FALSE=PR0            *  [3]
         DSLROUTE TYPE=DEFINE,FIELD=(MSGTYPE,SWAHMT,,,,,VFIRST),       *  [4]
               EMPTY=PR0,NOTFND=PR0
         DSLROUTE TYPE=TEST,COND=(MSGTYPE,'96',EQ,SHORT),TRUE=SEND          [5]
         DSLROUTE TYPE=TEST,COND=(MSGTYPE,'074',EQ),TRUE=MT074              [6]
         DSLROUTE TYPE=TEST,COND=(MSGTYPE,'075',EQ),TRUE=SEND               [7]
         DSLROUTE TYPE=TEST,COND=(MSGTYPE,'085',EQ),TRUE=SEND
         DSLROUTE TYPE=TEST,COND=(MSGTYPE,'090',EQ),FALSE=MSGACK            [8]
         DSLROUTE TYPE=DEFINE,FIELD=(SW311,SW311,,,,1,VFIRST),         *  [9]
               EMPTY=PR0,NOTFND=PR0
         DSLROUTE TYPE=TEST,TRUE=SEND,                                 *
               COND=(SW311,'/REQUEST FOR CERTIFICATE BLACKLI',EQ)
         DSLROUTE TYPE=TEST,TRUE=SEND,                                 *
               COND=(SW311,'/BLACKLIST SCR REQUEST/',EQ)
         DSLROUTE TYPE=TEST,TRUE=SEND,                                 *
               COND=(SW311,'/BLACKLIST UKMO CARD REQUEST/',EQ)
         DSLROUTE TYPE=TEST,TRUE=SEND,                                 *
               COND=(SW311,'/UPDATE WHITELIST FLAG REQUEST',EQ)
         DSLROUTE TYPE=TEST,TRUE=SEND,                                 *
               COND=(SW311,'/ACTIVATE ICC SET REQUEST/',EQ)
         DSLROUTE TYPE=TEST,TRUE=SEND,                                 *
               COND=(SW311,'/CHANGE ACCESS TECHNOLOGY REQUES',EQ)
         DSLROUTE TYPE=TEST,TRUE=SEND,                                 *
               COND=(SW311,'/INCREMENT ICC KERNEL VERSION RE',EQ)
         DSLROUTE TYPE=TEST,TRUE=SEND,FALSE=MSGACK,                    * [10]
               COND=(SW311,'/DELETE ICC SET REQUEST/',EQ)
MT074    DSLROUTE TYPE=DEFINE,FIELD=(SW312,SW312,,,,1,VFIRST),         * [11]
               EMPTY=PR0,NOTFND=PR0
         DSLROUTE TYPE=TEST,TRUE=SEND,FALSE=MSGACK,                    *
               COND=(SW312,'/20/REVOKED CERTIFICATES',EQ)
SEND     DSLROUTE TYPE=SET,TARGET=('TX2USESQ')                           [12]
MSGACK   DSLROUTE TYPE=DEFINE,FIELD=(MSGACK,MSGACK,,,,,VFIRST),        * [13]
               NOTFND=VE0
         DSLROUTE TYPE=TEST,COND=(MSGACK,'DSL',EQ,SHORT),TRUE=VE0
```

```
         DSLROUTE TYPE=TEST,COND=(MSGACK,'DWS',EQ,SHORT),TRUE=VE0
         DSLROUTE TYPE=TEST,COND=(MSGACK,'{1:',EQ,SHORT),FALSE=VE0       [14]
         DSLROUTE TYPE=DEFINE,FIELD=(F451,MSGACK,,,,,VFIRST),          * [15]
               DISP=53,LENGTH=1,EMPTY=VE0,NOTFND=VE0
         DSLROUTE TYPE=TEST,COND=(F451,'0',EQ),FALSE=VE0                 [16]
         DSLROUTE TYPE=DEFINE,FIELD=(APPL,SWBHAPI,,,,,VFIRST)           [17]
         DSLROUTE TYPE=TEST,COND=(APPL,'F',EQ),TRUE=ACK,FALSE=PR0        [18]
PR0      DSLROUTE TYPE=SET,TARGET=('L2PR0'),GOTO=END
ACK      DSLROUTE TYPE=SET,TARGET=('L2ACK'),GOTO=END
VE0      DSLROUTE TYPE=SET,TARGET=('L2VE0'),GOTO=END
END      DSLROUTE TYPE=FINAL,TARGET='L2VE0'
         END
```

Notes:

[1] This statement names the routing table with the label DWSL2IN. The variable field BASHEAD is defined from the TOF field SWBH which is the SWIFT basic header. This field is not used in the further processing; this definition is only made to identify the message in the routing trace, when this facility of MERVA ESA is used.

[2] With this statement the variable field APDUID is defined from the TOF field SWBHAPDU which contains the SWIFT APDU identifier of the basic header. If this information cannot be found, it is assumed that the message is not a correct SWIFT message and processing is continued at label VE0 where the message is routed to the verification function.

[3] The APDU identifier defines the type of the message; when it is not '01' it is not an application message, and the processing continues with label PR0 where the message is routed to the hardcopy function L2PR0.

[4] With this statement the variable field MSGTYPE is defined from the TOF field SWAHMT which contains the message type of an APDU 01. If this information cannot be found, processing is continued at label PR0 where the message is routed to the hardcopy function L2PR0.

[5] If the message type is 96n, that is, one of the message types used for the SWIFT bilateral key exchange (BKE), processing continues at label SEND.

[6] If the message type is 074, that is, one of the message types used for BKE, processing continues at the label MT074.

[7] If the message type is 075 or 085, that is, one of the message types used for BKE, processing continues at the label SEND.

[8] If the message type is **not** 090, it is a message type that is **not** used for BKE, and processing continues at the label MSGACK.

[9] A message type 090 was found, and the variable field SW311 is defined from the field 311. Tests on the text of field 311 follow to find out if it is a message type 090 for the SWIFT USE functions, if one is found, processing continues at the label SEND. As a routing variable field can only hold 32 characters, some of contents of the first line of field 311 are not shown completely in these tests, but all contents are unique in these 32 characters.

[10] This is the last test on field 311 of a message type 090, the FALSE parameter indicates that processing continues at the label MSGACK for all message types 090 that are not used by the SWIFT USE functions.

[11] At the label MT074, the variable field SW312 is defined from the field 312 of the message type 074. A test on the text of field 312 follows to find out

if it is a message type 074 for the SWIFT USE functions, if so, processing continues at the label SEND, if not, processing continues at the label MSGACK.

[12] At the label SEND, all messages that have come from the MERVA OS/2 that processes the SWIFT USE functions, and have been sent to the SWIFT network, and have been positively or negatively acknowledged, or have not been sent because of an error, are routed to the function TX2USESQ, from which MERVA Link will send them back to MERVA OS/2. Then processing continues at the next statement with the label MSGACK, that is, all these messages are now treated the same way as all the other messages.

[13] At the label MSGACK, the variable field MSGACK is defined from the TOF field MSGACK which contains the SWIFT acknowledgment message, when the message has been sent to the SWIFT network, or a MERVA ESA error message starting with the characters DSL or DWS. In the latter case the message has not been sent to the SWIFT network. This is tested in the two statements following the definition of the MSGACK field. If DSL or DWS is found, processing continues at the label VE0.

[14] When the MSGACK field starts with '{1:', the MSGACK field contains a SWIFT acknowledgment message (APDU 21). In all other cases processing is continued at label VE0 and the message is routed to the verification function.

[15] This statement defines the variable field F451 from the SWIFT field 451 of the APDU 21. This field contains the Acceptance/Rejection indicator. The data of this field is always at a fixed position in the APDU 21. By the DISP=53 parameter data is used from the displacement 53 of the TOF field MSGACK.

[16] If the variable field F451, that is, the SWIFT field 451, does **not** contain '0', the message has not been acknowledged positively and is routed to the verification function L2VE0. If the field F451 contains '0', processing continues with the next statement.

[17] This statement defines the variable field APPL from the application identifier in the SWIFT basic header. This identifier is 'F' for a message of the FIN application, 'A' for an APC message of the General Purpose Application, and 'L' for an LTC message of the General Purpose Application.

[18] The field APPL is tested for containing 'F'. If the condition is true, it is a financial message that is routed to the acknowledgment function L2ACK at label ACK. If the condition is false, it is a General Purpose Application message that is routed to the hardcopy function L2PR0 at label PR0.

Routing Table DWSL2OUT

The routing table DWSL2OUT is used for messages from SWIFT

```
DWSL2OUT DSLROUTE TYPE=DEFINE,FIELD=(BASHEAD,SWBH,,,,,VFIRST)             [1]
         DSLROUTE TYPE=DEFINE,FIELD=(MSGID,DSLEXIT,,,,,VFIRST)           [2]
         DSLROUTE TYPE=TEST,COND=(MSGID,'0DSL    ',EQ),TRUE=FREE
         DSLROUTE TYPE=DEFINE,FIELD=(ERROR,DSLLFBUF,,,,,VFIRST),       *  [3]
               FOUND=FREE
         DSLROUTE TYPE=TEST,COND=(MSGID,'SG22    ',EQ),TRUE=PR0           [4]
         DSLROUTE TYPE=TEST,COND=(MSGID,'SG23    ',EQ),TRUE=PR0
         DSLROUTE TYPE=TEST,COND=(MSGID,'SG42    ',EQ),TRUE=PR0
         DSLROUTE TYPE=TEST,COND=(MSGID,'SG43    ',EQ),TRUE=PR0
         DSLROUTE TYPE=TEST,COND=(MSGID,'SG26    ',EQ),TRUE=PR0
         DSLROUTE TYPE=TEST,COND=(MSGID,'SF25    ',EQ),TRUE=PR0
         DSLROUTE TYPE=TEST,COND=(MSGID,'SG13    ',EQ),TRUE=PR0
         DSLROUTE TYPE=TEST,COND=(MSGID,'SG15    ',EQ),TRUE=PR0
```

```
         DSLROUTE TYPE=DEFINE,FIELD=(SUFFIX,' ')                          [5]
         DSLROUTE TYPE=DEFINE,FIELD=(LT,SWBHLT,,,,,VFIRST),           *   [6]
               LENGTH=9,NOTFND=PR1,EMPTY=PR1
         DSLROUTE TYPE=TEST,COND=(LT,'VNDOBET2A',EQ),TRUE=REPORT
         DSLROUTE TYPE=TEST,COND=(LT,'VNDOSYN2A',EQ),TRUE=SYNONYM,    *
               FALSE=PR1
SYNONYM  DSLROUTE TYPE=DEFINE,FIELD=(SUFFIX,'S')
REPORT   DSLROUTE TYPE=TEST,COND=(MSGID,'SF010   ',EQ),TRUE=DO0           [7]
         DSLROUTE TYPE=TEST,COND=(MSGID,'SF011   ',EQ),TRUE=DO0
         DSLROUTE TYPE=TEST,COND=(MSGID,'SF015   ',EQ),TRUE=DO0
         DSLROUTE TYPE=TEST,COND=(MSGID,'SF066   ',EQ),TRUE=DO0
         DSLROUTE TYPE=TEST,COND=(MSGID,'SF082   ',EQ),TRUE=DO0
         DSLROUTE TYPE=TEST,COND=(MSGID,'SF083   ',EQ),TRUE=DO0
         DSLROUTE TYPE=DEFINE,FIELD=(MSGTYPE,SWAHMT,,,,,VFIRST),      *   [8]
               EMPTY=PR1,NOTFND=PR1
         DSLROUTE TYPE=TEST,COND=(MSGTYPE,'021',EQ),TRUE=DO0
         DSLROUTE TYPE=TEST,COND=(MSGTYPE,'0',EQ,SHORT),FALSE=AUT
         DSLROUTE TYPE=TEST,COND=(MSGTYPE,'076',EQ),TRUE=SEND             [9]
         DSLROUTE TYPE=TEST,COND=(MSGTYPE,'077',EQ),TRUE=SEND
         DSLROUTE TYPE=TEST,COND=(MSGTYPE,'087',EQ),TRUE=SEND
         DSLROUTE TYPE=TEST,COND=(MSGTYPE,'092',EQ),FALSE=PR1            [10]
             DSLROUTE TYPE=DEFINE,FIELD=(SW311,SW311,,,,1,VFIRST),    *  [11]
               EMPTY=PR1,NOTFND=PR1
         DSLROUTE TYPE=TEST,TRUE=SEND,                                *
               COND=(SW311,'/CERTIFICATE BLACKLIST DISTRIBUT',EQ)
         DSLROUTE TYPE=TEST,TRUE=SEND,                                *
               COND=(SW311,'/BLACKLIST SCR ACKNOWLEDGMENT/',EQ)
         DSLROUTE TYPE=TEST,TRUE=SEND,                                *
               COND=(SW311,'/BLACKLIST UKMO CARD ACKNOWLEDGE',EQ)
         DSLROUTE TYPE=TEST,TRUE=SEND,                                *
               COND=(SW311,'/UPDATE WHITELIST FLAG ACKNOWLED',EQ)
         DSLROUTE TYPE=TEST,TRUE=SEND,                                *
               COND=(SW311,'/ACTIVATE ICC SET ACKNOWLEDGMENT',EQ)
         DSLROUTE TYPE=TEST,TRUE=SEND,                                *
               COND=(SW311,'/CHANGE ACCESS TECHNOLOGY ACKNOW',EQ)
         DSLROUTE TYPE=TEST,TRUE=SEND,                                *
               COND=(SW311,'/INCREMENT ICC KERNEL VERSION AC',EQ)
         DSLROUTE TYPE=TEST,TRUE=SEND,                                *
               COND=(SW311,'/DELETE ICC SET ACKNOWLEDGMENT/',EQ)
         DSLROUTE TYPE=TEST,TRUE=SEND,                                *
               COND=(SW311,'/LSN/SSN THRESHOLD WARNING/',EQ)
         DSLROUTE TYPE=TEST,TRUE=SEND,                                *
               COND=(SW311,'/ICC KERNEL VERSION EXPIRY WARNI',EQ)
         DSLROUTE TYPE=TEST,TRUE=SEND,                                *
               COND=(SW311,'/ICC SET EXPIRY WARNING/',EQ)
         DSLROUTE TYPE=TEST,TRUE=SEND,                                *
               COND=(SW311,'/LT ACTIVATION REPORT/',EQ)
         DSLROUTE TYPE=TEST,TRUE=SEND,                                *
               COND=(SW311,'/PENULTIMATE WHITELIST FLAG WARN',EQ)
         DSLROUTE TYPE=TEST,TRUE=SEND,                                *
               COND=(SW311,'/FINAL WHITELIST FLAG WARNING/',EQ)
         DSLROUTE TYPE=TEST,TRUE=SEND,FALSE=PR1,                      *  [12]
               COND=(SW311,'/LAST ICC KERNEL VERSION WARNING',EQ)
AUT      DSLROUTE TYPE=DEFINE,FIELD=(AUT,MSGACK,,,,,VFIRST),          *  [13]
               DISP=0,LENGTH=6,EMPTY=AO0,NOTFND=AO0
         DSLROUTE TYPE=TEST,COND=(AUT,'DWS765',EQ),TRUE=MSGERR
         DSLROUTE TYPE=TEST,COND=(AUT,'DWS766',EQ),TRUE=MSGERR,       *
               FALSE=AO0
MSGERR   DSLROUTE TYPE=DEFINE,FIELD=(MSGERR,MSGTRERR,,,,,VFIRST,      *  [14]
               LASTDA),NOTFND=DO0,EMPTY=DO0
         DSLROUTE TYPE=TEST,COND=(MSGERR,'0000',EQ),FALSE=DO0
         DSLROUTE TYPE=TEST,COND=(MSGTYPE,'96',EQ,SHORT),TRUE=SEND       [15]
         DSLROUTE TYPE=SET,TARGET=('L2SDO',SUFFIX),GOTO=END              [16]
PR0      DSLROUTE TYPE=SET,TARGET=('L2PR0'),GOTO=END
PR1      DSLROUTE TYPE=SET,TARGET=('L2PR1',SUFFIX),GOTO=END
AO0      DSLROUTE TYPE=SET,TARGET=('L2AO0',SUFFIX),GOTO=END
FREE     DSLROUTE TYPE=SET,TARGET=('L2FREE'),GOTO=END
```

```
DO0      DSLROUTE TYPE=SET,TARGET=('L2DO0',SUFFIX),GOTO=END
SEND     DSLROUTE TYPE=SET,TARGET=('TX2USESQ'),GOTO=END                  [17]
END      DSLROUTE TYPE=FINAL,TARGET='L2PR1'
         END
```

Notes:

[1] This statement names the routing table with the label DWSL2OUT. The variable field BASHEAD is defined from the TOF field SWBH containing the SWIFT basic header of the message. This definition is only made to identify the message in the routing trace, when this facility of MERVA ESA is used.

[2] With this statement, the variable field MSGID is defined from the TOF field DSLEXIT containing the MERVA ESA message identification. All possible MERVA ESA message identifications are defined in the message type table DSLMTTT. When the DSLEXIT field contains '0DSL' the message could not be transformed to the TOF format with the appropriate message control block (MCB), because the message contained format errors. In this case the message is routed to the free format function L2FREE (TRUE=FREE).

[3] With this statement, the variable field ERROR is defined from the TOF field DSLLFBUF containing part of the message, if the message could not be transformed completely to the TOF format with the appropriate message control block (MCB). If this field is found in the TOF, the message is routed to the free format function L2FREE (FOUND=FREE).

[4] The following statements select specific acknowledgment system message types for routing to the hardcopy function L2PR0 (TRUE=PR0). There the acknowledgments are printed in sequence with the originating system messages. The message identifications selected are:

- SG22 Login Acknowledgment
- SG23 Select Acknowledgment
- SG42 Login Negative Acknowledgment
- SG43 Select Negative Acknowledgment
- SG26 Logout Acknowledgment
- SF25 Quit Acknowledgment for Financial Application
- SG13 System Abort Application Confirmation
- SG15 System Abort Logical Terminal Confirmation

[5] With this statement, the variable field SUFFIX is defined from a literal ' ' (blank). This suffix is used when setting target functions for the master logical terminal.

[6] With the following statements, the variable field LT is defined from the TOF field SWBHLT and tested for specific contents. The field SWBHLT contains the name of the logical terminal that receives the SWIFT output message, it is a subfield of the SWIFT basic header field SWBH. The LENGTH=9 parameters specifies the length of the data used from the TOF field SWBHLT.

The variable field LT is tested for the name of the master logical terminal (literal 'VNDOBET2A') and the name of the synonym logical terminal (literal 'VNDOSYN2A'). These two logical terminal names are also defined in the logical terminal table DWSLTT.

If the master logical terminal is found, processing continues at the label REPORT, for the synonym logical terminal, the field SUFFIX is redefined

with the literal 'S'. This suffix is used when setting target functions for the synonym logical terminal. If neither the name of the master nor synonym logical terminal is found, the message is routed to the hardcopy print function L2PR1.

[7] With the following statements the SWIFT reports for the FIN application are routed to the distribution function L2DO0. The reports are:

- SF010 Non Delivery Warning
- SF011 Delivery Notification
- SF015 Delayed NAK
- SF066 Undelivered Message Solicited Report
- SF082 Undelivered Message Report at Fixed Hour
- SF083 Undelivered Message Report at Cut-Off Time

[8] With the following statements, the variable field MSGTYPE is defined from the TOF field SWAHMT that contains the message type of the SWIFT application header. If the message type is not found, the message is routed to the hardcopy print function L2PR1.

A test is made for the message type 021 (retrieved message) that can contain a banking message. If the message type 021 is found, the message is routed to L2DO0.

Then a test for SWIFT system messages in general is done, that is a test for the first character being 0. If the first character is not 0, it is a banking message of the FIN application and processing continues at label AUT.

[9] A system message was found, and a test is made if it is one of the system messages used for the SWIFT USE functions. For the message types 076, 077 and 087, processing continues at label SEND.

[10] If the message type is **not** 092, it is a message type that is **not** used for BKE, and processing continues at the label PR1.

[11] A message type 092 was found, and the variable field SW311 is defined from the field 311. Tests on the text of field 311 follow to find out if it is a message type 092 for the SWIFT USE functions, if one is found, processing continues at the label SEND. As a routing variable field can only hold 32 characters, some of contents of the first line of field 311 are not shown completely in these tests, but all contents are unique in these 32 characters.

[12] This is the last test on field 311 of a message type 092, the FALSE parameter indicates that processing continues at the label PR1 for all message types 092 that are not used by the SWIFT USE functions.

[13] With the following statements, the variable field AUT is defined from the TOF field MSGACK that contains the result of the authentication as a diagnostic message. A test is made for the message identifications DWS765 (authentication successful with actual key) and DWS766 (message not to be authenticated). If one of these message identifications is found, processing continues at label MSGERR (TRUE parameter). If there is an authentication error, the message is routed to the authentication output function L2AO0.

[14] With the following statements, the variable field MSGERR is defined from the TOF field MSGTRERR (it contains the error code of the message trace field MSGTRACE). A test is made for 0000 which shows **no error**. If the MSGTRERR field is not found or empty or contains a different error code, the message is routed to the distribution output function L2DO0. Otherwise processing continues with the next statement.

[15] If the message type is 96n, that is, one of the message types used for the SWIFT bilateral key exchange (BKE), processing continues at label SEND.

[16] With the following statements, the targets are set to the functions L2SDO, L2PR0, L2PR1, L2AO0, L2FREE, and L2DO0 as determined by the conditions tested before. When setting the targets for L2SDO, L2AO0, L2DO0, and L2PR1, the appropriate literal is concatenated with the content of the field SUFFIX which was defined as ' ' (blank) for the master logical terminal and S for the synonym logical terminal.

[17] At the label SEND, the target is set for the function TX2USESQ for all messages used for the SWIFT USE functions and must therefore be sent to the USE workstation using MERVA Link.

A DSLROUTE TYPE=FINAL statement with the error target function L2PR1 is the last statement in this routing table.

Examples of Routing Tables for the Telex Link

The Telex Link supplies examples of routing tables that you can use to create and process telex messages. Some of these routing tables you can modify. For example, if you have defined more than one function for creating telex messages, you can change the path telex messages taken between functions.

Note: You cannot modify the routing paths necessary for the correct processing of the communication between the Telex Link and the Headoffice Telex on a fault-tolerant system. The Telex Link via workstation is described in *MERVA Workstation Based Functions*

The Telex Link supplies the following examples of routing tables:

ENLRTDE0 Routing after the data entry function TXDE0 and the sequential data set input function TXSDI

ENLRTVE0 Routing after the verification function for SWIFT messages TXVE0

ENLRTAI0 Routing after the authorization function for SWIFT messages TXAI0

ENLRTTKC Routing after the test-key calculation function TXTKC (also used for the test-key calculation error function TXTKCERR)

ENLRTTKV Routing after the test-key verification function TXTKV (also used for the test-key verification error function TXTKVERR)

ENLRTHCF Routing of telex messages sent to the Headoffice Telex on a fault-tolerant system and received from the Headoffice Telex on a fault-tolerant system

The routing tables ENLRTDE0, ENLRTTKC, and ENLRTHCF show the important routing criteria of the Telex Link. The routing module ENLRTTKV uses a similar decision as ENLRTTKC, and the routing modules ENLRTVE0 and ENLRTAI0 combine decisions important for the Telex Link (as shown in ENLRTDE0) and the SWIFT Link (DWSL2DE0).

Routing Table ENLRTDEO

The following shows the ENLRTDE0 routing table supplied with the Telex Link, after data entry:

```
ENLRTDE0 DSLROUTE TYPE=DEFINE,FIELD=(TXIND,ENLTXIND,,,,,VFIRST),       *  [1]
               LENGTH=1,NOTFND=NOTELEX,EMPTY=NOTELEX
         DSLROUTE TYPE=TEST,COND=(TXIND,'Y',EQ),FALSE=NOTELEX
         DSLROUTE TYPE=DEFINE,FIELD=(TKIND,ENLTKIND,,,,,VFIRST),       *  [2]
```

```
              LENGTH=3,NOTFND=TXERROR,EMPTY=TXERROR
         DSLROUTE TYPE=TEST,COND=(TKIND,'YES',EQ),TRUE=TKC
         DSLROUTE TYPE=DEFINE,FIELD=(TXTYPE,ENLTXPRI,,,,,VFIRST),      *  [3]
              LENGTH=1,NOTFND=TXERROR,EMPTY=TXERROR
         DSLROUTE TYPE=TEST,COND=(TXTYPE,'P',EQ),TRUE=NRM
         DSLROUTE TYPE=TEST,COND=(TXTYPE,'N',EQ),TRUE=NRM
         DSLROUTE TYPE=TEST,COND=(TXTYPE,'T',EQ),TRUE=NRM
         DSLROUTE TYPE=TEST,COND=(TXTYPE,'U',EQ),TRUE=URG,FALSE=TXERROR
TKC      DSLROUTE TYPE=SET,TARGET='TXTKC',GOTO=END                        [4]
URG      DSLROUTE TYPE=SET,TARGET='TXURG',GOTO=END
NRM      DSLROUTE TYPE=SET,TARGET='TXNRM',GOTO=END
NOTELEX  DSLROUTE TYPE=DEFINE,FIELD=(SWIFT,SWBH,,,,,VFIRST),           *  [5]
              FOUND=SWIFT
         DSLROUTE TYPE=SET,TARGET='TXNOTX',GOTO=END
SWIFT    DSLROUTE TYPE=SET,TARGET='TXAI0',GOTO=END
TXERROR  DSLROUTE TYPE=SET,TARGET='TXERROR',GOTO=END                      [6]
END      DSLROUTE TYPE=FINAL,TARGET='TXERROR'                             [7]
         END
```

Notes:

[1] This statement names the routing table with the label ENLRTDE0. A variable field with the name TXIND is defined from the TOF field ENLTXIND, which contains an indicator if the message is a telex message. VFIRST requests the use of the first appearance of ENLTXIND in the TOF. Only the first byte is requested, using the LENGTH=1 parameter.

If the field is not found or empty, processing is continued at the label NOTELEX (NOTFND=NOTELEX,EMPTY=NOTELEX).

If the field is found, the variable field TXIND is tested for the content “Y”, that is, the telex indicator is “YES”. If it is not “Y”, it is not a telex message, and processing is continued at the label NOTELEX (FALSE=NOTELEX).

[2] The message is a telex message. A variable field with the name TKIND is defined from the TOF field ENLTKIND. This field contains an indicator if the telex message requires a test key. Use of the VFIRST parameter indicates that the first appearance of ENLTKIND in the TOF is to be used. Three bytes are requested using the LENGTH=3 parameter.

If the field is not found or is empty, processing is continued at the label TXERROR (NOTFND=TXERROR,EMPTY=TXERROR), as each telex message must contain this field.

If the field is found, the variable field TKIND is tested for the contents “YES”, indicating that a test key is required, and processing is continued at the label TKC (TRUE=TKC).

[3] The telex message does not require a test key. A variable field with the name TXTYPE is defined from the TOF field ENLTXPRI which contains the telex type. VFIRST requests the use of the first appearance of ENLTXPRI in the TOF.

If the field is not found or is empty, processing is continued at the label TXERROR (NOTFND=TXERROR,EMPTY=TXERROR), as each telex message must contain this field.

If the field is found, the variable field TXTYPE is tested for the following contents:

P Indicating a print telex for the normal ready queue (TRUE=NRM)

N Indicating a normal priority telex for the normal ready queue (TRUE=NRM)

T Indicating a timed telex for the normal ready queue (TRUE=NRM)

U Indicating an urgent priority telex for the urgent ready queue (TRUE=URG)

If none of these values is found, processing is continued at the label TXERROR (FALSE=TXERROR), as one of the four values must be in the telex type field.

[4] With the following three statements, the targets are set to the functions TXTKC, TXURG, and TXNRM, as determined by the conditions tested before, and processing is continued at the label END (GOTO=END).

[5] The message is not a telex message. A variable field with the name SWIFT is defined from the TOF field SWBH, that if present, indicates that the message is a SWIFT message. SWBH is part of the SWIFT message header.

If the field is found, processing is continued at the label SWIFT (FOUND=SWIFT), where the target function TXAI0 is set.

If the field is not found, processing is continued with the next statement where the target function TXNOTX (neither telex message nor SWIFT message) is set.

[6] With this statement, the target is set to the function TXERROR when one of the previous DEFINE or TEST statements found an error.

[7] A DSLROUTE TYPE=FINAL statement with the error target function TXERROR is the last statement in this routing table.

Routing Table ENLRTTKC

The following shows the ENLRTTKC routing table supplied with the Telex Link, after test-key calculation:

```
ENLRTTKC DSLROUTE TYPE=DEFINE,FIELD=(TXIND,ENLTXIND,,,,,VFIRST),       *  [1]
               LENGTH=1,NOTFND=NOTELEX,EMPTY=NOTELEX
         DSLROUTE TYPE=TEST,COND=(TXIND,'Y',EQ),FALSE=NOTELEX
         DSLROUTE TYPE=DEFINE,FIELD=(TKFLAG,ENLTKFLG,,,,,VFIRST),      *  [2]
               NOTFND=TKCERR,EMPTY=TKCERR
         DSLROUTE TYPE=TEST,COND=(TKFLAG,'OK',EQ,SHORT),FALSE=TKCERR
         DSLROUTE TYPE=DEFINE,FIELD=(TXTYPE,ENLTXPRI,,,,,VFIRST),      *  [3]
               LENGTH=1,NOTFND=TXERROR,EMPTY=TXERROR
         DSLROUTE TYPE=TEST,COND=(TXTYPE,'P',EQ),TRUE=NRM
         DSLROUTE TYPE=TEST,COND=(TXTYPE,'N',EQ),TRUE=NRM
         DSLROUTE TYPE=TEST,COND=(TXTYPE,'T',EQ),TRUE=NRM
         DSLROUTE TYPE=TEST,COND=(TXTYPE,'U',EQ),TRUE=URG,FALSE=TXERROR
TKCERR   DSLROUTE TYPE=SET,TARGET='TXTKCERR',GOTO=END                      [4]
URG      DSLROUTE TYPE=SET,TARGET='TXURG',GOTO=END
NRM      DSLROUTE TYPE=SET,TARGET='TXNRM',GOTO=END
NOTELEX  DSLROUTE TYPE=DEFINE,FIELD=(SWIFT,SWBH,,,,,VFIRST),           *  [5]
               FOUND=SWIFT
         DSLROUTE TYPE=SET,TARGET='TXNOTX',GOTO=END
SWIFT    DSLROUTE TYPE=SET,TARGET='TXAI0',GOTO=END
TXERROR  DSLROUTE TYPE=SET,TARGET='TXERROR',GOTO=END                       [6]
END      DSLROUTE TYPE=FINAL,TARGET='TXERROR'                              [7]
         END
```

Notes:

[1] This statement names the routing table with the label ENLRTTKC. A variable field with the name TXIND is defined from the TOF field ENLTXIND, which contains an indicator if the message is still a telex

message. The VFIRST parameter specifies that the first appearance of ENLTXIND in the TOF field is to be used. Only the first byte is requested using the LENGTH=1 parameter.

If the field is not found or is empty, processing is continued at the label NOTELEX (NOTFND=NOTELEX,EMPTY=NOTELEX).

If the field is found, the variable field TXIND is tested for the content "Y", that is, the telex indicator is "YES". If it is not "Y", it is not a telex message, and processing is continued at the label NOTELEX (FALSE=NOTELEX).

[2] The message is a telex message. A variable field with the name TKFLAG is defined from the TOF field ENLTKFLG which contains the test-key flag that indicates whether the test key was successfully calculated. The VFIRST parameter specifies that the first appearance of ENLTKFLG in the TOF is to be used.

If the field is not found or is empty, processing is continued at the label TKCERR (NOTFND=TKCERR,EMPTY=TKCERR), as the test-key calculation was either not done or was not successful.

If the field is found, the variable field TKFLAG is tested for the content "OK" in the first two bytes (SHORT modifier). If "OK" is not found, processing is continued at the label TKCERR (FALSE=TKCERR).

[3] The test key was successfully calculated. A variable field with the name TXTYPE is defined from the TOF field ENLTXPRI which contains the telex type. VFIRST requests the use of the first appearance of ENLTXPRI in the TOF.

If the field is not found or is empty, processing is continued at the label TXERROR (NOTFND=TXERROR,EMPTY=TXERROR), as each telex message must contain this field.

If the field is found, the variable field TXTYPE is tested for the following contents:

P Indicating a print telex for the normal ready queue (TRUE=NRM)

N Indicating a normal priority telex for the normal ready queue (TRUE=NRM)

T Indicating a timed telex for the normal ready queue (TRUE=NRM)

U Indicating an urgent priority telex for the urgent ready queue (TRUE=URG)

If none of these values is found, processing is continued at the label TXERROR (FALSE=TXERROR), as one of the four values must be in the telex type field.

[4] With the following three statements the targets are set to the functions TXTKCERR, TXURG and TXNRM as determined by the conditions tested before, and processing is continued at the label END (GOTO=END).

[5] The message is not a telex message. A variable field with the name SWIFT is defined from the TOF field SWBH, that if present, indicates that the message is a SWIFT message. SWBH is part of the SWIFT message header.

If the field is found, processing is continued at the label SWIFT (FOUND=SWIFT), where the target function TXAI0 is set.

If the field is not found, processing is continued with the next statement where the target function TXNOTX (neither telex message nor SWIFT message) is set.

[6] With this statement, the target is set to the function TXERROR when one of the previous DEFINE or TEST statements found an error.

[7] A DSLROUTE TYPE=FINAL statement with the error target function TXERROR is the last statement in this routing table.

Routing Table ENLRTHCF

The routing table ENLRTHCF, shown below,, is defined in ENLPRM and is used for all routing that is necessary for messages sent to or received from the Headoffice Telex on a fault-tolerant system. The Telex Link provides information in the field ENLSTAMP that indicates the processing stage of a telex message.

```
ENLRTHCF DSLROUTE TYPE=DEFINE,FIELD=(TXSTAMP,ENLSTAMP),              * [1]
               LENGTH=8,NOTFND=ERROR,EMPTY=ERROR
         DSLROUTE TYPE=TEST,COND=(TXSTAMP,'ACKENLXM',EQ),TRUE=WAIT      [2]
         DSLROUTE TYPE=TEST,COND=(TXSTAMP,'NAKENLXM',EQ),TRUE=NAK       [3]
         DSLROUTE TYPE=TEST,COND=(TXSTAMP,'ACKXMIT ',EQ),TRUE=ACK       [4]
         DSLROUTE TYPE=TEST,COND=(TXSTAMP,'NAKXMIT ',EQ),TRUE=NAK       [5]
         DSLROUTE TYPE=TEST,COND=(TXSTAMP,'TELEXRCV',EQ),TRUE=RCV       [6]
         DSLROUTE TYPE=TEST,COND=(TXSTAMP,'DUPLRQNF',EQ),TRUE=PDR       [7]
         DSLROUTE TYPE=TEST,COND=(TXSTAMP,'ENLOUT  ',EQ),TRUE=CLEAN     [8]
         DSLROUTE TYPE=TEST,COND=(TXSTAMP,'ENLIN   ',EQ),TRUE=CLEAN     [9]
         DSLROUTE TYPE=TEST,COND=(TXSTAMP,'CLEANUP ',EQ),TRUE=CLEAN     [10]
         DSLROUTE TYPE=TEST,COND=(TXSTAMP,'INVALID ',EQ),TRUE=INVR      [11]
         DSLROUTE TYPE=TEST,COND=(TXSTAMP,'UNSXMACK',EQ),TRUE=INVR      [12]
ERROR    DSLROUTE TYPE=SET,TARGET=('TXERROR'),GOTO=END                  [13]
WAIT     DSLROUTE TYPE=SET,TARGET=('TXWAIT'),GOTO=END
ACK      DSLROUTE TYPE=SET,TARGET=('TXACK')
         DSLROUTE TYPE=SET,TARGET=('TXPR0'),GOTO=END
NAK      DSLROUTE TYPE=SET,TARGET=('TXNAK'),GOTO=END
RCV      DSLROUTE TYPE=SET,TARGET=('TXRCV')
         DSLROUTE TYPE=SET,TARGET=('TXSTPLR'),GOTO=END
PDR      DSLROUTE TYPE=SET,TARGET=('TXPDR')
         DSLROUTE TYPE=SET,TARGET=('TXSTPLR'),GOTO=END
CLEAN    DSLROUTE TYPE=SET,TARGET=('TXCLEAN'),GOTO=END
INVR     DSLROUTE TYPE=SET,TARGET=('TXINVR')
END      DSLROUTE TYPE=FINAL,TARGET='TXERROR'                           [14]
         END
```

Notes:

[1] This statement names the routing table with the label ENLRTHCF. A variable field with the name TXSTAMP is defined from the TOF field ENLSTAMP, which contains an indicator for the processing stage of the telex message. Eight bytes are requested using the LENGTH=8 parameter.

If the field is not found or is empty, processing is continued at the label ERROR (NOTFND=ERROR,EMPTY=ERROR).

[2] If the variable field TXSTAMP contains the value “ACKENLXM”, a positive logical acknowledgment was received from the Headoffice Telex on a fault-tolerant system for an outgoing telex message. Processing is continued at the label WAIT for setting the target function TXWAIT.

[3] If the variable field TXSTAMP contains the value “NAKENLXM”, a negative logical acknowledgment was received from the Headoffice Telex on a fault-tolerant system for an outgoing telex message. Processing is continued at the label NAK for setting the target function TXNAK.

[4] If the variable field TXSTAMP contains the value “ACKXMIT ”, a positive transmission acknowledgment was received from the Headoffice Telex on a

fault-tolerant system for an outgoing telex message. Processing is continued at the label ACK for setting the target function TXACK.

[5] If the variable field TXSTAMP contains the value “NAKXMIT ”, a negative transmission acknowledgment was received from the Headoffice Telex on a fault-tolerant system for an outgoing telex message. Processing is continued at the label NAK for setting the target function TXNAK.

[6] If the variable field TXSTAMP contains the value “TELEXRCV”, an incoming telex message was received from the Headoffice Telex on a fault-tolerant system. Processing is continued at the label RCV for setting the two target functions TXRCV and TXSTPLR.

[7] If the variable field TXSTAMP contains the value “DUPLRQNF”, an incoming telex message was received from the Headoffice Telex on a fault-tolerant system, and the last received queue TXSTPLR was empty. Therefore, the Telex Link cannot determine whether the message is a duplicate. Processing is continued at the label PDR for setting the two target functions TXPDR and TXSTPLR.

[8] The variable field TXSTAMP contains the value “ENLOUT ” when a message is stored in the TXHCFSND queue. If this stamp is found during routing, something is wrong and processing is continued at the label CLEAN for setting the target function TXCLEAN, which can be a dummy queue if you want to get rid of these messages.

[9] The variable field TXSTAMP contains the value “ENLIN ”, when a message is stored in the TXHCFRCV queue. If this stamp is found during routing, something is wrong and processing is continued at the label CLEAN for setting the target function TXCLEAN, which can be a dummy queue if you want to get rid of these messages.

[10] If the variable field TXSTAMP contains the value “CLEANUP ”, a message was found in the TXHCFSND or TXHCFRCV queue during the startup of the Telex Link. Processing is continued at the label CLEAN for setting the target function TXCLEAN, which can be a dummy queue if you want to get rid of these messages.

[11] If the variable field TXSTAMP contains the value “INVALID ”, a message was received from the Headoffice Telex on a fault-tolerant system that cannot be identified as a telex message. Processing is continued at the label INVR for setting the target function TXINVR.

[12] If the variable field TXSTAMP contains the value “UNSXMACK”, a logical or transmission acknowledgment was received from the Headoffice Telex on a fault-tolerant system and the outgoing telex message was not found in the TXSTPPDE, TXWAIT, or TXNAK queue. Processing is continued at the label INVR for setting the target function TXINVR.

[13] With this and the following statements, the targets are set to the functions TXERROR (also for incorrect contents of the stamp field), TXWAIT, TXACK and TXPR0, TXNAK, TXRCV and TXSTPLR, TXPDR and TXSTPLR, TXCLEAN, and TXINVR as determined by the conditions tested before, and processing is continued at the label END (GOTO=END).

[14] A DSLROUTE TYPE=FINAL statement with the error target function TXERROR is the last statement in this routing table.

Examples of Routing Tables for the MERVA Link

The MERVA Link requires a MERVA ESA routing table to provide the Message Integrity Protocol (MIP) service element. It also supports MERVA ESA routing tables to connect user application queues to the MERVA Link processing queues. With some restrictions, the MERVA Link internal routing of messages can be controlled via MERVA ESA routing tables.

MERVA Link Message Routing

The message flow in the MERVA Link can be summarized as follows: an outgoing application message is routed to a MERVA ESA queue, which is a member of a MERVA Link Send Queue Cluster. This queue is called a MERVA Link send queue.

If the ASP that owns this send queue is not closed, the MERVA Link moves this message to the MERVA Link application control queue, sends the message to the partner and requests a transfer confirmation. When this confirmation has been received the message is updated with transfer control information and routed to the next applicable queue(s). This action is named “routing of a confirmed message”.

Dependent on an agreement between cooperating applications, the partner application, which has received the message, returns an acknowledgment message when it has actually processed that message (not only obtained this message, which was confirmed earlier). The MERVA Link tries to correlate an acknowledgment message with the reported message, and adds receipt control information to the reported message. Independent of the success of this correlation, the acknowledged message or the acknowledgment message itself is routed to the applicable next queue(s). This action is named “routing of an incoming acknowledgment message”.

A message sent by the partner application is formatted to the MERVA ESA format and routed to the applicable incoming application message queue(s). This action is named “routing of an incoming application message”.

A closed ASP must not transfer any message to its partner ASP. It routes messages from a MERVA Link send queue to a send queue of another transmission media (for example, telex). With the help of the MERVA System Control Facility command **recover**, in-process messages of a closed ASP can be copied with a PDM indicator to a send queue of another transmission media. This copy process is also controlled by a MERVA ESA routing table. This action is named “routing messages of a closed ASP”.

An undeliverable message causes a sending ASP to become inoperable. An inoperable ASP cannot transfer messages to its partner ASP. With the help of the MERVA System Control Facility command **iprecov**, an inoperable ASP can be recovered from an undeliverable message by removing that message from the set of IP messages in the application control queue. This move process is also controlled by a MERVA ESA routing table. This action is named “routing of a recovered undeliverable message”.

Routing Confirmed Messages: A confirmed message has a message class of “CF” or “CA” in the MERVA Link control field EKACLASS. A message of class “CF” can be an application message or an acknowledgment message. The contents of the MERVA Link control field EKARECRC shows the difference. EKARECRC (receipt return code) containing 00, 04, or 08 in a message of the class “CF” identifies an acknowledgment message. A message of class “CA” is an application message that

has already been acknowledged. The MERVA Link control field EKARECRC contains 00, 04, or 08 identifying an acknowledged message.

An application message must be routed to an ACK Wait Queue if an acknowledgment is expected from the partner and the MERVA Link must correlate this acknowledgment with this message. You must define the name of this ACK Wait Queue in the partner table to ensure that a confirmed application message is routed to this queue.

An acknowledgment message is not acknowledged by the receiving partner. Its transfer, however, is confirmed. Therefore, an acknowledgment message should not be routed to an ACK Wait Queue. It must be routed to any final log queue or it must be discarded (routed to a dummy queue).

Routing Incoming Acknowledgment Messages: An acknowledgment message or an acknowledged message have a message class of LR (last received) in the MERVA Link control field EKACLASS. The contents of the MERVA Link control field EKARECRC can be one of the following receipt return codes:

00 Final receipt

04 Not final receipt

08 Nonreceipt, which is a negative acknowledgment (NACK)

All messages of class LR must be routed to the MERVA Link application control queue to provide the Message Integrity Protocol service element. This service element cannot be provided without the support of the MERVA ESA routing table.

An acknowledgment message or an acknowledged message with receipt return code 00 or 08 (final ACK or final NACK) must be routed to an application queue that contains finished messages.

An acknowledgment message or an acknowledged message with receipt return code 04 (not-final ACK) must be routed to the ACK Wait Queue as another acknowledgment is expected for this message (either another not final or a final acknowledgment).

Routing Incoming Application Messages: An incoming application message has a message class of LR (last received) in the MERVA Link control field EKACLASS. The contents of the MERVA Link control field EKARECRC is not 00, 04, or 08.

Here all messages must be routed to the MERVA Link application control queue to provide the Message Integrity Protocol service element. This service element cannot be provided without this support from the MERVA ESA routing table.

An incoming application message must be routed to an application queue that contains messages received from the partner. This queue is the input queue for an application that processes the incoming message and generates an acknowledgment to be sent to the originator of that message.

Routing Messages of a Permanently Closed ASP: Issue the MERVA Link Control Facility **aclose** command to set an ASP into CLOSED status. A closed ASP does not transmit any messages to its partner ASP. It routes all messages in the MERVA Link send queue cluster to a send queue of another transmission media.

In addition to routing messages in a send queue of another transmission media, MERVA Link in-process messages in the application control queue can also be

routed (copied) to another transmission media. This copy process is initiated by a MERVA Link Control Facility **recover** command.

A message routed out of a send queue of a closed ASP has a message class of RS in the MERVA Link control field EKACLASS. This message can be an application message or an acknowledgment message. EKARECRC (receipt return code) containing 00, 04, or 08 identifies an acknowledgment message. An acknowledgment message can, however, be identified by application defined information.

An in-process message recovered from the application control queue of a closed ASP has a message class of RC in the MERVA Link control field EKACLASS. The MERVA Link control field EKAPDUPM contains the characters PDM to indicate that this message may be duplicate (an in-process message, for example, in the application control queue might have already been transmitted to the partner ASP when the **recover** command is issued). This message may be an application message or an acknowledgment message. EKARECRC (receipt return code) containing 00, 04, or 08 identifies an acknowledgment message. An acknowledgment message can, however, be identified by application defined information.

Routing Messages of a Temporarily Closed ASP: An ASP can be temporarily set in CLOSED status upon request of the user exit associated with this ASP. A temporarily closed ASP does not transmit the message currently in process to its partner ASP. It routes it to a send queue of another transmission media.

The class of a message routed out of a send queue of a temporarily closed ASP is specified by the user exit. This message may be an application message or an acknowledgment message. EKARECRC (receipt return code) containing 00, 04, or 08 identifies an acknowledgment message. An acknowledgment message can, however, be identified by application defined information.

Routing Recovered Undeliverable Messages: If you issue the MERVA System Control command **iprecov** to recover an ASP from an undeliverable message, or if you ask for an automatic IP message recovery when a message cannot be delivered to its recipient application, the undeliverable message is removed from the application control queue. A recovered undeliverable message has a message class of RI in the MERVA Link control field EKACLASS.

MERVA Link Sample Routing Tables

The MERVA ESA Routing Tables used in the MERVA Link sample are EKARTS and EKARTSCQ. These routing tables control both, the operation of the MERVA Link and the activities of a MERVA ESA terminal user who is authorized to view and modify the messages in all applicable queues, and to request message routing via the **eom** or **get next** command.

The MERVA Link sample contains a sequence of activities performed by an authorized terminal user. This terminal user represents an application using the MERVA Link in the MERVA Link sample.

The MERVA Link sample routing tables provide the routing definition required by the MERVA Link sample scenario described in the *MERVA for ESA Installation Guide*.

The MERVA ESA Routing Tables for the MERVA Link sample are coded as shown in Figure 18 and in Figure 19 on page 76. They are available in source code with explanations that are not shown in these figures.

```
EKARTS   DSLROUTE TYPE=DEFINE,FIELD=(QN,MSGTRFUN,,,,,VFIRST,LASTDA)
         DSLROUTE TYPE=DEFINE,FIELD=(RECRC,EKARECRC,,,,,VFIRST)
         DSLROUTE TYPE=DEFINE,FIELD=(SUB,EKAAMSUB,,,,,VFIRST),LENGTH=8
         DSLROUTE TYPE=DEFINE,FIELD=(OA,EKAOAPPL,,,,,VFIRST)
         DSLROUTE TYPE=DEFINE,FIELD=(ON,EKAONODE,,,,,VFIRST)
         DSLROUTE TYPE=DEFINE,FIELD=(RA,EKARAPPL,,,,,VFIRST)
         DSLROUTE TYPE=DEFINE,FIELD=(RN,EKARNODE,,,,,VFIRST)
*
         DSLROUTE TYPE=TEST,COND=(QN,'EKAEOM',EQ,SHORT),TRUE=TSTAP
         DSLROUTE TYPE=TEST,COND=(QN,'EKAEAM',EQ,SHORT),TRUE=TSTRR
         DSLROUTE TYPE=TEST,COND=(QN,'EKAAWQ',EQ,SHORT),TRUE=TSTAW
         DSLROUTE TYPE=TEST,COND=(QN,'EKACMQ',EQ,SHORT),TRUE=CPYCM
         DSLROUTE TYPE=TEST,COND=(QN,'EKARMQ',EQ,SHORT),TRUE=CPYRM
         DSLROUTE TYPE=TEST,COND=(QN,'EKAEMQ',EQ,SHORT),TRUE=CPYRM
         DSLROUTE TYPE=SET,TARGET='EKAEOM',GOTO=END
*
TSTAP    DSLROUTE TYPE=TEST,COND=(RN,'N1',EQ,SHORT),TRUE=SETBTB
         DSLROUTE TYPE=SET,TARGET='EKA1IS1',GOTO=END
*
TSTRR    DSLROUTE TYPE=TEST,COND=(RECRC,'00',EQ),TRUE=TSTRRON
         DSLROUTE TYPE=TEST,COND=(RECRC,'04',EQ),TRUE=SETEAMS
         DSLROUTE TYPE=TEST,COND=(RECRC,'08',EQ),TRUE=TSTRRON
         DSLROUTE TYPE=TEST,COND=(SUB,'$ACK$ 00',EQ),TRUE=TSTRRON
         DSLROUTE TYPE=TEST,COND=(SUB,'$ACK$ 04',EQ),TRUE=SETEAMS
         DSLROUTE TYPE=TEST,COND=(SUB,'$ACK$ 08',EQ),TRUE=TSTRRON
         DSLROUTE TYPE=SET,TARGET='EKAEAM',GOTO=END
*
SETEAMS  DSLROUTE TYPE=SET,TARGET='EKAEAM'
TSTRRON  DSLROUTE TYPE=TEST,COND=(ON,'N1',EQ,SHORT),TRUE=SETBTB
*
SETBTB   DSLROUTE TYPE=SET,TARGET='EKA1IS1',GOTO=END
*
TSTAW    DSLROUTE TYPE=TEST,COND=(RECRC,'00',EQ),TRUE=SETCM
         DSLROUTE TYPE=TEST,COND=(RECRC,'08',EQ),TRUE=SETCM
         DSLROUTE TYPE=SET,TARGET='EKAAWQ',GOTO=ENDCPYCM
         DSLROUTE TYPE=SET,TARGET='EKACMQ'
         DSLROUTE TYPE=SET,TARGET='EKAEOM',GOTO=END
*
CPYRM    DSLROUTE TYPE=SET,TARGET='EKARMQ'
         DSLROUTE TYPE=SET,TARGET='EKAEOM',GOTO=END
*
SETCM    DSLROUTE TYPE=SET,TARGET='EKACMQ',GOTO=END
ERROR    DSLROUTE TYPE=SET,TARGET='EKAEOM'
END      DSLROUTE TYPE=FINAL
         END
```

Figure 18. EKARTS Sample Unique Routing Table

```
EKARTSCQ DSLROUTE TYPE=DEFINE,FIELD=(CLASS,EKACLASS,,,,VFIRST)
         DSLROUTE TYPE=DEFINE,FIELD=(RECRC,EKARECRC,,,,VFIRST)
         DSLROUTE TYPE=DEFINE,FIELD=(DELRC,EKADELRC,,,,VFIRST)
         DSLROUTE TYPE=DEFINE,FIELD=(ACQNM,EKAACQNM,,,,VFIRST)
*
         DSLROUTE TYPE=TEST,COND=(CLASS,'IP',EQ),FALSE=TSTLC
         DSLROUTE TYPE=SET,TARGET=ACQNM,GOTO=END
TSTLC    DSLROUTE TYPE=TEST,COND=(CLASS,'LC',EQ),FALSE=TSTCF
         DSLROUTE TYPE=SET,TARGET=ACQNM,GOTO=END
*
TSTCF    DSLROUTE TYPE=TEST,COND=(CLASS,'CF',EQ),FALSE=TSTLR
         DSLROUTE TYPE=TEST,COND=(RECRC,'0',EQ,SHORT),TRUE=SETDQ
         DSLROUTE TYPE=SET,TARGET='EKAAWQ',GOTO=END
SETDQ    DSLROUTE TYPE=SET,TARGET='EKADMY',GOTO=END
*
TSTLR    DSLROUTE TYPE=TEST,COND=(CLASS,'LR',EQ),FALSE=TSTRC
         DSLROUTE TYPE=TEST,COND=(RECRC,'04',EQ),TRUE=SETLRA
         DSLROUTE TYPE=TEST,COND=(RECRC,'0',EQ,SHORT),TRUE=SETLRF
         DSLROUTE TYPE=TEST,COND=(DELRC,'00',EQ),TRUE=SETLRI
SETLRE   DSLROUTE TYPE=SET,TARGET='EKAEMQ'
         DSLROUTE TYPE=SET,TARGET='EKAEAM'
         DSLROUTE TYPE=SET,TARGET=ACQNM,GOTO=END
SETLRI   DSLROUTE TYPE=SET,TARGET='EKARMQ'
         DSLROUTE TYPE=SET,TARGET='EKAEAM'
         DSLROUTE TYPE=SET,TARGET=ACQNM,GOTO=END
SETLRA   DSLROUTE TYPE=SET,TARGET='EKAAWQ'
         DSLROUTE TYPE=SET,TARGET=ACQNM,GOTO=END
SETLRF   DSLROUTE TYPE=SET,TARGET='EKACMQ'
         DSLROUTE TYPE=SET,TARGET=ACQNM,GOTO=END
*
TSTRC    DSLROUTE TYPE=TEST,COND=(CLASS,'RC',EQ),FALSE=TSTRI
         DSLROUTE TYPE=SET,TARGET='EKAEOM',GOTO=END
TSTRI    DSLROUTE TYPE=TEST,COND=(CLASS,'RI',EQ),FALSE=TSTRR
         DSLROUTE TYPE=SET,TARGET='EKAEOM',GOTO=END
TSTRR    DSLROUTE TYPE=TEST,COND=(CLASS,'RR',EQ),FALSE=TSTRS
         DSLROUTE TYPE=SET,TARGET='EKAEOM',GOTO=END
TSTRS    DSLROUTE TYPE=TEST,COND=(CLASS,'RS',EQ),FALSE=OTHER
         DSLROUTE TYPE=SET,TARGET='EKAEOM',GOTO=END
OTHER    DSLROUTE TYPE=SET,TARGET='EKAEOM'
END      DSLROUTE TYPE=FINAL
         END
```

Figure 19. EKARTSCQ Sample ACQ Routing Table

Processing Routing Tables

The installation of a routing table requires:

- The routing table name defined in the function table entry with the ROUTE= parameter of the DSLFNT macro, or in the SWIFT Link logical terminal table with the ROUTIN, ROUTOUT and ROUTSK parameters of the DWSLT macro.
- The assembly of the new or changed routing tables (and MERVA ESA function table or the SWIFT Link logical terminal table or both if changed).
- Link-editing of the new or changed routing tables.

Note: MERVA ESA, the SWIFT Link, and MERVA Link load all routing tables.

Assigning the Program Function (PF) Keys

A Program Function Key table (DSLMPFxx) is used to assign commands to the program function (PF) keys for a screen. Several tables can be defined so that PF keys can be used differently by users or for different functions. The following explains how to define the PF Key tables.

The DSLMPFK Macro

The PF Key macro (DSLMPFK) is used to define a PF Key Table (refer to the *MERVA for ESA Macro Reference*). A PF key table is used by MERVA ESA during a user session. The table defines which command is to be executed when the user presses a PF key. The table name to be used can be defined in the function table entry and in the user profile for each user individually.

During a user session, another PF Key table can be selected for use by the **pfkeys** screen command. This PF Key table is used during the current user session, until a new one is selected with the **pfkeys** command. If a PF Key table is not defined in the user profile and no **pfkeys** command is used, a default PF Key table name as specified in the MERVA ESA Function table entry is used. If no MERVA ESA Function table entry is available, the PF Key table DSLMPF00 is used. The current PF key settings can be inspected by using the **show pfkeys** command.

The PF key definitions are related to an actual function by the PFKSET parameter of the DSLFNT macro or by information in the MERVA ESA user file. The PF key definitions are assigned in groups, and the PFGROUP parameter of the DSLFNT macro defines which group is to be used in the function. The GROUP parameter allows you to specify different PF key settings for different user functions in the same PF key table.

In each PF Key table up to 9 PF key information lines can be specified. These PF key information lines can be displayed on the reserved lines on the screen. They show the current PF key settings. To display the n-th PF key information line the field PFKLINE, DA=n must be specified in the screen device section of the Message Control Block, which is used to define the display format. This MCB is preferably a frame definition MCB, and should be specified in the FRAME= parameter of a DSLFNT macro.

The following describes the three types of DSLMPFK macro:

- DSLMPFK TYPE=INITIAL

  The INITIAL macro defines the PF key table header.
- DSLMPFK TYPE=ENTRY

  The ENTRY macro defines a PF key entry, or a PF key information line.
- DSLMPFK TYPE=FINAL

  The FINAL macro closes the definition of a PF key table.

Coding Considerations

The following is a list of items to be considered when coding the DSLMPFK macro:

- The MERVA ESA screen editing uses the PF Key table to generate the command connected to a PF key. The command is written into the TOF field 'DSLCMDL'.
- The system field separation routine extracts the PF key information lines from the current PF Key table. Up to 9 PF key information lines can be specified in a table. The field name for this information is DSLPFKL, and the data area index

is used to select a specific information line. The DSLMMFS interface provides an access to the current PF Key table. The code type for this request is TYPE=GETPFK. The address of the requested PF Key table is stored in a field of the Message Format Service permanent storage. PF Key tables can also be specified in the Message Format Service program table DSLMPTT.

Overview of Available PF Keys for All MERVA ESA Functions

Table 1 gives an overview of the PF keys available with the PF Key table DSLMPF00 as supplied with MERVA ESA. It is recommended that you **not** change the PF key groups defined for MERVA ESA function programs.

Table 1. Program Function Key Settings (Part 1 of 4)

	Default	**Function Selection**	**Command**	**User File Maintenance**			**Message Selection**
				Record (Update)	**Record (Display Only)**	**List**	
PFK Group	00 00	FUN 04	CMD+1 09	USR+1 13	USR+2 14	USR+3 15	MSG+0 16
PF KEY							
PF01	Help	Help	Help	Help	Help	Help	Help
PF02	Retrieve	Retrieve	Repeat	Retrieve	Retrieve	Retrieve	Retrieve
PF03	Return	Signoff	Return	Return	Return	Return	Return
PF04			DF	Display	Display	Display	
PF05			DU	List	List	List	Get Next
PF06			DM Last	ListFirst	ListFirst	ListFirst	Get First
PF07		Page -1		Page -1	Page -1		
PF08		Page +1		Page +1	Page +1		
PF09	Hardcopy	Hardcopy	Hardcopy	Hardcopy	Hardcopy	Hardcopy	Hardcopy
PF10			DP	Delete		Delete	
PF11			DQ Filled	Replace			List
PF12			DLA	Add			
PF13	Help	Help	Help	Help	Help	Help	Help
PF14	Retrieve	Retrieve	Repeat	Retrieve	Retrieve	Retrieve	Retrieve
PF15	Return	Signoff	Return	Return	Return	Return	Return
PF16			DF	Display	Display	Display	
PF17			DU	List	List	List	Get Next
PF18			DM Last	ListFirst	ListFirst	ListFirst	Get First
PF19		Page -1		Page -1	Page -1		
PF20		Page +1		Page +1	Page +1		
PF21	Hardcopy	Hardcopy	Hardcopy	Hardcopy	Hardcopy	Hardcopy	Hardcopy
PF22			DP	Delete		Delete	
PF23			DQ Filled	Replace			List
PF24			DLA	Add			
Enter							

Table 2. Program Function Key Settings (Part 2 of 4)

	Message Processing		**Message Processing NOPROMPT**	**Message Selection List Panel**
	Prompt	**Telex**		
PFK Group	MSG+1 17	MSG+1 17	MSG+2 18	MSG+3 19
PF Key				
PF01	Help	Help	Help	Help
PF02	Retrieve	Retrieve	Retrieve	Retrieve
PF03	EOM	EOM	EOM	Return
PF04	Repeat	Txinsert 1	Repeat	Get QSN
PF05	Get Next	Get Next	Get Next	Get Next
PF06	Requeue	Requeue	Requeue	Get First
PF07	Page -1	Page -1	Page -1	List Back
PF08	Page +1	Page +1	Page +1	List Last
PF09	Hardcopy	Hardcopy	Hardcopy	Hardcopy
PF10	Prompt Line	Txsplit	Prompt Line	
PF11	Prompt	Txjoin	Prompt	List First
PF12	Escape	Escape	Escape	List Off
PF13	Help	Help	Help	Help
PF14	Retrieve	Retrieve	Retrieve	Retrieve
PF15	EOM	EOM	EOM	Return
PF16	Repeat	Txinsert 1	Repeat	Get QSN
PF17	Get Next	Get Next	Get Next	Get Next
PF18	Requeue	Requeue	Requeue	Get First
PF19	Page -1	Page -1	Page -1	List Back
PF20	Page +1	Page +1	Page +1	List Last
PF21	Hardcopy	Hardcopy	Hardcopy	Hardcopy
PF22	Prompt Line	Prompt Line	Prompt Line	
PF23	Prompt	Prompt	Prompt	List First
PF24	Escape	Escape	Escape	List Off
Enter				

Table 3. Program Function Key Settings (Part 3 of 4)

	General File Maintenance				Authenticator Key File Maintenance		
	File Selection	**Record (Update)**	**Record (Display Only)**	**List**	**Record (Update)**	**List**	**Record (Display Only)**
PFK Group	FLM 20	FLM+1 21	FLM+2 22	FLM+3 23	AUT+1 25	AUT+3 27	AUT+5 29
PK Key							
PF01	Help	Help	Help	Help	Help	Help	Help
PF02	Retrieve	Retrieve	Retrieve	Retrieve	Retrieve	Retrieve	Retrieve
PF03	Return	Escape	Escape	Escape	Return	Return	Return
PF04		Display	Display	Display	Display	Display	Display
PF05		List	List	List	List	List	List
PF06		ListFirst	ListFirst	ListFirst	ListFirst	ListFirst	ListFirst
PF07		Page -1	Page -1			Page -1	
PF08		Page +1	Page +1			Page +1	
PF09	Hardcopy	Hardcopy	Hardcopy	Hardcopy	Exchange		
PF10		Delete		Delete	Delete	Delete	
PF11		Replace			Replace		
PF12		Add			Add		
PF13	Help	Help	Help	Help	Help	Help	Help
PF14	Retrieve	Retrieve	Retrieve	Retrieve	Retrieve	Retrieve	Retrieve
PF15	Return	Escape	Escape	Escape	Return	Return	Return
PF16		Display	Display	Display	Display	Display	Display
PF17		List	List	List	List	List	List
PF18		ListFirst	ListFirst	ListFirst	ListFirst	ListFirst	ListFirst
PF19		Page -1	Page -1			Page -1	
PF20		Page +1	Page +1			Page +1	
PF21	Hardcopy	Hardcopy	Hardcopy	Hardcopy	Exchange		
PF22		Delete		Delete	Delete	Delete	
PF23		Replace			Replace		
PF24		Add			Add		
Enter							

Table 4. Program Function Key Settings (Part 4 of 4)

	Authenticator Key File Maintenance					MERVA Link Control Facility	MERVA System Control
	List (Display Only)	**Record (Update)**	**List (Update)**	**Record (Authorize)**	**List (Authorize)**		
PFK Group	AUT+7 31	AUT+9 33	AUT+11 35	AUT+13 37	AUT+15 39	41	42
PF Key							
PF01	Help	Help	Help	Help	Help	Help	Help
PF02	Retrieve	Retrieve	Retrieve	Retrieve	Retrieve	Retrieve	Repeat
PF03	Return	Return	Return	Return	Return	Return	Return
PF04	Display	Display	Display	Display	Display	SELECT	DF
PF05	List	List	List	List	List	START	DU
PF06	ListFirst	ListFirst	ListFirst	ListFirst	ListFirst	Nextgrp	DM Last
PF07	Page -1		Page -1		Page -1	Backward	Page -1
PF08	Page +1		Page +1		Page +1	Forward	Page +1
PF09				Exchange		SWAP	SWAP
PF10		Delete	Delete			Kickoff	DP
PF11		Replace		Reject		Listinop	DQ Filled
PF12		Add		OK		Lstall	DL
PF13	Help	Help	Help	Help	Help	Help	Help
PF14	Retrieve	Retrieve	Retrieve	Retrieve	Retrieve	Retrieve	Repeat
PF15	Return	Return	Return	Return	Return	Return	Return
PF16	Display	Display	Display	Display	Display	SELECT	DF
PF17	List	List	List	List	List	START	DU
PF18	ListFirst	ListFirst	ListFirst	ListFirst	ListFirst	Nextgrp	DM Last
PF19	Page -1		Page -1		Page -1	Backward	Page -1
PF20	Page +1		Page +1		Page +1	Forward	Page +1
PF21				Exchange		SWAP	SWAP
PF22		Delete	Delete			Kickoff	DP
PF23		Replace		Reject		Listinop	DQ Filled
PF24		Add		OK		Lstall	DL
Enter							

Figure 20 shows the beginning of the PF Key table DSLMPF00, which sets the PF keys as shown in Table 1 on page 78.

```
PF00     TITLE   'PROGRAM FUNCTION KEY DEFINITIONS'
*-------------------------------------------------------------*
*        D E F A U L T  SETTINGS FOR
*        PROGRAM FUNCTION  KEYS
*-------------------------------------------------------------*
DSLMPF00 DSLMPFK TYPE=INITIAL
         PRINT   NOGEN
         DSLMPFK AID=ENTER,CS=YES,GROUP=00                        [1]
         DSLMPFK AID=PF01,CMD='Help'                              [2]
         DSLMPFK AID=PF02,CMD='Retrieve'
         DSLMPFK AID=PF03,CMD='Return'
         DSLMPFK AID=PF09,CMD='Hardcopy'
         DSLMPFK AID=PF13,CMD='Help'
         DSLMPFK AID=PF14,CMD='Retrieve'
         DSLMPFK AID=PF15,CMD='Return'
         DSLMPFK AID=PF21,CMD='Hardcopy'
         DSLMPFK PFKLINE='1=Help      2=Retrieve  3=Return     4=   * [3]
                   5=          6=             '
         DSLMPFK PFKLINE='7=          8=          9=Hardcopy  10=   *
                  11=         12=             '
*-------------------------------------------------------------*
*        PROGRAM FUNCTION  KEY DEFINITIONS
*        FOR FUNCTION SELECTION
*-------------------------------------------------------------*
         DSLMPFK AID=ENTER,CS=YES,GROUP=FUN                       [4]
         DSLMPFK AID=PF03,CMD='Signoff'
         DSLMPFK AID=PF07,CMD='Page -1'
         DSLMPFK AID=PF08,CMD='Page +1'
         DSLMPFK AID=PF09,CMD=                                    [5]
         DSLMPFK AID=PF15,CMD='Signoff'
         DSLMPFK AID=PF19,CMD='Page -1'
         DSLMPFK AID=PF20,CMD='Page +1'
         DSLMPFK AID=PF21,CMD=
         DSLMPFK PFKLINE='1=Help      2=Retrieve  3=Signoff    4=   *
                   5=          6=             '
         DSLMPFK PFKLINE='7=Page -1   8=Page +1   9=          10=   *
                  11=         12=             '
         ...
         ...
         ...
         DSLMPFK TYPE=FINAL
         END
```

Figure 20. Example of a PF Key Table (DSLMPF00)

Notes:

[1] Within a PF Key table, default values valid within the table are defined in the group GROUP=00. In Table 1 on page 78, all group numbers for the PF Key groups are indicated.

[2] Since the **help** command is assigned to PF01 in group 00, and all the other groups have no definition of PF01, the **help** command is available with PF01 in all the other groups. The same applies to the **hardcopy** command assigned to PF9.

[3] The key descriptions for the PF fields defined here are displayed in the bottom frame on the screen, according to the function displayed.

[4] The group number for function selection is indicated by the assembler equate FUN.

The following assembler equates are generated by the DSLMPFK TYPE=INITIAL macro:

```
FUN      EQU   4         Function selection
CMD      EQU   8         Operator command processing
USR      EQU   12        User file maintenance
MSG      EQU   16        Message processing
FLM      EQU   20        General File maintenance
AUT      EQU   24        The SWIFT Link Authenticator-Key File maintenance
SHW      EQU   248       Display with the show command
HLP      EQU   252       Display with the help command
```

Table 1 on page 78 shows where additional numbers are also used in the pertinent group in the PF Key group line. The group numbers 101 to 247 are for use by the customer.

[5] With this macro the default setting for PF9 is removed.

Refer to the *MERVA for ESA Macro Reference* for further information about PF keys and the DSLMPFK macro.

Processing PF Key Tables

User commands can be assigned to different PF keys in one of the following ways:

- By changing DSLMPF00
- By creating new PF Key tables, and specifying them in the MERVA ESA Function table entries, or in the User File records. New PF Key tables may be specified in the MFS program table DSLMPTT.

In both cases, follow these steps:

1. Change the existing PF Key Table or code a new one.
2. Assemble the changed or new PF Key Table.
3. Link-edit the appropriate module for the changed or new PF Key Table:
   - If the PF Key Table is specified in the MERVA ESA Message Format Service program table (DSLMPTT), with the parameter LINK=YES, DSLMMFS must be link-edited.
   - If the PF Key Table is specified in DSLMPTT with the parameter LINK=NO or is not defined in DSLMPTT, the PF Key Table must be link-edited. It is then dynamically loaded.

Customizing Error and Diagnostic Messages for Operators and Users

For MERVA ESA and all MERVA ESA controlled network links (for example, the SWIFT Link), the Message Table DSLMSGT contains error and diagnostic messages for operators and users. All messages in DSLMSGT are generated through the macro DSLMSG, described in the *MERVA for ESA Macro Reference*.

DSLMSGT is generated through the DSLGEN macro during MERVA ESA installation. It contains an assembler COPY statement for each network link controlled by MERVA ESA; all messages are combined in one table. The copy code of each network link contains the message definitions with the DSLMSG macro.

DSLMSGT is assembled and link-edited during the installation process and loaded by the MERVA ESA programs that use it. The address of DSLMSGT is provided in the field COMMSGTA of DSLCOM.

A message can be retrieved from the message table by the retrieval program DSLOMSG, which gets the address of the message table as an input parameter,

therefore allowing for different message tables. The DSLMSG macro allows for defining the same message in more than one language in the same message table with the LAN parameter.

Application programs like DSLEUD, DSLHCP, and DSLSDY, select the language according to the definition in the user file record or **form** command (DSLEUD), or the definition in the function table entry that they process (PRFORM parameter of the DSLFNT macro, programs DSLHCP and DSLSDY). These messages usually have a 4-digit number in the message identification (DSLxxxx or DWSxxxx).

Other programs than DSLEUD, DSLHCP, and DSLSDY can support only one language per MERVA ESA installation. These messages usually have a 3-digit number in the message identification (DSLxxxI or DSLxxxA or DWSxxxI or DWSxxxA).

DSLMSGT is composed of copy codes for each network link. Some of these copy codes contain additional copy codes:

- DSLMSGTC for the Base Functions messages:
  - MERVA ESA operator messages DSLxxxI or DSLxxxA (copy code DSLOMSC)
  - MERVA ESA user messages DSL1xxx (copy code DSLEMSC)
  - MERVA ESA Message Format Service messages DSL3xxx (copy code DSLMMSC)
- DWSMSGTC for the SWIFT Link messages:
  - SWIFT Link operator messages DWSxxxI or DWSxxxA (copy code DWSOMSC)
  - SWIFT Link user messages DWS1xxx (copy code DWSEMSC)
  - SWIFT Link checking and separation messages DWS3xxx and DWS4xxx (copy code DWSMMSC)
- ENLMSGTC for the Telex Link messages:
  - Telex Link operator messages ENL9xxI
  - Telex Link user messages ENL35xx
- EKAMSGTC for the MERVA Link operator messages EKA7xxI, EKA7xxE, and EKA7xxW
- EKAMSGSC for the FMT/ESA with MERVA Link operator messages EKA8xxE

These operator or user messages can be changed if:

1. Another language is used as the only language
2. Another language is used as an additional language (multiple language support)
3. You add new messages

Translation of Messages into Another Language

If only one language is available for an operator or user message, and the default language ID 'E' is used (even if the language is not English), then no special provisions are required in user file records or function tables (PRFORM parameter), as the changed language is accessed by default. You can also translate only a part of the messages, that is, those parts presented to users who do need not to have any knowledge of English for their work.

For the translation, the appropriate copy code (see above) must be changed, and the appropriate message is translated considering the variable data (indicated by the variable numbers @n in the message text). The variable data can take any place

in the message text; therefore it need not have a fixed position, or, if there is more than one variable, they need not be in a specific order.

Note: The headers of the command responses for the MERVA ESA display commands (for example, **df**, **dm**, and **dp**) must never be changed, as the executing programs do not use variables but a fix list format. Comment statements in the copy codes indicate which messages are responses to commands.

Multiple Language Support

In countries such as Belgium, Canada, and Switzerland, where several languages are used by the users, the user message can be defined in several languages in the same message table. This can be done for the messages contained in the copy codes DSLEMSC, DSLMMSC, DWSEMSC, DWSMMSC, ENLMSGTC, EKAMSGTC, and EKAMSGSC.

Note: MERVA Link supports only English messages.

The messages are translated into the other languages and can be added to the copy code with the appropriate LAN parameter, for example, LAN=F for French or LAN=G for German. It is also possible to add other languages and to remove the English message texts.

It is possible to have only a part of the messages available in more than one language, for example, messages presented to users with no knowledge of English. In this case, the messages not available in a particular language are replaced by the first message found in DSLMSGT with the message identifier DSLxxxx, DWSxxxx, ENLxxxx, or EKAxxxx.

It is possible to mix the different languages and to have two or more versions of the same error message one after the other, or to collect all French or German messages in a group before or after other language messages.

Figure 21 shows an example of a message in two languages:

```
DSL1011  DSLMSG 'Password missing or incorrect'                    [1]
DSL1011  DSLMSG 'Passwort fehlt oder ist falsch',LAN=G             [2]
```

Figure 21. Example of Two Languages for the Same Message

Notes:

[1] This message is in English, as the LAN parameter is omitted and defaulted to LAN=E.

[2] This is the same message in German, and the parameter LAN=G is specified. The message identification defined in the label field is the same as for the English version of the message, therefore identifying both messages as having the same purpose.

The same rules apply for message variables as for translating messages into one other language.

Adding User-Defined Messages

User-defined messages are coded following the same rules as the messages provided by MERVA ESA using one or more languages. The following rules also apply:

1. Define a unique message identification for the new messages, preferably with a special acronym in the first three positions, for example, USRxxxx.
2. Supply your messages preferably in a separate copy book.
3. When the message is issued because of an MFS reason code of a user-written checking, default setting, editing, expansion or separation exit routine, then the 17 variables available with MFS error messages must be used. These variables are described in the copy code DSLMMSC.
4. When the message is issued by a user-written program that calls DSLOMSG for retrieval of the error message, then the variables to be used must be defined by your program as a list of substitution items. For details see the description of the DSLOMS macro in the *MERVA for ESA Macro Reference*.

Processing the Changed Message Table

After changing one of the copy codes used by DSLMSGT, DSLMSGT must be assembled and link-edited.

Chapter 2. The MERVA ESA Environment

This chapter explains how you customize the following areas in MERVA ESA:

- The parameters used in the module DSLPRM. This section shows how to use the DSLPARM macro.
- Use of a security manager.
- The parameters used in the nucleus server table, DSLNSVT, to allow parallel nucleus server processing and the definitions for intertask and interservice communication.
- The definition of the transaction table DSLTXTT.
- The files that use the MERVA ESA general file services. This section shows how to code the general file table (DSLFLTT).
- The features of the terminal screens and printers by means of the terminal feature definition table (DSLTFDT).

Defining Basic MERVA ESA Parameters in Module DSLPRM

For the basic customizing of the MERVA ESA environment the module DSLPRM is used. This module uses the DSLPARM macro, described in the *MERVA for ESA Macro Reference*.

The DSLPARM macro can be used by user-written programs to map the customizing parameters.

DSLPRM Module Sample

An example of the MERVA ESA customizing parameter module is shown below.

```
DSLPARM SVC=253,                                        * [1]
      NQE=10000,                                        * [2]
      CVTEXTO=36,                                       * [3]
      USER=10,                                          * [4]
      USERSTO=(200,200),                                * [5]
      MCBNUM=4,                                         * [6]
      QDS=(2),                                          * [7]
      MFSSTOR=(8192,32000,1536),                        * [8]
      TOFSIZE=(2048,4096),                              * [9]
      NICBUF=12000,                                     *
      MAXBUF=1048576,                                   *
      LRGMSG=YES,                                       *
      JRNBUF=(,YYYY),                                   * [10]
      JSWITCH=MANUAL,                                   * [11]
      DSLID=IFT2,                                       * [12]
      CID=DSL,                                          * [13]
      EXQUE=YES,                                        * [14]
      EXUSR=YES,                                        * [15]
      EXJRN=YES,                                        * [16]
      CWAOFF=20,                                        * [17]
      ITC=(TSQ,APPC),                                   * [18]
      ITCAREQ=(,FDMAMF5),                               * [19]
      ITCASRV=(,FDMAMF5,MERVAITC),                      * [20]
      ITCQSRV=DSLNX,                                    * [21]
      WSASRV=(,FDMAMF5,MERVAFCS)                          [22]
END
```

Figure 22. DSLPRM Sample 1

Notes:

[1] For MVS only. Specifies 253 as SVC number for the installation of the interregion communication program DSLNICP. This parameter is mandatory in MERVA ESA IMS. In MERVA ESA CICS, this parameter is only required if the MERVA ESA batch programs DSLSDI, DSLSDO, or DSLSDY are used.

[2] A maximum number of 10,000 queue elements is specified for storing in the queue data set.

[3] For MVS only. CVTEXTO=36 specifies that in the extension table used for storing the address of the MERVA ESA interregion communication area (DSLICA) the offset 36 is used. The address of the extension table is stored in the CVTUSER field of the MVS common vector table (CVT).

This parameter has a default of zero and should be used to avoid two MERVA ESA installations competing over the first fullword in the extension table.

[4] USER=10 specifies that a maximum number of 10 users can be active.

Under IMS, this parameter is also used to format the MERVA ESA end-user SPA file. If this parameter is changed later, the SPA file must be reallocated and formatted using the DSLEBSPA program.

[5] USERSTO=(200,200) specifies with the first parameter that 200 bytes of permanent storage (space in SPA) are reserved for user exits that can be used for communication purposes with DSLEUD and external function programs, or with the external function programs alone. The second parameter specifies 200 bytes as working storage for the user exits.

[6] MCBNUM=4 specifies that (4+2) MCBs are loaded concurrently for each session. The default (10) was reduced to save storage.

[7] QDS=(2) specifies that duplicate queue data sets are available. Processing with the second data set is not continued after an I/O error.

Note: Both data sets must be made equal after abnormal end of processing before MERVA ESA can be restarted. For more information refer to the descriptions of messages DSL360I and DSL361I in *MERVA for ESA Messages and Codes*.

[8] MFSSTOR=(8192,32000,1536) specifies storage sizes for MFS buffers. The MFS permanent storage is increased to 8192 bytes (default=6144) to allow more nesting levels to be processed. The MFS temporary storage and the MFS storage used for retype verification are not changed.

[9] With the following specifications, the sizes of the buffers are increased to allow the processing of large messages. The sizes of the buffers are increased dynamically by MERVA ESA programs until the value specified in the MAXBUF parameter is reached, in this example 1MB (1048576 bytes). The initial TOFSIZE is 2KB; the TOF grows dynamically by 4KB when more space is needed, until the limit of 1MB is reached. When LRGMSG=YES is specified a large message cluster must be allocated and assigned to the MERVA ESA region; the large messages created in MERVA ESA can be stored in the MERVA ESA large message cluster. The parameter of JRNBUF indicates that journaling should segment buffers that are too large to fit into one journal data set record.

[10] JRNBUF=YYYY specifies a 4-digit year for the journal record header. This specification implies the use of segmented journal records for buffers too large to fit into one journal data set record. Use of the 4-digit year specification is mandatory.

[11] The JSWITCH parameter defines the initial switching mode of the journal data sets. The specification MANUAL causes MERVA ESA to switch between journal data sets only when the operator requests it. This specification is useful when a journal is archived by a separate batch job. Journal output can be switched to the alternate data set for the duration of the batch job.

[12] DSLID=IFT2 specifies an identifier of 4 characters that is added to all unsolicited operator messages issued at the operating system console. If two or more MERVA ESA installations are running in the same MVS or VSE system, this identifier determines which MERVA ESA installation issues which messages. In VSE, the first three characters of DSLID are used for the MERVA ESA interpartition communication to identify for the MERVA ESA batch programs DSLSDI, DSLSDO, and DSLSDY with which MERVA ESA installation they want to communicate. It is necessary to specify different DSLID parameters in each MERVA ESA.

[13] CICS only. This parameter specifies a 3-character operator identifier. MERVA ESA can be started from a CICS terminal only by an operator whose identifier starts with these 3 characters. The operator identifier is defined in the CICS signon table (DFHSNT). For CICS/ESA V4, the operator identifier must be specified in RACF® or an alternative security system used with CICS/ESA. In VSE, this operator must sign on at the VSE system console before entering MERVA ESA operator commands.

[14] Specifies whether the MERVA ESA queue test commands **move**, **copy**, **delete**, **delx**, and **free** can be used by the MERVA ESA operators. Because these commands bypass the routing process between MERVA ESA functions, they should only be used in a test installation. In a production

environment, EXQUE=NO must be specified. If EXQUE=YES or EXQUE=MASTER is specified, it is the responsibility of the installation to prevent misuse of these commands, for example, by specifying the commands individually in the “Unauthorized Commands” section of the MERVA ESA user-file record.

[15] Use this parameter to prevent a user of a bank sharing one MERVA ESA and SWIFT Link installation with several other banks, from creating a user-file record for accesses to functions of another bank during online maintenance of the MERVA ESA User File.

Note: The restriction of EXUSR=YES does not apply to master users.

[16] Specifies whether the MERVA ESA test command **jrn** (display journal) can be used by MERVA ESA operators. In a production environment, EXJRN=NO must be specified. If EXJRN=YES is specified, MERVA ESA operators can inspect journal records online. These journal records can contain sensitive information.

[17] CICS only. Use this parameter to specify which storage MERVA ESA uses in the CICS Common System Work Area (DFHCWA).

[18] This parameter specifies which method for MERVA ESA intertask communication should be used. The first subparameter applies to CICS transactions. The value TSQ specifies that CICS temporary storage queues should be used instead of the direct buffer move operation. The second subparameter applies to MVS programs not running under CICS. The value APPC specifies that APPC/MVS should be used instead of the direct buffer move operation through the MERVA ESA SVC.

[19] ITCAREQ specifies the APPC/MVS values used by the requester’s side of the intertask communication. The second parameter specifies the NOSCHED LU name which is used by the server’s side of the intertask communication. The TP name is taken from the server parameters ITCASRV.

[20] ITCASRV specifies the APPC/MVS values used by the server’s side of the intertask communication. The second subparameter specifies the NOSCHED LU name, the third subparameter specifies the TP name under which this server registers itself to APPC/MVS.

[21] ITCQSRV specifies the CICS TS queue values used by the server’s side of the intertask communication. This parameter specifies a prefix for the temporary storage queues used by MERVA ESA intertask communication. The remaining 3 characters will be filled with numeric values created dynamically.

[22] If you use the APPC/MVS version of the MERVA Message Processing Client Server, you must specify the WSASRV parameter. The Server registers with APPC to handle all requests from clients for conversation allocation with the specified Logical Unit and Transaction Program.

You can redistribute the permanent storage available for MERVA ESA end-user programs by changing permanent storage requests for several MERVA ESA buffers. The following MERVA ESA parameters request permanent storage (adjust their values to meet the requirements of your system):

- MCBNUM, storage required is (MCBNUM+2)*24, that is, 144 bytes in this sample parameter module.

- MFSSTOR, storage required is the sum of the first and third parameter, that is, 9728 bytes in this sample parameter module.
- USERSTO, storage required is given with the first parameter, that is, 200 bytes in this sample parameter module.

DSLPRM Settings for Large Messages

When receiving large SWIFT messages, especially those with multiple repeatable sequences, MERVA ESA needs space to format the messages internally—sometimes more than twice the size of the message. To handle such messages, use the following parameter settings in the parameter module DSLPRM:

APISMSG=12288 Allows large SWIFT messages of up to 10000 bytes. All your applications using the MSGSWIFT buffer must be recompiled when this value is increased! It is recommended for all new application programs to use the API calls MSGG and MSGP to format messages. For these API calls, the APISMSG parameter is *not* needed.

JRNBUF=(16000,YYYY) The record size in the journal data set must have a minimum of 16000. Set the record length of the corresponding VSAM clusters to the same value as is specified for this parameter.

LRGMSG=YES Indicates that large messages are supported.

MAXBUF=250000 This would allow for the worst case of repeatable sequences. Normally a value of **64000** should be sufficient. Any larger value of up to 2097152 (2MB) can be specified.

TOFSIZE=(18432,8192) Specifies that the TOF buffer can be increased dynamically up to the size of MAXBUF.

Defining the Parameters for Using a Security Manager

You can use a security manager to control the signon to MERVA ESA and to the user file maintenance function. This can be an external security manager (ESM), such as RACF under MVS, or the basic security manager (BSM) of VSE. Note that, for VSE, VSE/ESA Version 2.4 or later is required to use the BSM or any supported ESM.

When using a security manager, specify the following parameters in the module DSLPRM using macro DSLPARM:

- EUDTRAN=DSLE

  This is the default transaction code of the MERVA ESA end user driver DSLEUD. If DSLE is not defined in your installation, specify another valid transaction code for DSLEUD.
- EXDSP=YES

  This is only required if you want to start the end user driver by entering the transaction code DSLP from the CICS or IMS terminal. DSLP starts the program DSLPTMRV which itself starts the end user driver. Also refer to “Transparent Usage of the DSLEUD” on page 93.
- EXSEC=YES
- EXUID=YES
- MERVUSR=DSLUSER

This is the default pseudo user ID used by MERVA ESA when the user file is empty. If DSLUSER is not defined in your installation as a user ID known to the security manager for a CICS or IMS signon, specify another valid user ID.

- PGCALL=YES
- USERSTO=(8,208)

  The first subparameter shows the required SPA size reserved for user programs. If you have already defined a SPA size reserved for user programs in your installation, increase it by 8. In this case, you have to adjust the layout of the data in the SPA, as far as the DSLEUD user exits DSLEU001 and DSLEU003 are concerned.

  The second subparameter shows the default value for the temporary storage for DSLEUD user exits. It is not used in the context discussed. If required, you can specify your own value.
- USFPW=NO
- RACFSVC=nnn

  For MVS, this parameter is optional if MERVA ESA is running under CICS/ESA V4.1 or later. The parameter is required for the previous versions of CICS and for MERVA ESA running under IMS.

  nnn specifies a type-3 SVC number for the module DSLEUSVC. A decimal value from 200 to 255 can be specified. The value must be different to the value specified in parameter SVC (interregion communication).

Required Parameter Settings

You can decide whether you want to use the features of the security manager interface. The following parameter combination ensures that both the MERVA ESA signon and the usage of the MERVA ESA user file maintenance function are protected by a security manager:

- EXSEC=YES
- EXUID=YES
- PGCALL=YES

For an MVS installation, if MERVA ESA is not running under CICS/ESA V4.1 or later, you must also set the parameter RACFSVC=nnn. This lets the security manager protect the usage of the MERVA ESA user file maintenance function.

If you specify NO for any of the parameters EXSEC, EXSEC, or PGCALL, you switch off the security manager interface features. That is:

- Users sign on to MERVA ESA via the signon panel. This applies if you started the end user driver by entering the transaction code DSLP from the terminal. Note that the end user driver must accept the start from the terminal. This requires that EXSEC=YES is not specified in combination with EXUID=NO or PGCALL=NO.
- Users enter a MERVA ESA rather than a security manager recorded password for the user file maintenance function (provided that USFPW=YES is specified).

Defining an Authorized User

Before you can use the security manager to control access to MERVA ESA, you must create an authorized user via the MERVA ESA user file maintenance. The name of the user file record must be equal to an identifier that is known to the security manager and that is entered at CICS or IMS signon. The USR function at least should be assigned to the user to create further user file records.

Transparent Usage of the DSLEUD

When the security manager interface was not available, you entered one of the transaction codes assigned to the MERVA ESA end-user driver DSLEUD, for example, DSLE or SWEU, to start the end user driver. Now, with the security manager interface, you have to enter the transaction code DSLP when you want to start the end-user driver from the terminal.

If you want to continue to use the transaction code DSLE instead of DSLP, the following modifications enable the transparent use of the end-user driver:

- Specify parameter EUDTRAN=xxxx in the module DSLPRM.

  xxxx represents any valid transaction code assigned to DSLEUD except of DSLE, for example, SWEU.
- Remove DSLE from the system definition.

  In CICS, remove it from the TRANSACTION definitions in the CSD.

  In IMS, remove it from the IMS application definitions.
- Rename DSLP to DSLE.

  In CICS, rename it in the TRANSACTION definitions in the CSD.

  In IMS, rename it in the IMS application definitions. The new definition must be included in the stage-1 input code of the IMS generation.

Now, when you enter DSLE at the terminal, program DSLPTMRV is started as you assigned DSLE to DSLPTMRV.

When DSLPTMRV is running, it starts program DSLEUD using the transaction code specified for parameter EUDTRAN.

Notes:

1. DSLP can only be entered at the terminal. It cannot be used for a program-to-MERVA switch. Program-to-MERVA switches are described in the *MERVA for ESA System Programming Guide*.
2. Parameters following DSLP are ignored when entered.
3. When you assigned a new transaction code to DSLEUD, a program-to-MERVA switch from your application to MERVA ESA might fail if the application still uses the former DSLEUD transaction code. In this case you have to change the application and let it take the correct transaction code from the field NPEUDTRN. This field can be obtained from the DSLPRM DSECT (Assembler only) or by using the MERVA ESA API function FLDG (Assembler, C/370™, COBOL, and PL/I).

Defining the Nucleus Server Table DSLNSVT

To customize the parallel processing in the nucleus server environment of MERVA ESA, the table DSLNSVT is used. All service programs running under control of the nucleus are defined in the tables DSLNPTT, DSLNTRT, and DSLNCMT. These programs are called nucleus servers. The table DSLNSVT defines whether a nucleus server runs under control of DSLNUC or as a separate task. It uses the DSLNSV macro as described in the *MERVA for ESA Macro Reference*.

Sample DSLNSVT Tables

The nucleus server table is loaded by DSLNUC during startup. If found, the contents is interpreted. It consists of a main section and an entry for each service specified to run as a nucleus server. The default table defines that parallel processing is not used. The entries in the nucleus server table are subdivided into

groups. When the customization specifies that nucleus servers should run as separate tasks, you must remove the comments for an entire group. The programs within one group must run together in one task and execute in a synchronous way.

If you specify SERVER=TASK, under IMS the nucleus server is attached as a subtask to the DSLNUC maintask. Under CICS, the nucleus server is started as a CICS task. Under CICS MVS, you can specify SERVER=BATCHTASK. This indicates that the nucleus server runs as an MVS subtask. The advantage is that MVS then relieves CICS of the subtask management. Omit the SERVER parameter for subroutines under control of a nucleus server.

The example below shows the nucleus server table as delivered with the MERVA ESA product. You can modify this table to create your own version.

```
NSVT     TITLE 'MERVA ESA Nucleus Server Table'
         COPY   DSLSYSET                                      [1]
         DSLNSV TYPE=INITIAL                                  [2]
**** Main entry for DSLNUC must be the first definition
         DSLNSV NAME=DSLNUC,SERVER=MAIN                       [3]
         DSLNSV NAME=DSLNCMD
*
**** DSLNMOP must not run as a subtask!
         DSLNSV NAME=DSLNMOP,SERVER=MAIN                      [4]
         DSLNSV NAME=CONSOLE,MODNAME=DSLNMOP
         DSLNSV NAME=DSLNDM
*
************************************************************
* The following programs can be changed to run as tasks.
* Remove the asterisk and the number in the first three columns.
* In each case the whole group of entries must be activated.
************************************************************
**** Journal program group and message counter group
*01      DSLNSV NAME=DSLJRNP,SERVER=TASK                      [5]
*01B     DSLNSV NAME=MSGCOUNT,MODNAME=DSLCNTP, SERVER=TASK
*01B     DSLNSV NAME=DSLCNTDL
**** Queue management group
*02      DSLNSV NAME=DSLQMGT,SERVER=TASK                      [6]
*02      DSLNSV NAME=DSLQLRGC
*02      DSLNSV NAME=DSLQMGTR
*02      DSLNSV NAME=DSLRTRSW
*
**** The command server group
*03      DSLNSV NAME=DSLNCS,SERVER=TASK                       [7]
*
**** The user file program group
*04      DSLNSV NAME=DSLNUSR,SERVER=TASK                      [8]
*04      DSLNSV NAME=DSLNDU
*
**** The remote task communication group
*05      DSLNSV NAME=DSLNRTCP,SERVER=TASK                     [9]
*05      DSLNSV NAME=RTCOMM,MODNAME=DSLNRTCP
*
**** Intertask Communication: Intra/Interregion group
*06      DSLNSV NAME=DSLNTS,SERVER=TASK                       [10]
*06      DSLNSV NAME=BATCH,MODNAME=DSLNTS                     [11]
*06      DSLNSV NAME=TRANSACT,MODNAME=DSLNTS (CICS only)      [12]
*06      DSLNSV NAME=DSLNSHU
*
**** Intertask Communication via CICS TS queues (CICS only)
*07      DSLNSV NAME=CICSSRV,MODNAME=DSLNTSQ,SERVER=TASK      [13]
*
**** Intertask Communication via APPC/MVS server (MVS only)
*08      DSLNSV NAME=APPCSRV,MODNAME=DSLNTSA,SERVER=TASK      [14]
*
**** Intertask Communication via MQSeries (MVS only)
```

```
*09      DSLNSV NAME=MQISRV,MODNAME=DSLNTSM,SERVER=TASK      [15]
*
**** The syncpoint program group
*10      DSLNSV NAME=SYNPOINT,MODNAME=DSLISYNP,SERVER=TASK   [16]
*
**** The SWIFT link program group
*11      DSLNSV NAME=SWIFTII,MODNAME=DWSDGPA,SERVER=TASK     [17]
*11      DSLNSV NAME=SWIFTAUT,MODNAME=DWSAUTIN
*11      DSLNSV NAME=DWSAUTP
*11      DSLNSV NAME=DWSAUTIN
*11      DSLNSV NAME=DWSDCMD
*11      DSLNSV NAME=DWSDCMR
*11      DSLNSV NAME=DWSDIVA
*11      DSLNSV NAME=DWSXTRCE
*11a     DSLNSV NAME=SWIFTIIA,MODNAME=DWSDGPA,SERVER=TASK
*11a     DSLNSV NAME=DWSDCMDA,MODNAME=DWSDCMD
*11b     DSLNSV NAME=SWIFTIIB,MODNAME=DWSDGPA,SERVER=TASK
*11b     DSLNSV NAME=DWSDCMDB,MODNAME=DWSDCMD
*11c     DSLNSV NAME=SWIFTIIC,MODNAME=DWSDGPA,SERVER=TASK
*11c     DSLNSV NAME=DWSDCMDC,MODNAME=DWSDCMD
*
**** The SWIFT link load session key program group
*12      DSLNSV NAME=SWLOADSK,MODNAME=DWSDLSK,SERVER=TASK    [18]
*
**** The Telex link via fault tolerant system program group
*13      DSLNSV NAME=TELEX,MODNAME=ENLSTPL,SERVER=TASK       [19]
*13      DSLNSV NAME=ENLCMDL
*
**** The nucleus server display and trace command group
*14A     DSLNSV NAME=DSLNDR,SERVER=TASK                      [20]
*14B     DSLNSV NAME=DSLNDRR,SERVER=TASK
*14C     DSLNSV NAME=DSLNDRQA,SERVER=TASK
*14D     DSLNSV NAME=DSLNTRC,SERVER=TASK                     [21]
*
         DSLNSV TYPE=FINAL                                   [22]
         END
```

Notes:

[1] This must be the first macro definition. It specifies the globals for the environment for which the table is to be compiled for.

[2] This must be the second macro definition.

[3] This must be the third macro definition. The nucleus must always be specified to run as the maintask. It specifies the services the nucleus still provides plus specific command execution routines.

[4] The operator console interface must also run in the maintask. Note that the name CONSOLE, specified as DESC parameter in the DSLNPTT, must also be specified. The operator message processing has its own command execution routine to display the operator messages and must also be specified.

[5] The journal processing is an independent central service and is a good choice for a nucleus server to run as a subtask. The MSGCOUNT nucleus server also processes message counters and the command execution routine for the DCLOG command.

[6] The queue management is an independent central service and is a good choice for a nucleus server to run as a subtask. The other members of this group are command execution routines to switch the queue trace, the routing trace, and for the LMC display functions.

[7] The command service is a good choice for a nucleus server to run as a subtask. It processes the MERVA ESA commands which are entered via the operator console.

[8] This module does the user file processing. It is an independent service which is a good candidate for a nucleus server to run as a subtask.

[9] This service provides the Remote Task Communication. It is defined in the DSLNPTT with DESC=RTCOMM and in the DSLNTRT with the program name DSLNRTCP. Therefore both names must be defined in this group.

[10] The intertask communication group consists of nucleus servers used for communication between the nucleus and the various interfaces such as the end-user driver, the hardcopy print routine, and the sequential data set input and output processing. Note that it consists also of the command execution routine for shutdown and reshutdown.

[11] To define a nucleus server to use interregion communication, the name BATCH, as defined in the DSLNPTT, must be specified.

[12] When running under CICS, you should define the intraregion task server, which is defined in the DSLNPTT with DESC=TRANSACT.

[13] This intertask communication nucleus server uses the CICS temporary storage queue method. Do not specify it if running under IMS.

[14] This intertask communication nucleus server uses the APPC/MVS method. Do not specify it if running under VSE.

[15] This intertask communication nucleus server uses MQSeries. Do not specify it if MERVA ESA is running under VSE.

[16] This nucleus server creates a syncpoint for the nucleus when running under CICS or as a BMP under IMS.

[17] This group defines the nucleus server for the SWIFT Link. It consists of the general purpose application, the authenticator key file support and its initialization, and command execution routines for SWIFT Link commands. The SWIFT Link is defined in the DSLNPTT with DESC=SWIFTII. The nucleus servers required to use multiple instances of SWIFT Link are also defined here. For further information about using multiple instances for SWIFT Link, see “Chapter 3. The SWIFT Link” on page 127.

[18] This nucleus server loads the SWIFT session keys.

[19] This nucleus server processes the Telex Link. It consists also of a command execution routine for Telex commands.

[20] These command services process display commands used for problem diagnosis in the parallel processing of the nucleus. They are good candidates to run as subtasks in the case they are needed.

[21] The nucleus trace service is used for debugging only. If needed, it is a good candidate to run as a subtask.

[22] This must be the last macro definition.

An example of the nucleus server table using parallel processing is shown below.

```
NSVT     TITLE 'MERVA ESA Nucleus Server Table'
         COPY   DSLSYSET
         DSLNSV TYPE=INITIAL
**** Main entry for DSLNUC must be the first definition
         DSLNSV NAME=DSLNUC,SERVER=MAIN
         DSLNSV NAME=DSLNCMD
*
**** DSLNMOP must not run as a subtask!
         DSLNSV NAME=DSLNMOP,SERVER=MAIN
         DSLNSV NAME=CONSOLE,MODNAME=DSLNMOP
         DSLNSV NAME=DSLNDM
*
************************************************************
**** Journal program group
         DSLNSV NAME=DSLJRNP,SERVER=TASK                    [23]
*
**** Queue management group
         DSLNSV NAME=DSLQMGT,SERVER=TASK                    [24]
         DSLNSV NAME=DSLQLRGC
         DSLNSV NAME=DSLQMGTR
         DSLNSV NAME=DSLRTRSW
*
**** Intertask Communication via APPC/MVS server     MVS only
         DSLNSV NAME=APPCSRV,SERVER=BATCHTASK,PRIO=-1       [25]
         DSLNSV NAME=APPCSRV1,SERVER=BATCHTASK,PRIO=-2      [26]
*
**** The SWIFT link program group
         DSLNSV NAME=SWIFTII,MODNAME=DWSDGPA,SERVER=TASK    [27]
         DSLNSV NAME=SWIFTAUT,MODNAME=DWSAUTIN
         DSLNSV NAME=DWSAUTP
         DSLNSV NAME=DWSAUTIN
         DSLNSV NAME=DWSDCMD
         DSLNSV NAME=DWSDIVA
         DSLNSV NAME=DWSXTRCE
*
         DSLNSV TYPE=FINAL
         END
```

Figure 23. Example of a MERVA ESA Nucleus Server Table Using Parallel Processing

Notes:

[23] The journal service runs as a separate task.

[24] The queue management service runs as a separate task.

[25] The intertask communication server using APPC/MVS runs as a separate task. For MERVA ESA running under CICS MVS, this nucleus server runs as an MVS task. The priority is one less than the priority of DSLNUC.

[26] This installation has defined another intertask communication server for APPC/MVS which also runs as a separate task. This server must be defined in the DSLNPTT with the DESC=APPCSRV1 parameter. Defining more than one intertask communication server can improve the throughput when using the intertask communication via APPC/MVS.

[27] The SWIFT Link network driver runs as a separate task.

Defining the Parameters for Using QDS on DB2

To customize MERVA ESA for queue management using DB2®, the following parameters must be set in the parameter module DSLPRM:

QIO=DSLQMDIO DSLQMDIO is the queue management DB2 I/O module

DB2SS=subsystem	DB2 subsystem name (applies to DB2 MVS only)
DB2PLB=planname	DB2 plan name for the batch nucleus program (applies to DB2 MVS only)

The parameters QDS and LRGMSG are not applicable when using QDS on DB2 and are ignored. To use direct queue management in batch or API programs, specify the parameter SDDB2=YES.

Customizing MERVA ESA Intertask Communication Using MQSeries

If you run MERVA ESA in a multisystem environment with shared resources and a MERVA ESA application runs on a different system from MERVA ESA, you must use an intertask communication method other than the traditional intertask communication (the default), which allows communication only between address spaces.

The following methods are available under MVS only:

- Intertask communication using APPC
- Intertask communication using MQSeries

The following sections describe how to customize intertask communication using MQSeries.

Parameters for Intertask Communication Using MQSeries

If you want to use intertask communication using MQSeries, the second value of the ITC parameter in module DSLPRM must be set to MQI. The following queue name definitions must be specified:

- ITCMSND
- ITCMRTQ
- ITCMRCV

Notes:

1. The queue names you specify for intertask communication must match the MQI queue definitions.
2. To allow for multiple independent communication with MERVA ESA using a common DSLPRM, define a model queue for the reply-to queue. The name of this model queue has to be specified as the value of the first ITCMRTQ subparameter. For the second subparameter, specify a name with a prefix followed by an asterisk. This allows the local message queue manager to generate a dynamic message queue with a unique name for replies returned by MERVA ESA to the various applications.

```
DSLPARM
      ...
       MQMNAME=CSQ1,                                    * [1]
       ITC=(TSQ,MQI),                                   * [2]
       ITCMSND=(DSL.SYS1.ITC.SEND.NUC1),                * [3]
       ITCMRCV=(DSL.SYS1.ITC.RECEIVE.ALL),              * [4]
       ITCMRTQ=(DSL.SYS1.ITC.REPLY.ALL),                * [5]
       ITCMWTT=30000,                                   * [6]
       ...
```

Figure 24. DSLPARM Parameters for Intertask Communication Using MQSeries

Notes:

[1] MQMNAME specifies the name of the local MQSeries queue manager. The name is different for each system in a multisystem environment. This parameter is required if you want to use an MQSeries queue manager other than the default MQSeries queue manager in your installation. Note that not all installations provide a default MQM.

[2] ITC specifies the MERVA ESA intertask communication method to be used. The first subparameter applies to CICS transactions. The value TSQ specifies that CICS temporary storage queues should be used instead of the direct buffer move operation. The second subparameter applies to MVS programs not running as CICS transactions. The value MQI specifies that MQSeries should be used.

[3] ITCMSND specifies the name of an MQI send queue used by the intertask communication using MQSeries. This is the queue to which service requests are put. This specification is required when an application, for example, DSLSDI runs in its own address space.

[4] ITCMRCV specifies the name of an MQI receive queue used by the intertask communication using MQSeries. This specification is required by the task server for MQSeries on the primary MERVA ESA instance. This is the queue from which service requests are retrieved by MERVA ESA.

If there are multiple entries for DSLNTSM specified in the DSLNPTT and the PARM parameter has different values (for example, 1 for the first entry, 2 for the second entry), then the constructed queue name is DSL.SYS1.ITC.RECEIVE.ALL.1 for the first nucleus task server for MQSeries, and DSL.SYS1.ITC.RECEIVE.ALL.2 for the second nucleus task server for MQSeries.

[5] ITCMRTQ specifies the name of an MQI reply-to queue used by the intertask communication via MQSeries. This is the queue from which request responses are retrieved. This specification is required when an application, for example, DSLSDI runs in its own address space.

[6] ITCMWTT specifies the maximum period an application should wait for a response to a service request. In this example a response is expected within 30 seconds.

Customizing the Nucleus Program Table (DSLNPTT)

Intertask communication for MQSeries requires that the MERVA ESA MQSeries nucleus task server program (DSLNTSM) be defined in the nucleus program table. This program is defined in an entry in the copy book DSLNPTTC as shown in Figure 25 on page 100. This copy book is included in module DSLNPTT.

```
      ...
* Intertask communication: MQSeries nucleus task server
*  - To run this server, the server parameters in DSLPRM are needed.
*  - The server is reentrant, multiple servers can be defined.
*    The PARM parameter creates multiple server instances,
*    the DESC descriptor must then be different for each server.
*  - The ECB number is one.
*  - AUTO=NO requires operator command 'S MQISRV1' to start the server
         DSLNPT TYPE=PGM,NAME=DSLNTSM,STRTREQ=4,STOPREQ=8,LANG=HLL,    *
               DESC=MQISRV1,ECB=1,PRTY=5,AUTO=YES,PARM=
      ...
```

Figure 25. Specifying the MERVA ESA MQSeries Nucleus Task Server in the Nucleus Program Table

Customizing the Nucleus Server Table

Intertask communication for MQSeries requires the MERVA ESA MQSeries Nucleus Task Server to run on the primary MERVA ESA instance. This is specifed in the Nucleus Server Table (DSLNSVT). Refer to Figure 28 on page 104 for an example.

Defining MQI Queues

You have to define all MQI queues for intertask communication named in DSLPARM and the MQI channels to MQSeries. Refer to "Defining MQI Queues" on page 107 and to "MQI Queue Examples" on page 108.

Customizing MERVA ESA Interservice Communication

The new interservice communication gives you more flexibility to run MERVA ESA. You can run more than one MERVA ESA instance, one primary and one or more secondary MERVA ESA instances. The primary MERVA ESA instance gets all the service requests according to the nucleus server table definition. A service request can be processed by the primary instance or distributed to a secondary MERVA ESA instance. Data and control information are passed using MQSeries for MVS/ESA™. A message is put into the MQI receive queue of the MERVA ESA instance that provides the requested service. After the request has been processed the result is passed back as a reply message to the MQI reply-to queue of the requester.

The XCF signalling services of MVS/ESA are used for the exchange of status information and failure notification.

Figure 26 on page 101 shows a scenario with three MERVA ESA instances, each running on a different MVS system. You can also run more than one MERVA ESA instance on the same MVS system. In this scenario, NUC1 is defined as primary and runs on System1, NUC2 is defined as secondary and runs on System2, and NUC3 is defined as secondary and runs on System3.

To use interservice communication:

1. Define the new parameters for interservice communication in the MERVA ESA parameter module DSLPRM.
2. Modify the nucleus server table (DSLNSVT) to define the services provided by a nucleus.
3. Set up the MQSeries resources. See Table 5 on page 108 for a single system environment and Figure 29 on page 109 for a multisystem environment.

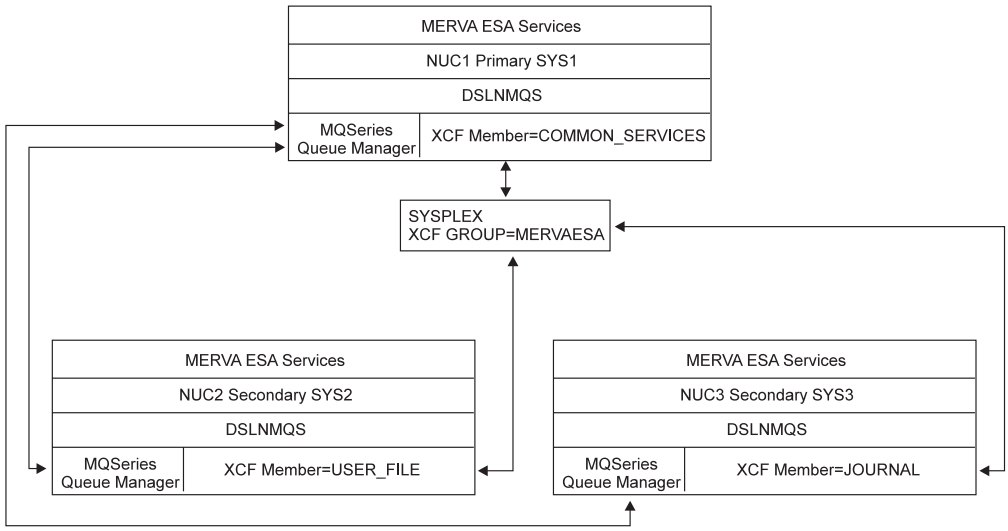

Figure 26. MERVA ESA in a Parallel Sysplex Environment

Defining the Parameters for Interservice Communication Using MQSeries

Traditionally, MERVA ESA runs on a single system. Specifications in the nucleus server table determine whether a single service runs within the nucleus or as a subtask. Specifying subtasks allows execution of services in parallel.

If you run MERVA ESA in a multisystem environment with shared resources, you can distribute MERVA ESA services among several systems. In this case, the MERVA ESA nucleus is separated into MERVA ESA instances each running on a different system. All MERVA ESA instances communicate via interservice communication using MQSeries.

If you decide to do this, you have to consider:

- The system which communicates with applications via intertask communication must always be defined as the primary MERVA ESA instance. All others must be defined as a secondary MERVA ESA instance. There can only be one primary MERVA ESA instance, while there can be multiple secondary MERVA ESA instances. You define this in the ISCNUC parameter.
- Each MERVA ESA instance requires the following MQI queue definitions in DSLPRM:
  - ISCMSND
  - ISCMRTQ
  - ISCMRCV

Notes:

1. The queue names you specify for interservice communication must match the MQI queue definitions.
2. The name you specify for the queue in the ISCMSND parameter is only the base name. The complete queue name is constructed by preceding or appending the value of the QNAME parameters in the nucleus server table. You specify only one base name, but you use as many MQI send queues as there are QNAMEs. You have to consider this when defining your MQSeries environment.

```
DSLPARM                                                 *
      ...
      MQMNAME=CSQ1,                                     * [1]
      ISCNUC=PRIMARY,                                   * [2]
      ISCMQID=(NUC1,NUC1,SUFFIX),                       * [3]
      ISCMSND=(DSL.SYS1.SEND),                          * [4]
      ISCMRCV=(DSL.SYS1.RECEIVE.NUC1),                  * [5]
      ISCMRTQ=(DSL.SYS1.REPLY_TO.NUC1),                 * [6]
      ISCSTART=AUTO,                                    * [7]
      ISCXCF=(MERVAESA,COMMON_SERVICES),                * [8]
      ISCXJWT=1000                                        [9]
END
```

Figure 27. DSLPRM Parameters for Intertask Communication

Notes:

[1] MQMNAME specifies the name of the local MQSeries queue manager. The name is different for each system in a multisystem environment. This parameter is required if you want to use an MQSeries queue manager other than the default MQSeries queue manager in your installation. Note that not all installations provide a default MQM.

[2] ISCNUC specifies that the local MERVA ESA instance is primary.

You specify PRIMARY if the local MERVA ESA instance uses intertask communication with your applications. The primary MERVA ESA instance also should run commonly used services. You specify SECONDARY if the local MERVA ESA instance runs a specific service, such as the user file maintenance.

[3] ISCMQID specifies MERVA ESA interservice communication. Prerequisite is that MERVA ESA runs on a multisystem environment. The following subparameters must be specified:

- The MERVA ID of the primary MERVA ESA instance. This ID is used for internal communication during initialization.
- The MERVA ID of the local MERVA ESA instance. This ID is compared with the QNAME value of the nucleus server table entry if a service is requested. If the names match, the service is in the local instance. Otherwise, the service request is directed to the MERVA ESA instance indicated by the QNAME.
- PREFIX or SUFFIX. The QNAME value is always part of the static MQI send queue. If you specify PREFIX, the QNAME value from the nucleus server table precedes the base name specified for the MQI send queue. If you specify SUFFIX, the QNAME value follows the base name.

In this example, the MERVA ID of the primary and the local MERVA ESA instance are identical; the local MERVA ESA instance is the primary.

[4] ISCMSND specifies the base name of an MQI send queue used for interservice communication. This specification is needed by the MQSeries of any MERVA ESA instance sending service request messages to another MERVA ESA instance. There are as many queue names assembled as there are different QNAME definitions in the nucleus server table. Figure 28 on page 104 contains three different QNAME definitions (NUC1, NUC2, NUC3), so three MQI send queues are generated (DSL.SYS1.SEND.NUC1, DSL.SYS1.ISC.SEND.NUC2, DSL.SYS1.ISC.SEND.NUC3).

[5] ISCMRCV specifies the name of an MQI receive queue used for

interservice communication. This specification is needed by the MQSeries of any MERVA ESA instance receiving service request messages.

[6] ISCMRTQ specifies the name of an MQI reply-to queue used for interservice communication. This specification is needed by the MQSeries of any MERVA ESA instance sending or receiving a reply message that consists of an MQI service response or exception report message.

[7] ISCSTART indicates to the primary MERVA ESA instance that it should start the secondary MERVA ESA instances automatically. This requires a cataloged procedure for each MERVA ESA instance and the appropriate system and security definitions. The name of the procedure to start and the system ID where the MERVA ESA instance is to be started is defined in the nucleus server table entry for the appropriate service.

[8] ISCXCF allows participation in XCF signalling facilities. You define an XCF group name and an XCF group member name. The first MERVA ESA instance started creates the named XCF group, which all other MERVA ESA instances then can join. In this example, the member name COMMON_SERVICES has been chosen to emphasize the tasks assigned to the primary MERVA ESA instance.

MERVA ESA uses XCF signalling in the case of a service failure, a MERVA ESA instance failure, or a complete system failure to inform all joined MERVA ESA instances to avoid sending request messages to services no longer available.

[9] ISCXJWT specifies the period the primary MERVA ESA instance waits during initialization until all MERVA ESA instances which are members of the XCF group with the name **MERVAESA** have joined. MERVA ESA instances which have not joined within the specified time do not participate in the MERVA ESA failure recovery. In this example, secondary MERVA ESA instances which do not join the XCF group with the name **MERVAESA** within ten seconds do not participate in XCF signalling.

Customizing the Nucleus Server Table for a Multisystem Environment

An example of the nucleus server table using parallel processing in a multisystem environment is shown below.

Note: It is important that the same nucleus server table be used by all MERVA ESA instances.

```
NSVT     TITLE 'MERVA ESA Nucleus Server Table'
         COPY   DSLSYSET
         DSLNSV TYPE=INITIAL
*************************************************************************
*     Main entry of the MERVA ESA nucleus.
*     Must be the first definition.
*************************************************************************
*00                                                                [1]
         DSLNSV NAME=DSLNUC,SERVER=MAIN
         DSLNSV NAME=DSLNCMD
*************************************************************************
* The Operator message processing is part of the base nucleus
* and must not run as a separate task.
*************************************************************************
*00                                                                [2]
         DSLNSV NAME=DSLNMOP,SERVER=MAIN
         DSLNSV NAME=CONSOLE,MODNAME=DSLNMOP
         DSLNSV NAME=DSLNDM
*
*************************************************************************
* The MQSeries server must always run as a separate task.
*************************************************************************
*01                                                                [3]
         DSLNSV NAME=MQSSRV,MODNAME=DSLNMQS,SERVER=TASK                *
*
```

Figure 28. Nucleus Server Table for a Multisystem Environment (Part 1 of 3)

```
*********************************************************************
* The following programs can be changed to run as tasks.
* You can also specify a MERVA ID as QNAME to define on which
* MERVA ESA instance the service has to run.
* Remove the asterisk and the number in the first three columns.
* In each case the whole group of entries must be activated.
*********************************************************************
*
**** Journal program group ***************************************
*01A                                                        [4]
         DSLNSV NAME=DSLJRNP,SERVER=TASK
               QNAME=NUC3,                                          *
               SYSNAME=SYS3,STCNAME=MERVAESA
         DSLNSV NAME=DSLJRNC,SERVER=TASK,                           *
               QNAME=NUC3,                                          *
               SYSNAME=SYS3,STCNAME=MERVAESA
*
**** Message counter group ***************************************
*01B
         DSLNSV NAME=MSGCOUNT,MODNAME=DSLCNTP,SERVER=TASK,          *
               QNAME=NUC1,                                          *
               SYSNAME=SYS1,STCNAME=MERVAESA
         DSLNSV NAME=DSLCNTDL
*
**** Intertask Communication: Intra/Interregion group ************
*02A                                                        [5]
         DSLNSV NAME=DSLNTS,SERVER=TASK,                            *
               QNAME=NUC1,                                          *
               SYSNAME=SYS1,STCNAME=MERVAESA
         DSLNSV NAME=BATCH,MODNAME=DSLNTS
         DSLNSV NAME=TRANSACT,MODNAME=DSLNTS (CICS only)
         DSLNSV NAME=DSLNSHU
*
****     Intertask Communication via CICS TS queues (CICS only)
*02B                                                        [6]
         DSLNSV NAME=CICSSRV,MODNAME=DSLNTSQ,SERVER=TASK,           *
               QNAME=NUC1,                                          *
               SYSNAME=SYS1,STCNAME=MERVAESA
*****     Intertask Communication via APPC/MVS server (MVS only)
*02C                                                        [7]
         DSLNSV NAME=APPCSRV,MODNAME=DSLNTSA,SERVER=TASK,           *
               QNAME=NUC1,                                          *
               SYSNAME=SYS1,STCNAME=MERVAESA
*
****     Intertask Communication via MQSeries server (MVS only)
*02D1                                                       [8]
         DSLNSV NAME=MQISRV1,MODNAME=DSLNTSM,SERVER=TASK,           *
               QNAME=NUC1,                                          *
               SYSNAME=SYS1,STCNAME=MERVAESA
*02D2    DSLNSV NAME=MQISRV2,MODNAME=DSLNTSM,SERVER=TASK,    [9]
*              QNAME=NUC1,
*              SYSNAME=SYS1,STCNAME=MERVAESA
*
**** The syncpoint program group                             *********
*07      DSLNSV NAME=SYNPOINT,MODNAME=DSLISYNP,SERVER=TASK   [10]
*
```

Figure 28. Nucleus Server Table for a Multisystem Environment (Part 2 of 3)

```
**** The command server group ***************************************
*08                                                          [11]
         DSLNSV NAME=DSLNCS,SERVER=TASK,                              *
               QNAME=NUC1,                                            *
               SYSNAME=SYS1,STCNAME=MERVAESA
*
**** The user file program group ************************************
*09                                                          [12]
         DSLNSV NAME=DSLNUSR,SERVER=TASK,                             *
               QNAME=NUC2,                                            *
               SYSNAME=SYS2,STCNAME=MERVAESA
         DSLNSV NAME=DSLNDU
               QNAME=NUC2,                                            *
               SYSNAME=SYS2,STCNAME=MERVAESA
*
**** Queue management group *****************************************
*11                                                          [13]
         DSLNSV NAME=DSLQMGT,SERVER=TASK,                             *
               QNAME=NUC1,                                            *
               SYSNAME=SYS1,STCNAME=MERVAESA
         DSLNSV NAME=DSLQLRGC
         DSLNSV NAME=DSLQMGTR
         DSLNSV NAME=DSLRTRSW
***** The SWIFT link program group ************************************
*12                                                          [14]
         DSLNSV NAME=SWIFTII,MODNAME=DWSDGPA,SERVER=TASK,             *
               QNAME=NUC1,                                            *
               SYSNAME=SYS1,STCNAME=MERVAESA
         DSLNSV NAME=SWIFTAUT,MODNAME=DWSAUTIN
         DSLNSV NAME=DWSAUTP
         DSLNSV NAME=DWSAUTIN
         DSLNSV NAME=DWSDCMD
         DSLNSV NAME=DWSDIVA
         DSLNSV NAME=DWSXTRCE
*
**** The SWIFT link load session key program group ******************
*13                                                          [15]
         DSLNSV NAME=SWLOADSK,MODNAME=DWSDLSK,SERVER=TASK,            *
               QNAME=NUC1,                                            *
               SYSNAME=SYS1,STCNAME=MERVAESA
*
**** The Telex link via fault tolerant system program group ***********
*14                                                          [16]
*        DSLNSV NAME=TELEX,MODNAME=ENLSTPL,SERVER=TASK,
*              QNAME=NUC1,
*              SYSNAME=SYS1,STCNAME=MERVAESA
*        DSLNSV NAME=ENLCMDL
*
**********************************************************************
         DSLNSV TYPE=FINAL                                   [17]
         END
```

Figure 28. Nucleus Server Table for a Multisystem Environment (Part 3 of 3)

Notes:

[1] This must be the first definition.

[2] This must be the second definition.

[3] This defines the module name of the MQSeries, and must be the third definition. Due to internal architecture constraints, this nucleus server must be defined to run as a subtask. The name you give this server is up to you.

[4] The primary MERVA ESA instance, which has the MERVA ID NUC1,

performs common tasks such as message counting, and is assigned to SYS1. The journal nucleus server, which has the MERVA ID NUC3, is assigned to SYS3.

[5] Nucleus task servers must be defined at the primary MERVA ESA instance. The QNAME NUC1 corresponds to the first value of the ISCMQID parameter in DSLPRM in Figure 27 on page 102. This is the nucleus task server for applications or jobs which run on the same system and can therefore communicate via the standard intertask communication.

[6] This is the nucleus task server for applications running in a CICS region using the CICS temporary storage queue.

[7] This nucleus task server can be defined if applications running on another system communicate with MERVA ESA via APPC/MVS.

[8] This nucleus task server can be defined if applications running on another system communicate with MERVA ESA via MQSeries.

[9] This is the second nucleus task server. You can define more nucleus task servers if your traffic is heavy. Because each nucleus task server has its own message queues, the queue names must be different. You do this by specifying a value in the PARM parameter of the entry in the DSLNPTT. This value is appended to the MQI receive queue name of the nucleus task server as specified in the ITCMRCV parameter of the DSLPRM.

[10] Syncpointing is also considered a common task.

[11] The command nucleus server is the interface to the MERVA ESA operator and is considered a common task. Commands can be forwarded to other instances as necessary. For example, SWIFT Link commands are forwarded to the MERVA ESA instance running SWIFT Link.

[12] The user file services are located on MERVA ESA instance two.

[13] The queue management nucleus server is on MERVA ESA instance one. All requests to this nucleus server are passed to this MERVA ESA instance.

[14] The SWIFT Link nucleus server is on MERVA ESA instance one.

[15] The Load Session Key nucleus server has great affinity to SWIFT Link and therefore runs on the same MERVA ESA instance.

[16] The TELEX Link nucleus server is on MERVA ESA instance one.

[17] This must be the last definition.

Defining MQI Queues

The member DSLKISC1 COPY in the sample library contains the MQI queue definitions for system SYS1. To update your environment:

1. In the job for SYS1, change the MQI channel and queue names to the names that will be constructed by MERVA ESA as described above.
2. Repeat the previous step for SYS2.
3. Use the modified members as input for the MQSeries utilitiy program CSQUTIL.

MQI Send Queues

An MQI send queue is defined to MQSeries either as a local queue or as the local definition of a remote queue. For a local queue, the following attributes must be specified in the DEFINE **QLOCAL** command:

- PUT(ENABLED)

- NOTRIGGER

For a remote queue defined locally, the following attributes must be specified in the DEFINE **QREMOTE** command:

- PUT(ENABLED)
- XMITQ(...)

For details, see the *MQSeries Command Reference*.

MQI Receive Queues

Receiving a message from MQSeries means that the program retrieves a message from the MQI receive queue and processes the message according to the request type. The parameter **ISCMRCV** in DSLPRM contains the names of the MQI receive queues.

The MQI receive queue is always defined to MQSeries as a local queue. The attribute GET(ENABLED) must be specified in the DEFINE **QLOCAL** command.

MQI Reply-To Queue

The MQI reply-to queue is a queue for sending and receiving responses to request messages. On the response receiving side, the reply-to queue is defined to MQSeries as a local queue. The same attributes of the DEFINE QLOCAL command apply as for normal receive queues.

MQI Queue Examples

Interservice Communication on the Same System

If you run NUC1 and NUC2 on the same system, you need an MQI send-receive queue and an MQI reply-to queue for each one. The following steps are performed for a service request from NUC1:

1. NUC1 puts a request message into the DSL.SEND_RECEIVE.NUC2 queue.
2. NUC2 processes the request and puts the reply message into the queue with the name DSL.REPLY_TO.NUC1.
3. NUC1 retrieves the reply message from the MQI reply-to queue.
4. NUC1 passes the result to the requesting application.

Table 5. MQSeries Message Flow in a Single System Environment

Primary NUC1	Secondary NUC2
DSL.SEND_RECEIVE.NUC1	DSL.SEND_RECEIVE.NUC2
DSL.REPLY_TO.NUC1	DSL.REPLY_TO.NUC2

Interservice Communication between Different Systems

If nuclei run in different systems, the MQI send queue must be the local definition of a remote queue. You also need to define MQSeries channels between the queue managers in the various systems. In the following figures, the name of the MQSeries queue manager on SYS1 is Q912, on SYS2 is Q921, and on SYS3 is Q931.

For example, DSL.CHAN12 is defined as a sender channel in SYS1 and DSL.CHAN12 is the corresponding receiver channel defined in SYS2. For a service request from NUC1 and NUC2, the following steps are performed:

1. The request message is put into the DSL.SYS1.SEND.NUC2 queue.
2. The request message is passed over the MQI transmission queue Q921 and the channel DSL.CHAN12 to the DSL.SYS2.RECEIVE.NUC2 queue.

3. NUC2 receives the request message, processes it, and sends the result back to NUC1 into the queue with the name DSL.SYS1.REPLY_TO.NUC1.
4. NUC1 receives the reply message and passes the result to the requesting application.

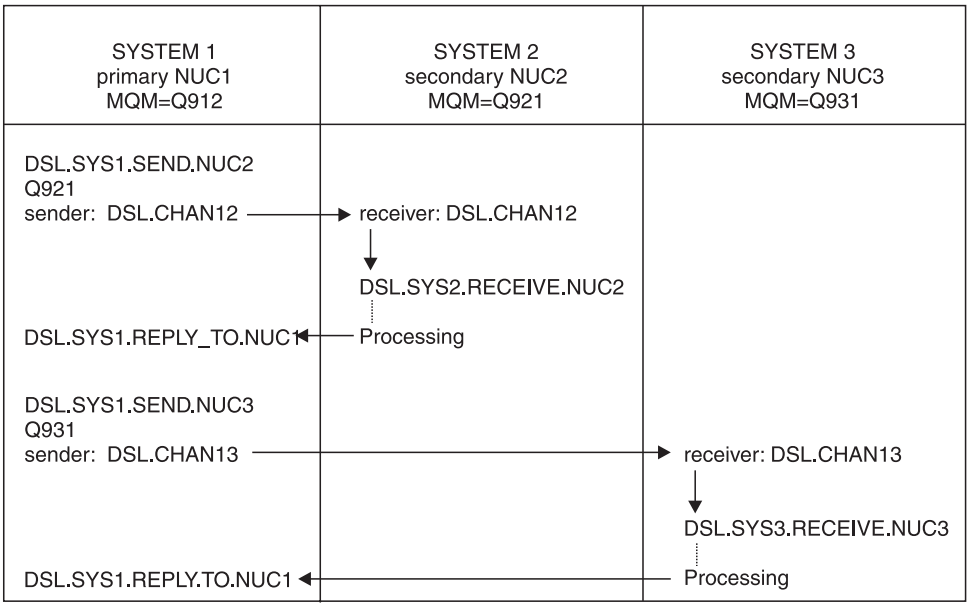

Figure 29. MQSeries Message Flow in a Multisystem Environment When the Requester is NUC1

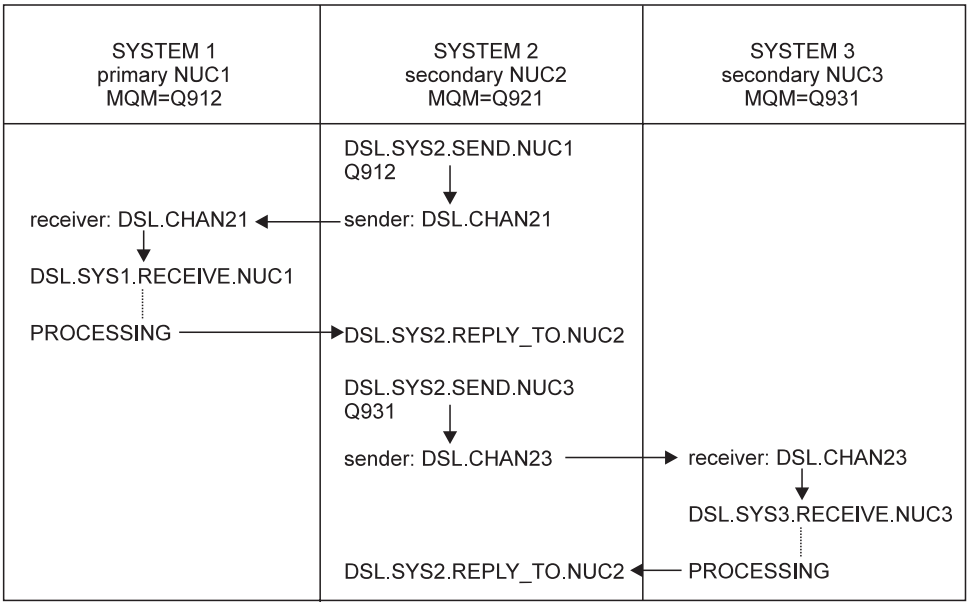

Figure 30. MQSeries Message Flow in a Multisystem Environment When the Requester is NUC2

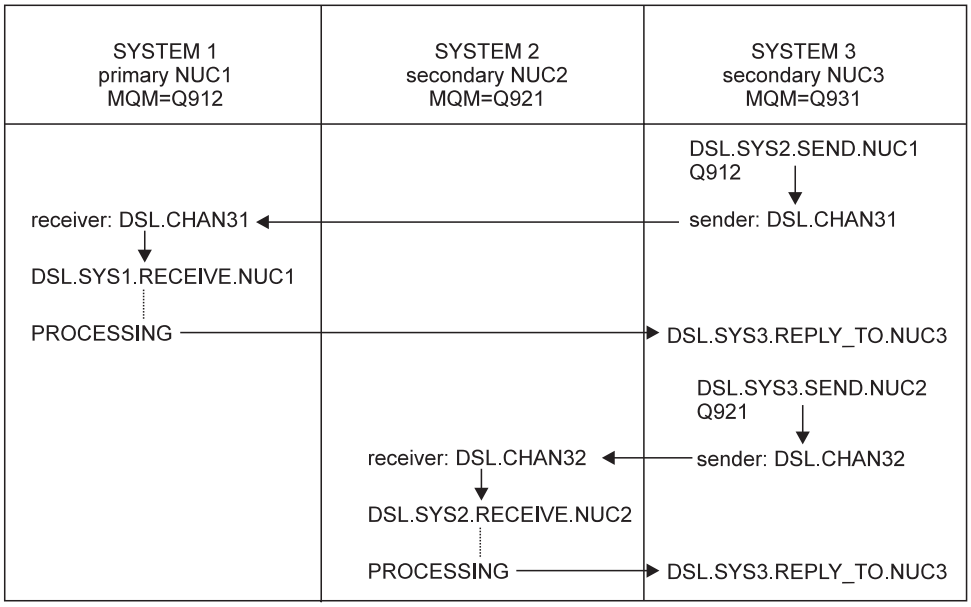

Figure 31. MQSeries Message Flow in a Multisystem Environment When the Requester is NUC3

MQI Send Queue Names

MERVA ESA constructs a unique MQI send queue name for each system by concatentating:

- The value of the parameter **ISCMSND** in the customization parameter module DSLPRM
- The value of the parameter **QNAME** in the nucleus server table DSLNSVT

Depending on the value of the third subparameter of ISCMQID, the value of the parameter **QNAME** is used as either a suffix or a prefix.

Table 6 on page 111 shows an example of how send queue names are constructed.

Table 6. How the MQI Send Queue Names Are Constructed

System 1	System 2	System 3
Type: Primary	Type: Secondary	Type: Secondary
QNAME: NUC1	QNAME: NUC2	QNAME: NUC3
Queue Manager: Q912	Queue Manager: Q921	Queue Manager: Q931
In DSLPRM: ISCMQID=(NUC1,NUC1,SUFFIX), ISCMSND=DSL.SYS1.SEND,	In DSLPRM: ISCMQID=(NUC1,NUC2,SUFFIX), ISCMSND=DSL.SYS2.SEND,	In DSLPRM: ISCMQID=(NUC1,NUC3,SUFFIX), ISCMSND=DSL.SYS3.SEND,
In DSLNSVT: QNAME=NUC1 . . . QNAME=NUC2 . . . QNAME=NUC3	In DSLNSVT: QNAME=NUC1 . . . QNAME=NUC2 . . . QNAME=NUC3	In DSLNSVT: QNAME=NUC1 . . . QNAME=NUC2 . . . QNAME=NUC3
Resulting send queue names: DSL.SYS1.SEND.NUC2 DSL.SYS1.SEND.NUC3	Resulting send queue names: DSL.SYS2.SEND.NUC1 DSL.SYS2.SEND.NUC3	Resulting send queue names: DSL.SYS3.SEND.NUC1 DSL.SYS3.SEND.NUC2

Defining the Transaction Table DSLTXTT

The transaction table is, in a way, an extension of the function table. The function table specifies which transaction is to be started when a message enters the queue. The transaction table does the same, but in the transaction table the transactions can be started in environments other than that in which the nucleus runs.

You can run the MERVA ESA nucleus in any of the following ways:

- As an MVS batch program
- As an IMS BMP
- As a CICS transaction

Regardless of how it was started, the nucleus can then start transactions in any of the following ways:

- As transactions in an IMS message-processing program (MPP) region
- As CICS transactions

If you run the nucleus as an MVS batch program or as an IMS BMP and start a transaction as a CICS transaction:

- The batch job where the MERVA ESA nucleus runs must define the data set containing the external CICS interface modules in its steplib sequence; for example:

  ```
  //STEPLIB DD  DISP=SHR,DSN=MERVA.SDSLLODB
  //        DD  DISP=SHR,DSN=CICS410.SDFHEXCI
  ```

- The CICS region where the started transactions are to run must be defined with the initialization parameters IRCSTRT=YES and ISC=YES.
- The CICS RDO group DFH$EXCI, which contains the external interface definitions, must be added to the list used for the startup of the CICS region; for example, by specifying the following as input to the batch utility DFHCSDUP:

  ```
  ADD GROUP(DFH$EXCI) LIST(DSLLIST)
  ```

- Make sure there is no local MERVA ESA nucleus running in the CICS region itself, or else the transactions will communicate with this nucleus rather than the batch nucleus that started the transaction. To ensure that the transaction uses the interregion communication method to communicate with the MERVA ESA nucleus, the customization parameter module DSLPRM in this CICS region must specify the parameter CINTER=YES.

If you start a transaction locally (that is, as a CICS transaction in the same CICS where the nucleus is running, or in an IMS MPP when the nucleus was started as an IMS BMP), then the transaction table (DSLTXTT) is optional. However, if the nucleus is running as an MVS batch program, or if you start a transaction remotely (that is, on a system other than that where the nucleus is running), then there must be an entry in the transaction table (DSLTXTT) whose NAME parameter has the same value as the parameter TRAN in the function table entry. If the table is not available or there is no corresponding entry in the table, the transaction is started based on the information in the function table, and might fail.

As in previous releases, the MERVA ESA source library contains a sample transaction table (DSLTXTT). The sample table has no active entries, therefore MERVA ESA starts transactions according to the information in the function table.

To create an entry, use the macro **DSLTXT** which is described in the *MERVA for ESA Macro Reference*, and the samples in "DSLTXT Sample Definitions". As shown in Figure 32, ***USR1*** and ***USR3*** have entries in the transaction table that is used to start the requested transaction. ***USR2*** does not have an entry in the transaction table; so it is started according to the function table.

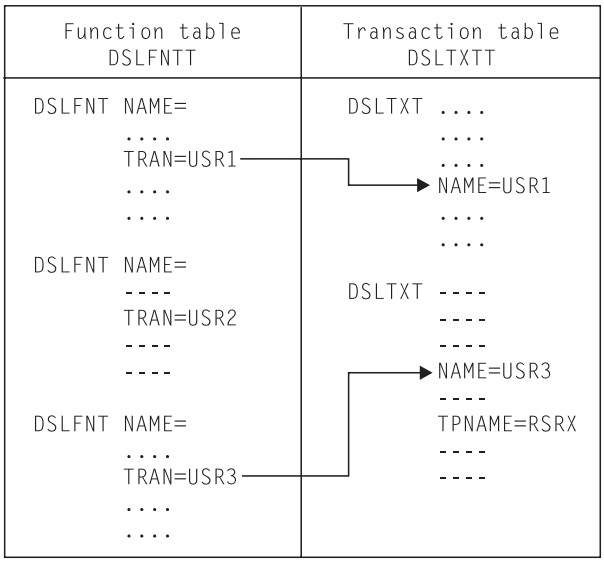

Figure 32. Relationship of Function Table to Transaction Table

DSLTXT Sample Definitions

The following examples show how you can use the MERVA ESA macro, DSLTXT:

DSLTXT NAME=USR1,METHOD=LOCAL
The transaction USR1 is started locally. This is the default and therefore such an entry would be redundant. The transaction runs in the same environment under which the nucleus was started, CICS or IMS BMP. If the nucleus runs as a batch program under MVS, the start of the transaction fails. **Note for IMS:** If the transaction code is not defined to IMS, but the extended terminal option in IMS is used, this error cannot be determined by MERVA ESA, and MERVA ESA inserts a message for this non-existing transaction. IMS creates a queue for this message dynamically where the message (TUCB) is inserted. You can use the IMS command /DIS LTERM USR1 to display the status of this queue.

DSLTXT NAME=USR1,METHOD=LOCAL,TPNAME=USRx,LTERM=PRT7
This is a simple translation of a transaction code. The transaction USR1 in the function table (DSLFNTT) has a reference to the entry in the transaction table (DSLTXTT) with the transaction USRx. The transaction USRx is started according to the entry in the transaction table.

DSLTXT NAME=USR1,METHOD=LOCAL,TPNAME=USR3,LTERM=PRT7,SYSID=CICB,USERID=MASB
This sample shows how to run a transaction on a remote CICS. When running under CICS, the start of the transaction USR1 is performed on the remote system CICB and the transaction is run under the user ID MASB.

DSLTXT NAME=DSLH,METHOD=CICSBATCH,TPNAME=USRH,LTERM=PRT7,SYSID=LUCICB
Start a transaction in the remote CICS with the LU name LUCICB. The transaction USRH is started under the specified CICS system. The EXEC CICS batch facility is used to start the transaction. Thus the MERVA ESA nucleus can run as a batch program under MVS. This is also possible when the MERVA ESA nucleus is running in an IMS BMP.

DSLTXT NAME=DSLH,METHOD=APPCMVS,SYMDEST=SYS1DEST,TPNMAP=USRH,LTERM=PRT7
An APPC connection is used to perform a transaction. The symbolic destination SYS1DEST identifies an LU and a TP name which is started via APPC/MVS. The transaction code USRH is passed in the transaction code field of the TUCB.

DSLTXT NAME=DSLH,METHOD=APPCMVS,LUNAME=FD00IMS,TPNAME=DSLCIMAP,TPNMAP=USRH,LTERM=PRT7
APPC/MVS is used to start a transaction on the system identified by LUNAME FD00IMS, in this case, the base LU of an IMS system. The IMS transaction code is identified by the TPNAME DSLCIMAP. The transaction code USRH is passed in the transaction code field of the TUCB. The MERVA ESA program DSLCIMAP maps the received TUCB prefixed with the transaction code DSLCIMAP to a standard MERVA ESA TUCB and inserts it into the IMS message queue for USRH. You can use this method to start MERVA ESA IMS transactions from a MERVA ESA nucleus that runs as a native batch program, not as a BMP.

Defining Files in MERVA ESA

Every file that you access by requesting general file services must be defined in the file table. The name of the file table can be changed in the module DSLPRM. MERVA ESA provides an example file table with the name DSLFLTT.

MERVA ESA provides the file table macro DSLFLT to code the file table entries, described in the *MERVA for ESA Macro Reference*.

The files that you define in DSLFLTT must also be defined to VSAM, and to CICS file control (CICS installations) or DL/I (IMS installations).

Installing Files for MERVA ESA General File Services

You use files for the MERVA ESA General File Services if you want to use one of the following files:

- A MERVA ESA Nicknames File
- The SWIFT Correspondents File with the SWIFT Link
- The Currency Code File with the SWIFT Link
- The Telex Correspondents File with the Telex Link
- A similar file of your own

To install files for the MERVA ESA general file services:

1. Define the file in the MERVA ESA file table (DSLFLTT).
2. Code a Message Control Block (MCB) that describes the layout of screen panels for the online maintenance and for printing records of the file, and that describes the layout of the records of the file in a TYPE=NET section.
3. Define the fields used in the MCB in the MERVA ESA Field Definition Table (DSLFDTT).
4. Define the MCB in the MERVA ESA Message Type Table (DSLMTTT).
5. Optionally define field expansion in the MERVA ESA Function Table (DSLFNTT).

In CICS, do the following:

1. Define a VSAM KSDS cluster that matches the attributes of the file defined in the MERVA ESA file table (DSLFLTT).
2. Define the file in CICS (FCT or RDO).

In IMS, do the following:

1. Generate an IMS DBD as a DL/I HISAM DB that matches the attributes of the file defined in the MERVA ESA file table (DSLFLTT).
2. Define a VSAM KSDS cluster that matches the IMS DBD.
3. Include DB PCBs in the IMS PSBs of the following MERVA ESA programs, if you want to process the file with these programs:
   - MERVA ESA General File Utility DSLFLUT for initializing and listing the file
   - MERVA ESA End-User Driver Program DSLEUD for online maintenance of the file and field expansion
   - MERVA ESA Checking and Expansion Program DSLCXT for field expansion
   - MERVA ESA Hardcopy Printer Program DSLHCP for field expansion
4. Include a DBD entry in the IMS ACB.
5. Include the necessary DATABASE DBD=... macro in the IMS system generation for each file to be used.
6. If you want to allocate the file dynamically, include appropriate entries in the IMS dynamic allocation member.

The following give examples of how to code the file table structure and file table entries.

Coding the File Table Structure

Figure 33 shows an example of coding the structure of the file table. This structure can also be the result of the MERVA ESA generation process using the DSLGEN macro.

Note: The first macro of the file table must be DSLFLT TYPE=INITIAL, and the last instruction must be DSLFLT TYPE=FINAL.

You can write your own file table copy members to define files that are specific to your installation. You should include them behind the following copy members:

- DSLFLTTC that defines the MERVA ESA Nicknames File
- DWSFLTTC that defines the SWIFT correspondents file and the SWIFT Currency Code File
- ENLFLTTC that defines the Telex correspondents file

```
DSLFLTT  DSLFLT TYPE=INITIAL
         COPY  DSLFLTTC           MERVA PART
         COPY  DWSFLTTC           SWIFT LINK PART
         COPY  ENLFLTTC           TELEX LINK PART
         COPY  XXXFLTTC           YOUR PART XXX
         COPY  YYYFLTTC           YOUR PART YYY

⋮

         DSLFLT TYPE=FINAL
         END
```

Figure 33. The File Table DSLFLTT

Coding File Table Entries

The following figures contain examples of coding file table entries:

- Figure 34 shows the definition of the MERVA ESA nicknames file.
- Figure 35 shows the definition of the Telex Correspondents File.
- Figure 36 shows the definition of the SWIFT Link files, consisting of the SWIFT Correspondents File and the SWIFT Currency Code File.

```
DSLFLT TYPE=DAT,                                        *  [1]
      DAT=DSLCORN,                                      *  [2]
      FLD=DSLCORN,                                      *  [3]
      NAME=DSLCORN,                                     *  [4]
      LENGTH=304,                                       *  [5]
      MSGID=0CORN,                                      *  [6]
      SHARED=YES,                                       *  [7]
      MAINT=YES,                                        *  [8]
FIELDS=(DSLCORN,DSLFLUP,DSLFLCUP,DSLCORID),             *  [9]
      COLS80=(DSLCORN,DSLFLUP,DSLCORID),                * [10]
      COLS132=(DSLCORN,DSLFLUP,DSLCORID),               *
      ROWS24=15,                                        * [11]
      ROWS27=18,                                        *
      ROWS32=23,                                        *
      ROWS43=34,                                        *
      SELECT=(PN,CN),                                   * [12]
      DESCR=('Private Nicknames',                       * [13]
      'Common Nicknames')

DSLFLT TYPE=FLD,                                        * [14]
      FLD=DSLCORN,                                      * [15]
      DAT=DSLCORN,                                      * [16]
      OFFSET=210,                                       * [17]
      LENGTH=32,                                        * [18]
      INFLEN=19,                                        * [19]
      CHECK=ALPHANUM,                                   * [20]
      DESCR='Nickname'                                  * [21]
```

Figure 34. File Table Copy Member DSLFLTTC (Nicknames File)

```
DSLFLT TYPE=DAT,                                        *  [1]
      DAT=ENLCORDA,                                     *  [2]
      FLD=DSLCORID,                                     *  [3]
      NAME=ENLCOR,                                      *  [4]
      LENGTH=504,                                       *  [5]
      MSGID=TCOR,                                       *  [6]
      SHARED=NO,                                        *  [7]
      MAINT=YES,                                        *  [8]
      FIELDS=(DSLCORID,DSLFLUP,DSLFLCUP,                *  [9]
      ENLFLCO1,ENLFLCO2,ENLFLNR1,ENLFLNR2,              *
      ENLFLAB1,ENLFLAB2,                                *
      ENLFLTKY),                                        *
      COLS80=(DSLCORID,DSLFLUP,ENLFLC01),               * [10]
      COLS132=(DSLCORID,DSLFLUP,ENLFLC01),              *
      ROWS24=15,                                        * [11]
      ROWS27=18,                                        *
      ROWS32=23,                                        *
      ROWS43=34,                                        *
      SELECT=TX,                                        * [12]
      DESCR='Telex Correspondents'                        [13]

DSLFLT TYPE=FLD,                                        * [14]
      FLD=DSLCORID,                                     * [15]
      DAT=ENLCORDA,                                     * [16]
      OFFSET=4,                                         * [17]
      LENGTH=24,                                        * [18]
      INFLEN=11,                                        * [19]
      CHECK=BASIC,                                      * [20]
      DESCR='Identifier'                                  [21]
```

Figure 35. File Table Copy Member ENLFLTTC (Telex Correspondents File)

```
DSLFLT TYPE=DAT,                                          *  [1]
      DAT=DWSCORDA,                                       *  [2]
      FLD=DSLCORID,                                       *  [3]
      NAME=DWSCOR,                                        *  [4]
      LENGTH=1738,                                        *  [5]
      MSGID=SCOR,                                         *  [6]
      SHARED=NO,                                          *  [7]
      MAINT=YES,                                          *  [8]
      FIELDS=(DSLCORID,DSLFLUP,DSLFLCUP,DWSCORBK,         *  [9]
      DWSCORAD,DWSCORZP,DWSCORIM,DWSSCORST,DWSCORBE),     *
      COLS80=(DSLCORID,DSLFLUP,DWSCORBK),                 * [10]
      COLS132=(DSLCORID,DSLFLUP,DWSCORBK),                *
      ROWS24=15,                                          * [11]
      ROWS27=18,                                          *
      ROWS32=23,                                          *
      ROWS43=34,                                          *
      SELECT=SW,                                          * [12]
      DESCR='SWIFT Correspondents'                          [13]

DSLFLT TYPE=FLD,                                          * [14]
      FLD=DSLCORID,                                       * [15]
      DAT=DWSCORDA,                                       * [16]
      OFFSET=4,                                           * [17]
      LENGTH=24,                                          * [18]
      INFLEN=11,                                          * [19]
      CHECK=SWIFT,                                        * [20]
      DESCR='Bank Identifier Code'                          [21]

DSLFLT TYPE=DAT,                                          *  [1]
      DAT=DWSCURDA,                                       *  [2]
      FLD=DWSCURID,                                       *  [3]
      NAME=DWSCUR,                                        *  [4]
      LENGTH=1622,                                        *  [5]
      MSGID=SCUR,                                         *  [6]
      SHARED=NO,                                          *  [7]
      MAINT=YES,                                          *  [8]
      FIELDS=(DWSCURID,DSLFLUP,DSLFLCUP,DWSCURNM,DWSCURF, *  [9]
      DWSCURIN,DWSCURIM,DWSCURST),                        *
      COLS80=(DWSCURID,DWSCURNM),                         * [10]
      COLS132=(DWSCURID,DWSCURNM),                        *
      ROWS24=15,                                          * [11]
      ROWS27=18,                                          *
      ROWS32=23,                                          *
      ROWS43=34,                                          *
      SELECT=CUR,                                         * [12]
      DESCR='Currency Code File'                            [13]

DSLFLT TYPE=FLD,                                          * [14]
      FLD=DWSCURID,                                       * [15]
      DAT=DWSCURDA,                                       * [16]
      OFFSET=4,                                           * [17]
      LENGTH=3,                                           * [18]
      CHECK=ALPHA,                                        * [20]
      DESCR='Currency Identifier Code'                    * [21]
```

Figure 36. File Table Copy Member DWSFLTTC (SWIFT Link files)

Notes:

[1] Entry related to the total data of the file.

[2] The MERVA ESA file name. This name refers to the file in DSLFLVP requests. In IMS, it is also the root segment name.

[3] The search field name. This name refers to the field in DSLTOFSV requests. In IMS, it is also the sequence field name.

[4] In IMS it is the DBD name of the related DL/I HISAM DB. In CICS it is the DD/DLBL name of the related VSAM KSDS cluster.

[5] The record length. In IMS, it is the root segment length. In CICS, it is the record length defined to VSAM.

[6] The message ID of the file MCB. This MCB is used for mapping record data to screen/printer devices.

[7] The MERVA ESA Nicknames File is shared: it contains common and private data. The SWIFT Correspondents File is not shared.

[8] The file is available in the online file maintenance. For the MERVA ESA Nicknames File, both common and private data is available.

[9] The TOF field names of the record fields:

DSLCORID Identifier (in the SWIFT Link, it is the bank identifier code)

DSLCORN Nickname

DSLFLCUP Time of creation

DSLFLUP Time of last update

DWSCORAD Correspondent's address

DWSCORBK Correspondent's name

DWSCORIM Time of first import from BIC update tape

DWSCORST Time of last import from BIC update tape

DWSCORBE Unedited data from BIC update tape

DWSCORZP Zip code

DWSCURID Currency code identifier

DWSCURNM Currency name

DWSCURF Fractional digits

DWSCURIN Currency (country) information

DWSCURIM Time of creation

DWSCURST Time of last update

ENLFLCO1 Correspondent's address line 1

ENLFLCO2 Correspondent's address line 2

ENLFLNR1 First telex number

ENLFLNR2 Second telex number

ENLFLTKY Test-key requirements

ENLFLAB1 Long answerback 1

ENLFLAB2 Long answerback 2

[10] Online file maintenance, the fields of the file to be displayed in a list panel, depending on the number of columns on the screen (see the COLS80 and COLS132 parameters).

[11] Online file maintenance, the number of records to be displayed in a list panel depending on the number of lines of the screen (ROWS24, ROWS27, ROWS32, and ROWS43).

[12] Online file maintenance, the file-selection identifier. Nonshared files have only one file-selection identifier. Shared files with MAINT=YES have two file-selection identifiers for:

- Private data
- Common data

[13] Online file maintenance, the file description on the file-selection menu. Nonshared files have only one description. Shared files with MAINT=YES have two descriptions for:

- Private data
- Common data

[14] Entry related to the search field of the file.

[15] The search field name.

[16] The MERVA ESA file name.

[17] The offset of the search field. In CICS, it is the key offset defined to VSAM. In IMS, it is the sequence field offset from the beginning of the root segment.

[18] The length of the search field. It is the key length defined to VSAM. In IMS, it is also the sequence field length.

[19] The length of the information contained in the search field. For the SWIFT Correspondents File, only characters 1-11 contain information. For the MERVA ESA Nicknames File, only characters 1-19 contain information. The rest of the search field must be padded with blanks.

Note: The search field characters 1-8 (the owner prefix) for shared files contain a user ID (showing private ownership) or '* ' (showing common ownership). This part of the search field is not displayed on the panels of the online file maintenance.

[20] The type of data the search field must contain besides the trailing blanks, and if applicable, besides the owner prefix.

The MERVA ESA Nicknames File must have alphanumeric data in the search field.

The SWIFT Correspondents File must have a BIC (SWIFT address) in the search field.

[21] Online file maintenance. The descriptive name of the search field that is displayed on the screen terminal. The search field is also used in error messages displayed on the terminal.

Defining the Page Sizes and Layouts

A MERVA ESA user screen, system, or hardcopy print page is divided into the following areas:

- The top frame
- The message area
- The bottom frame

The layout for these areas is defined in the appropriate device descriptions for the frame MCBs and the message type MCB.

The frame MCBs are specified in the MERVA ESA Function table definition (FRAME= parameter). The top and bottom frames contain the information for display on all the pages of a message. For example, it can contain:

- The logo of the financial institution
- The date and time
- The user identification

The message area contains data pages of the message currently being processed, or a function panel.

The default frame MCBs DSL0TOP and DSL0BOT provided by MERVA ESA, define the top and bottom frame. The frame definition can be given for screen, hardcopy, and system printer devices. The ID parameter selects the language specific frame for the screen functions. For print functions, the print format specification in the MERVA ESA Function table entry is used to select the appropriate printer frame device definition. The DSLLCOND macros can be specified, to select a top or bottom frame according to the TOF contents. The default nesting identifier for the frame is always 0. This nesting identifier should be specified explicitly if fields belonging to the message are to be displayed. Blank line compression does not take place for frame panels. Frame fields must be specified in sequence, the last frame DSLLDFLD specification determines the size of the frame.

The page size definition for screen terminals is given in the CICS terminal definitions, or in the MERVA ESA terminal feature definition table (DSLTFDT) for IMS. For screen terminals one physical screen page is formatted at a time.

The page size definition for hardcopy printer terminals is given in the MERVA ESA terminal feature definition table (DSLTFDT) both for CICS and IMS. The DSLTFDT definitions are connected to a logical terminal name as defined in CICS or a terminal macro of the IMS nucleus generation. The values are used to determine the printer's buffer size, and the page size that is to be used for formatting. For hardcopy printer terminals, one logical page is formatted at a time, but the page is printed in one or more segments according to the printer's buffer size. The page size in the DSLTFDT is defined in rows and columns as follows:

- **Rows** is a "lines" definition for one logical page. It is interpreted by MERVA ESA so that a physical page (your paper forms from one perforation to the next) can be divided into several logical pages, with each logical page consisting of as many lines as there are "rows" defined. To obtain a separate logical page per physical page, rows must be exactly the number of lines your paper form can hold. For correct formatting, make sure that your printer is positioned on the first line of your paper form when starting to print.
- **Columns** is a physical line width definition and must match the physical characteristics of the printer terminal. If your printer terminal can print 132 characters per line, the number of columns specified must also be 132 characters per line. Specifying a column of 80 characters would result in printing two lines in one. In turn, specifying 132 as the column parameter when your printer can only print 80 columns, results in the characters beyond column 80 being printed on the next line.

The page size definition for the system printer is given in the MERVA ESA terminal feature definition table (DSLTFDT) for both CICS and IMS. When no terminal feature definition table is available, or the table does not contain the definition for the logical terminal name DSLSDSY (same as the symbolic name of the DD statement for batch print output) the following default values are used:

- The default page length is 55.
- The default line length is 132. The table is used by the MERVA ESA batch printing program DSLSDY. For the system printer, one physical line is formatted at a time.

Terminal Feature Definition Macro (DSLTFD)

The MERVA ESA terminal feature definition table (DSLTFDT) is used to define the physical characteristics (for example, page sizes) of display and print devices such as hardcopy printer terminals and, for MERVA ESA operating under IMS, screen terminals. (For CICS, the page size definition for screen terminals is given in the terminal definitions, not the DSLTFDT.) The macro DSLTFD is used to generate a terminal feature definition table.

The page size definitions for hardcopy printer terminals and the system printer are given in DSLTFDT in both CICS and IMS. In CICS, the hardcopy printer definitions in the terminal definition and in the DSLTFDT must match.

MERVA ESA supplies a sample DSLTFDT that contains definitions for all supported terminal types. You can use the samples to define your terminals. Each terminal used for MERVA ESA in your installation must be defined in DSLTFDT (except the screen terminals in CICS), and is identified in DSLTFDT by the logical terminal name used by CICS or IMS.

You can either specify a name for you terminal feature definition table (this name must then be specified in the TFD parameter of the DSLPARM macro in DSLPRM), or use the default name DSLTFDT. For more information about DSLPRM, see the *MERVA for ESA Macro Reference*.

The terminal feature definition table consists of a sequence of DSLTFD macros. The first instruction should contain a label. If no label is specified in the first instruction, the default label DSLTFDT is used. The last statement of the table should be an END statement.

No assembler language statements except TITLE, SPACE, EJECT, END, and PRINT can be used in the table definition.

If errors are detected in a specific DSLTFD macro, appropriate MNOTEs are provided.

The BUFSIZE parameter specifies the physical terminal buffer size. It is used to calculate the addresses of the fields within the terminal buffer. Although it does not have any connection with the PAGESIZ parameter of DSLTFDT, it still must comply with the definition of VTAM®, IMS, and CICS (see Table 7).

Table 7. Terminal Feature Definition Table

	Terminal Buffer Size	**NCP/VTAM Rusizes=**	**CICS Definition Buffer=**	**IMS Terminal Outbuf=**	**DSLTFDT Bufsize=**
Video Terminal	1920	default	default	2420*	1920
Non-SCS Printer	2400			2400	2400
SCS Printer	1920	X'87C6' (768)	768	768	768

* For video terminals in IMS, the size of the OUTBUF for the terminal macro in the IMS nucleus generation must be 500 greater than the BUFSIZE.

Printer Terminal Page Sizes and SCS Printer Support

Both the 3277-2 and 3270 screen terminal models with larger sizes, extended highlighting, and extended color are supported by MERVA ESA. 3287/3289 printers attached to a 3274 as LUTYPE-1, using the SCS data stream can be used with MERVA ESA operating under both CICS and IMS. The following subsections show examples for using the different printers under CICS and IMS.

Note: When using SCS printers, MERVA ESA uses the number of lines of a page and the line length for setting up the vertical and horizontal forms (SVF, SHF), that is, those features must be available for the device.

MERVA ESA generates one set vertical format (SVF) control for each print segment. Some printers have difficulties to process such data streams correctly; either they allow only one SVF control per page or the number of vertical formats within one SVF is limited. It is possible to suppress the use of SVF controls in the generated SCS data streams by specifying the parameter TERMTYP=SCSPSIM in the DSLTFD macro. Instead of SVF controls the appropriate number of new line (NL) controls is generated in the data stream.

MERVA ESA never uses a forms-feed control character (FF) to skip to the next page. This is because the last line on a page is part of the bottom frame and the field corresponding to this bottom frame must be printed there.

Printer Definitions in CICS

The terminal characteristics are determined by the CICS terminal definitions, but the values for buffer size and page/line size as defined in the DSLTFDT are used by MERVA ESA.

Note: When defining buffer and page/line sizes, consider the corresponding features of the devices used. You must not define values greater than those of the devices otherwise unpredictable results can be obtained as the device data stream created by MERVA ESA is determined by the size definitions.

Definition of Printer Features in CICS: The printer page/line size are always derived from the definitions in the terminal feature definition table DSLTFDT.

- **Example 1** defines any 3270 printer (3284/3286/3287/3288/3289). DEFSCRN is omitted, and the default value of 24x80 is used.

```
DFHTCT TYPE=TERMINAL,TRMIDNT=V89A,ACCMETH=VTAM,
      TRMTYPE=3270P,TRMMODL=2,
      TRMSTAT=TRANSCEIVE,
      FEATURE=PRINT,TCTUAL=16,
      NETNAME=U72FD.
```

  The following definition statements for the CICS/ESA utility DFHCSDUP give an example of how to define a 3270 printer:

```
DEFINE TYPETERM(3270P000) GROUP(DSLGROUP)
       DEVICE(3270P) TERMMODEL(2) SHIPPABLE(NO) PAGESIZE(24,80)
       ALTPAGE(0,0) FMHPARM(NO) OBOPERID(NO) AUTOPAGE(YES)
       DEFSCREEN(24,80) ALTSCREEN(24,80) APLKYBD(NO) APLTEXT(NO)
       AUDIBLEALARM(NO) COLOR(NO) COPY(NO) DUALCASEKYBD(NO)
       EXTENDEDDS(NO) HILIGHT(NO) KATAKANA(NO) LIGHTPEN(NO)
       MSRCONTROL(NO) OBFORMAT(NO) PARTITIONS(NO) PRINTADAPTER(NO)
       PROGSYMBOLS(NO) VALIDATION(NO) FORMFEED(NO) HORIZFORM(NO)
       VERTICALFORM(NO) TEXTKYBD(NO) TEXTPRINT(NO) QUERY(NO)
       OUTLINE(NO) SOSI(NO) BACKTRANS(NO) CGCSGID(0,0) ASCII(NO)
       SENDSIZE(0) RECEIVESIZE(256) BRACKET(YES) LOGMODE(0)
       ERRLASTLINE(NO) ERRINTENSIFY(NO) ERRCOLOR(NO) ERRHILIGHT(NO)
       AUTOCONNECT(NO) ATI(YES) TTI(YES) CREASESESS(YES) RELREQ(NO)
```

```
       DISCREQ(NO) NEPCLASS(0) SIGNOFF(YES) XRFSIGNOFF(NOFORCE)
       ROUTEDMSGS(ALL) LOGONMSG(NO) BUILDCHAIN(NO) USERAREALEN(16)
       IOAREALEN(0,0) UCTRAN(NO) RECOVOPTION(SYSDEFAULT)
       RECOVNOTIFY(NONE)
DEFINE TERMINAL(V89A) GROUP(DSLGROUP)
       AUTINSTMODEL(NO) TYPETERM(3270P000) NETNAME(U72FD)
       CONSOLE(NO) PRINTERCOPY(NO) ALTPRINTCOPY(NO) TASKLIMIT(NO)
       TERMPRIORITY(0) INSERVICE(YES) ATTACHSEC(LOCAL) BINDSECURITY(NO)
```

- **Example 2** defines a VTAM LUTYPE-1 printer using the SCS data stream. A printer page of 66 lines is defined with a line length of 120.

  Note: If this page/line size is required the appropriate definition must be supplied in the DSLTFDT.

```
DFHTCT TYPE=TERMINAL,TRMIDNT=SCSX,ACCMETH=VTAM,
      LOGMODE=0,
      TRMTYPE=SCSPRT,TRMMODL=2,TIOAL=(2048,4096),
      TRMSTAT=TRANSCEIVE,RELREQ=(YES,YES),
      TCTUAL=16,HF=YES,VF=YES,CHNASSY=YES,
      RUSIZE=256,BUFFER=768,DEFSCRN=(66,120),
      NETNAME=ALUB106
```

  LOGMODE=0 shows that an MODETABLE entry has been defined for the LU during NCP/VTAM system generation, for example:

```
T3287SCS MODEENT LOGMODE=T3287SCS,FMPROF=X'03',TSPROF=X'03',
            PRIPROT=X'B1',SECPROT=X'90',COMPROT=X'3080',
            RUSIZES=X'87C6',PSERVIC=X'01000000E100000000E10000',
            PSNDPAC=X'01',SRCVPAC=X'01'
```

  The following definition statements for the CICS/ESA utility DFHCSDUP give an example of how to define an SCS printer:

```
DEFINE TYPETERM(3790000) GROUP(DSLGROUP)
       DEVICE(SCSPRINT) TERMMODEL(2) SHIPPABLE(NO) PAGESIZE(24,80)
       ALTPAGE(0,0) FMHPARM(NO) OBOPERID(NO) AUTOPAGE(YES)
       DEFSCREEN(0,0) ALTSCREEN(0,0) APLKYBD(NO) APLTEXT(NO)
       AUDIBLEALARM(NO) COLOR(NO) COPY(NO) DUALCASEKYBD(NO)
       EXTENDEDDS(NO) HILIGHT(NO) KATAKANA(NO) LIGHTPEN(NO)
       MSRCONTROL(NO) OBFORMAT(NO) PARTITIONS(NO) PRINTADAPTER(NO)
       PROGSYMBOLS(NO) VALIDATION(NO) FORMFEED(NO) HORIZFORM(NO)
       VERTICALFORM(YES) TEXTKYBD(NO) TEXTPRINT(NO) QUERY(NO)
       OUTLINE(NO) SOSI(NO) BACKTRANS(NO) CGCSGID(0,0) ASCII(NO)
       SENDSIZE(768) RECEIVESIZE(256) BRACKET(YES) LOGMODE(0)
       ERRLASTLINE(NO) ERRINTENSIFY(NO) ERRCOLOR(NO) ERRHILIGHT(NO)
       AUTOCONNECT(NO) ATI(YES) TTI(YES) CREASESESS(YES) RELREQ(NO)
       DISCREQ(NO) NEPCLASS(0) SIGNOFF(YES) XRFSIGNOFF(NOFORCE)
       ROUTEDMSGS(ALL) LOGONMSG(NO) BUILDCHAIN(YES) USERAREALEN(0)
       IOAREALEN(2048,4096) UCTRAN(NO) RECOVOPTION(SYSDEFAULT)
       RECOVNOTIFY(NONE)
DEFINE TERMINAL(SCSX) GROUP(DSLGROUP)
       AUTINSTMODEL(NO) TYPETERM(3790000) NETNAME(ALUB106) CONSOLE(NO)
       PRINTERCOPY(NO) ALTPRINTCOPY(NO) TASKLIMIT(NO) TERMPRIORITY(0)
       INSERVICE(YES) ATTACHSEC(LOCAL) BINDSECURITY(NO)
```

- **Example 3** defines a 3270 printer controlled as VTAM LUTYPE-3 using the data stream compatibility mode (DSC). A page size of 55x80 is defined.

```
DFHTCT TYPE=TERMINAL,TRMIDNT=SLU3,ACCMETH=VTAM,
      TRMTYPE=LUTYPE3,TRMMODL=2,CHNASSY=YES,
      RELREQ=(YES,YES),BRACKET=YES,
      TCTUAL=16,RUSIZE=256,BUFFER=1536,
      FEATURE=(COLOR,HILIGHT),
      DEFSCRN=(55,80)
      NETNAME=ALUB104
```

Screen and Printer Definitions in MERVA ESA

DSLTFDT, Table Structure: For IMS installations the features of the printers and screen terminals are supplied in the Terminal Feature Definition Table DSLTFDT.

Sample Table Entries for DSLTFDT:

```
TFDT     TITLE 'MERVA ESA TERMINAL FEATURE DEFINITION TABLE'
DSLTFDT  DSLTFD TYPE=INITIAL                                          [1]
DEFSCRN  DSLTFD LTERM=DEFSCRN,PAGESIZ=(24,80),BUFSIZE=1920,TERMTYP=3270  [2]
         DSLTFD LTERM=L782,PAGESIZ=(24,80),TERMTYP=3270               [3]
         DSLTFD LTERM=L783,PAGESIZ=(32,80),TERMTYP=3270,WRC=WRCEWA
         DSLTFD LTERM=L784,PAGESIZ=(43,80),TERMTYP=3270,WRC=WRCEWA
         DSLTFD LTERM=L785,PAGESIZ=(27,132),TERMTYP=3270,WRC=WRCEWA
         DSLTFD LTERM=L792,PAGESIZ=(24,80),TERMTYP=3270,              *
               FEATURE=(COLOR,EXTHIL)
         DSLTFD LTERM=L793,PAGESIZ=(32,80),TERMTYP=3270,WRC=WRCEWA,   *
               FEATURE=(COLOR,EXTHIL)
DEFPRINT DSLTFD LTERM=DEFPRINT,PAGESIZ=(24,80),BUFSIZE=1920,          * [4]
               TERMTYP=3270P
PRTSCS   DSLTFD LTERM=I4S33F06,PAGESIZ=(72,132),BUFSIZE=768,          * [5]
               TERMTYP=SCSP,FEATURE=(EXTHIL,COLOR)
PRTTRM   DSLTFD LTERM=I4ZMEDB,PAGESIZ=(72,132),BUFSIZE=2400,          * [6]
               TERMTYP=3270P
L86A     DSLTFD LTERM=L86A,PAGESIZ=(24,80),BUFSIZE=1920,              *
               TERMTYP=3270P
L89A     DSLTFD LTERM=L89A,PAGESIZ=(55,132),BUFSIZE=1920,             *
               TERMTYP=3270P,FEATURE=(COLOR)
         DSLTFD LTERM=DSLSDSY,PAGESIZ=(72,132),TERMTYP=SYSP             [7]
         DSLTFD TYPE=FINAL                                              [8]
         END
```

Figure 37. MERVA ESA Terminal Feature Definition Table

Notes:

[1] TYPE=INITIAL

This must be the first DSLTFD macro. It assigns the label DSLTFDT as the name of the terminal feature definition table.

[2] DSLTFD LTERM=DEFSCRN

This is an entry for the default screen terminal and must not be removed. The logical terminal name is DEFSCRN for the standard 3270 screen model used in an installation with 24 rows and 80 columns (PAGESIZ=(24,80)). The ERASE WRITE command (default) is used.

Note: If a screen logical terminal name is not found in this table the specification of the default screen is applied.

[3] DSLTFD LTERM=L782

This and the following DSLTFD macros assign logical terminal names L782 to L793 to the following screen models:

- 3278-2
- 3278-3
- 3278-4
- 3278-5
- 3279-2B
- 3279-3B

These definitions are used only for MERVA ESA running under IMS.

[4] DSLTFD LTERM=DEFPRINT

This is an entry for the default printer and must not be removed but should be changed according to the characteristics of the printer mostly used in your installation. The terminal type is 3270P for a printer, page and buffer sizes are as the default values (24 lines and 80 columns). The number of columns specified should correspond to the maximum size of the printer page for correct line feed.

[5] DSLTFD LTERM=I4S33F06

This is an entry for an LU-1 SCS printer with all the features available (color and extended highlighting attributes). The terminal type is SCSP, a BUFSIZE of 768 bytes, according to the NCP/VTAM RUSIZE, and 72 lines are used.

[6] DSLTFD LTERM=I4ZMEDB

This is an entry for a 3286 printer (3270 data stream). The terminal type is 3270P, a BUFSIZE of 2400 bytes and a physical page of 72 lines and 132 columns are used. Two other definitions for the same type of printer, but with different page sizes, follow in the example.

[7] DSLTFD LTERM=DSLSDSY

This is an entry for the definition of the system printer (TERMTYP=SYSP).

[8] TYPE=FINAL

This must be the last DSLTFD macro and is followed by the assembler END statement.

Customizing the MERVA Message Processing Client Server

The server for MERVA Message Processing Client workstations supports connections using:

- APPC mapped conversations without synchronization
- TCP/IP

When MERVA ESA is not running under CICS, the server is implemented in two parts:

- A listener task (the controlling task)
- A conversation subtask

Under CICS, there is only the conversation subtask, and CICS supplies the listener services.

CICS APPC Connections

Because the subtask DSLAFM01 is already defined as transaction DSLF in the CICS definition provided by MERVA ESA, no special customization is necessary for CICS installations.

CICS initiates the transaction upon receiving an allocate request from a client for a conversation with the CICS LU and the TP DSLF.

Each workstation that is to be served by DSLAFM01 requires a SESSION and CONNECTION definition in CICS:

```
DEFINE CONNECTION(conn)
       ACCESSMETHOD(VTAM) PROTOCOL(APPC)
       NETNAME(client lu)

⋮

DEFINE SESSIONS(session)
       CONNECTION(conn) PROTOCOL(APPC)

⋮
```

Refer to *CICS/ESA Version 4.1 Intercommunication Guide* for information on how to define connections and sessions.

IMS APPC Connections

For IMS installations an APPC/MVS Server program, DSLAFA01, is provided to initiate the DSLAFM01 subtask. DSLAFA01 is controlled by the WSASRV parameter in the MERVA ESA parameters module DSLPRM.

DSLAFA01 is a batch program which registers itself with APPC/MVS as the server for any allocation requests for conversations with the LU and TP defined in the WSASRV parameter of DSLPRM. See "DSLPRM Module Sample" on page 87. Refer to the APPC/MVS documentation for more information on APPC/MVS Servers, and how to control use of APPC/MVS transaction programs.

TCP/IP Connections

For all environments, the TCP/IP version of the server program is implemented as a batch program. The listener program DSLAFATM registers with TCP/IP and waits for connection requests from workstations, and the subtask DSLAFMTM is initiated to manage each conversation. The only customization possible is specifying the port number used by DSLAFATM, which is specified in the WSTSRV parameter of DSLPRM.

MERVA ESA User File

All MERVA Message Processing Client workstation users must be defined in the MERVA ESA User File before they can sign on to MERVA ESA. How to update the User File is discussed in *MERVA for ESA User's Guide*.

Chapter 3. The SWIFT Link

This chapter explains how to customize the following areas in the SWIFT Link for the SWIFT network:

- Define the SWIFT Link customizing parameters in the module DWSPRM. This explains how to use the DWSPARM macro.
- Define the data communication lines to SWIFT in the line definition modules. This explains how to use the DWSVLINE macro. The DWSVLINE macro is used to define the connection to the MERVA Extended Connectivity product that runs on a 37xx controller and connects to the SWIFT network via X.25.
- Define the logical terminals for connecting to the SWIFT network in the Logical Terminal Table (DWSLTT).
- Customize the SWIFT Link to exploit parallel processing of nucleus servers. Multiple instances of the SWIFT Link server can run under a single MERVA ESA nucleus.
- Update the currency code table DWSCURT, which is used for checking and the currency code help panel. Alternatively, the SWIFT currency code file can be used to store the currency codes in MERVA ESA. The currency code file can be maintained online using the file maintenance function.
- Define the central institutions table (DWSCIT) if you want to use the PREMIUM or FIN-Copy service offered by SWIFT.

The SWIFT Link customizing parameters in the DWSPRM module, the line definition modules, the SWIFT Link logical terminal table, and the central institutions table are separate load modules that are loaded by the SWIFT Link programs when needed.

Defining SWIFT Link Parameters in Module DWSPRM

For basic customizing of the SWIFT Link environment the module DWSPRM is used. This module uses the DWSPARM macro, described in the *MERVA for ESA Macro Reference*. Figure 38 shows an example of the SWIFT Link customizing parameters.

```
DWSPRM   DWSPARM LINENAM=DWSLIN,                                *  [1]
                 CHECK=(NO,NO),                                 *  [2]
                 AUTHUP=MYUAUTPW,                               *  [3]
                 AUTHAGE=200,                                   *  [4]
                 CIT=DWSCIT,                                    *  [5]
                 LSKQUE=SLSLOAD,                                *  [6]
                 LTTQUE=SLSLTT,                                 *  [7]
                 INTACK=600,                                    *  [8]
                 INTRES=120,                                    *  [9]
                 LOG2=DWSLOG2,                                  * [10]
                 RSKQUE=SLSRECV,                                * [11]
                 SWIN=10000,                                    * [12]
                 SWOUT=11000,                                   * [13]
                 FORMAT=TOF                                       [14]
         END
```

Figure 38. Example of SWIFT Link Customizing Parameters for the SWIFT Network

Notes:

[1] LINENAM=DWSLIN

LINENAM specifies the first six characters of the line definition module names, the seventh character, and optionally the eighth character, are the number of the line. Line numbers from 1 through 30 are supported. DWSLIN is the default. If you use different names, you must supply the first six characters here.

[2] CHECK=(NO,NO)

This parameter shows that both messages received from the SWIFT network are not checked (first subparameter ″NO″), and messages sent to the SWIFT network are not checked for formal syntax (second subparameter ″NO″). This improves the performance when communicating with the SWIFT network. If SWIFT output messages are not checked, they should be routed to a function calling the MERVA ESA expansion and checking transaction DSLCXT (see function L3CXT). If SWIFT input messages are not checked, they should be checked before routing them to the ready queues for sending, to avoid negative acknowledgments from SWIFT.

[3] AUTHUP=MYUAUTPW

MYUAUTPW is specified as the password for scrambling the authenticator keys before they are unloaded to a sequential file. You supply the password used with this parameter. You should use this parameter to protect the unload file from unauthorized access.

[4] AUTHAGE=200

A number of 200 authenticator key entries is specified for the aging table used by the authenticator key file support program DWSAUTP. Because of this specification 200 keys instead of 100 (default) are stored in main storage after an authenticator key was requested. This reduces the number of accesses to the file during authentication.

You should tune your SWIFT Link specifications to reduce file accesses and to improve the performance of your system. The diagnostic message DWS771I issued during the termination of the DWSAUTP program gives statistics about the number of keys retrieved from the aging table and from the key file.

[5] CIT=DWSCIT

Specify the name of the central institutions table if you are using the PREMIUM or FIN-Copy service of SWIFT.

[6] LSKQUE=SLSLOAD

When SWIFT secure login/select (SLS) is used with session key pre-generation and preload, this parameter specifies the queue where the SWIFT Link Load Session Keys program DWSDLSK expects the session keys received from the workstation that processes the SWIFT USE functions (USE workstation). From this queue, the session keys are moved to the session key queues defined with the SKEYQ parameters in DWSLTT.

[7] LTTQUE=SLSLOAD

This parameter specifies the queue where the SWIFT Link saves information from DWSLTT between a termination and the next startup. This information includes login and select sequence numbers (LSN and SSN), input and output sequence numbers (ISN and OSN) and information related to SLS.

[8] INTACK=600

This specifies a timeout period for ISN ACKs. If an ACK does not arrive within 600 seconds (10 minutes), the session is aborted.

[9] INTRES=120

This specifies a time interval in seconds during which the SWIFT Link does **not** try a resumption after a suspension was complete on a switched line to the SWIFT network, even if there is something ready for sending. This interval is ignored if a MERVA ESA operator enters a command that generates a message for sending to the SWIFT network, for example, a **login**, **logout**, **select** or **quit** command.

[10] LOG2=DWSLOG2

LOG2 defines the name of the user exit that retrieves the session keys for login and select when paper tables are used. In one MERVA ESA installation, some logical terminals can use paper tables for login and select, others can use SLS. This parameter defines the sample DWSLOG2 that contains the session keys used for test and training. You can modify DWSLOG2 for a live logical terminal, or use a different module name, and you can either include the session keys in the table or retrieve them from a data base.

[11] RSKQUE=SLSRECV

When SWIFT secure login/select (SLS) is used with single session key retrieval from the USE workstation, this parameter specifies the queue where the SWIFT Link expects the session key response messages. From this queue, the session keys are moved to the pertinent entry in DWSLTT, and login or select can proceed.

[12] SWIN=10000

This parameter defines the buffer length used to format outgoing SWIFT messages. It is the maximum length of a message that can be sent by SWIFT Link. For some message types S.W.I.F.T. allows a message length of up to 10000 bytes. The maximum length of individual message types is determined by the specification in the message type table. A message that is larger than the specification in the message type table is not sent to SWIFT.

[13] SWOUT=11000

This parameter defines the buffer length used to receive incoming SWIFT messages. It is the maximum length of a message that can be received by SWIFT Link. For some message types S.W.I.F.T. allows a message length of up to 10000 bytes for input messages. When transmitted as an output message by SWIFT some additional header and trailer information may have been added. A value of 11000 is large enough to accommodate all possible cases.

[14] FORMAT=TOF

This parameter defines that messages are stored in the standard MERVA ESA tokenized format in the queue data set. Message checking can only be performed on messages in the tokenized format. If you specify FORMAT=ELF, messages are stored in external line format. Messages in external line format are not tokenized, but the message buffer is stored in a single field in the TOF. Message checking is not performed on the external line format. As formatting and checking operations are responsible for

most of the consumed CPU cycles, the use of the external line format can reduce the CPU load and increase the throughput of the SWIFT Link.

The use of the FORMAT=ELF parameter might require changes to routing modules. Because the SWIFT fields (SWxxxxxx) are no longer available for direct access, they cannot be used in routing modules. MERVA ESA provides field names to access the SWIFT header and trailer fields in external line format. These field names (EFxxxxxx) are handled by an MFS separation routine which extracts the required field data from the external line format.

If you use FORMAT=ELF, applications which create or process SWIFT messages must be changed to provide the messages in external line format.

Defining Communication Lines to the SWIFT Network

Here are examples of how to define the following types of line to the SWIFT network:

- Leased lines for SWIFT X.25
- Switched lines for automatic dialing for SWIFT X.25

The connection to the SWIFT network via X.25 requires the product MERVA Extended Connectivity running under NCP on a 37xx controller. The connection between MERVA ESA SWIFT Link and MERVA Extended Connectivity is a VTAM LU Type 1 session. Refer to the *MERVA Extended Connectivity Installation and User's Guide* for more details about customizing MERVA Extended Connectivity.

Select the line definitions that are most appropriate to your bank's needs. You must supply the specifications according to the examples described in the following. A maximum of 30 lines for X.25 connections can be selected, with leased lines, switched lines, or a combination of them. You can define more line modules than actual lines exist in your installation. This is useful when one physical line is used in different operation modes.

If you use line definition module names other than DWSLINx, you must define these module names to CICS.

The line definition modules for X.25 use the DWSVLINE macro. The macro is described in the *MERVA for ESA Macro Reference*.

Line Definition for a Public Data Network Line for SWIFT X.25

Figure 39 shows how to define a leased line to the SWIFT network via X.25 using a public data network in the line definition module DWSLIN7.

```
DWSLIN7  DWSVLINE LINETYP=LEASED,                              * [1]
               LNSAPNM=VNDOBET299000,                          * [2]
               RDTEADR=031347037729800,                        * [3]
               CUD=0101,                                       * [4]
               PLUNAME=FD0AC388,                               * [5]
               SLUNAME=X78VU42                                   [6]
         END
```

Figure 39. Example of a SWIFT Link Line Definition Module (X.25 Leased Line)

Notes:

[1] LINETYP=LEASED

This parameter defines a leased line to the SWIFT network.

[2] LNSAPNM=VNDOBET299000

This parameter specifies the symbolic address (local NSAP name) of the financial institution assigned by S.W.I.F.T.

[3] RDTEADR=031347037729800

This parameter specifies the remote DTE address. This represents the address an X.25 connection is established to, the called DTE address. You can specify a list of remote DTE addresses here. If a list is specified, the remote DTE addresses are tried sequentially until the network connection is established. The value is provided by S.W.I.F.T. In this case, it refers to an ACCUNET address in the USA. Refer to the *S.W.I.F.T. User Handbook* for more details about the DTE address field.

[4] CUD=0101

This parameter specifies the variable part of the call user data. Two octets are defined in this case. This value to be specified is provided by S.W.I.F.T.

[5] PLUNAME=FD0AC388

This parameter defines the primary logical unit name; it is the name MERVA ESA uses to open an ACB to VTAM.

[6] SLUNAME=X78VU42

This parameter defines the secondary logical unit name; it is the name used by MERVA Extended Connectivity running on the communication controller under NCP. This name is logically connected to an SVC on a specific physical line to the X.25 network. This line must be defined as a leased line to SWIFT.

Line Definition for a Leased Line for SWIFT X.25

Figure 40 shows how to define a leased line to the SWIFT network via X.25 in the line definition module DWSLIN8.

```
DWSLIN8  DWSVLINE LINETYP=LEASED,                                  * [1]
               LNSAPNM=VNDOBET299000,                              * [2]
               LDTEADR=99,                                         * [3]
               LOGMODE=SCS3790,                                    * [4]
               PLUNAME=ID0AC385,                                   * [5]
               SLUNAME=F39VU12,                                    * [6]
               TRACE=(N,N,N,N,N,Y,N,N,N,N)                           [7]
         END
```

Figure 40. Example of a SWIFT Link Line Definition Module (X.25 Leased Line)

Notes:

[1] LINETYP=LEASED

This parameter defines a leased line to the SWIFT network.

[2] LNSAPNM=VNDOBET299000

This parameter specifies the symbolic address (local NSAP name) of the financial institution assigned by S.W.I.F.T.

[3] LDTEADR=99

This parameter specifies the local DTE address. For leased lines, S.W.I.F.T. allows the user to use subaddressing of up to two digits. Refer to the S.W.I.F.T. User Handbook for more details about the DTE address field.

[4] LOGMODE=SCS3790

This parameter specifies the LOGMODE table entry used by MERVA ESA. The bind of the session between MERVA ESA and MERVA Extended Connectivity is done with the characteristics defined in the LOGMODE table entry SCS3790. If this parameter is specified, the RRUSIZE and SRUSIZE parameters are ignored.

[5] PLUNAME=ID0AC385

This parameter defines the primary logical unit name; it is the name MERVA ESA uses to open an ACB to VTAM.

[6] SLUNAME=F39VU12

This parameter defines the secondary logical unit name; it is the name used by MERVA Extended Connectivity running on the communication controller under NCP. This name is logically connected to an SVC on a specific physical line to the X.25 network. This line must be defined as a leased line to SWIFT.

[7] TRACE=(N,N,N,N,N,Y,N,N,N,N)

This parameter defines the initial setting of the trace switches for this line. In this case the sixth trace switch is set on, resulting in a trace of all data buffers transferred between MERVA ESA and MERVA Extended Connectivity. The trace output is written to the SYSPRINT data set (MVS) or to SYSLST (VSE).

Use the trace facility only on request of the IBM service and support organization. Do not use this facility in a production environment for performance reasons.

Line Definition for an Auto Dial Line for SWIFT X.25

Figure 41 shows how to define a switched line to the SWIFT network via X.25 for automatic dialing in the line definition module DWSLIN9.

```
DWSLIN9  DWSVLINE LINETYP=SWITCHED,                                  * [1]
               LPHONE=9497031166282,                                 * [2]
               LNSAPNM=VNDOBET299000,                                * [3]
               PHONE=((0015559876,D),(0015559877,S)),                * [4]
               PLUNAME=ID0AC381,                                     * [5]
               RRUSIZE=2048,                                         * [6]
               SLUNAME=F39VU11,                                      * [7]
               SRUSIZE=8192                                            [8]
         END
```

Figure 41. Example of a SWIFT Link Line Definition Module (X.25 Auto Dial Line)

Notes:

[1] LINETYP=SWITCHED

This parameter defines that line 9 is a PSTN line to the SWIFT network.

[2] LPHONE=9497031166282

This parameter defines the local telephone number used for return calls from SWIFT. This number is used when a resumption is initiated by

SWIFT. When this number is not specified resumption initiated by SWIFT is not possible. The number must start with a 9 followed by the country code, area code, and local phone number.

[3] LNSAPNM=VNDOBET299000

This parameter specifies the symbolic address (local NSAP name) of the financial institution assigned by S.W.I.F.T.

[4] PHONE=((0015559876,D),(0015559877,S))

This parameter defines a list of telephone numbers as agreed with S.W.I.F.T. to dial the SWIFT network. If S.W.I.F.T. gives you more than one telephone number, you can define up to 9 telephone numbers here. If a line is busy, MERVA ESA automatically tries the next telephone number in the list. In the example, the subparameter D indicates that the telephone number is a dedicated port at SWIFT side; the subparameter S indicates that the telephone number is a shared port. S.W.I.F.T. informs you whether the assigned telephone number is for a dedicated or a shared port.

[5] PLUNAME=ID0AC381

This parameter defines the primary logical unit name; it is the name MERVA ESA uses to open an ACB to VTAM.

[6] RRUSIZE=2048

This parameter defines the request unit size for receiving. Because chaining is used to transfer data between MERVA ESA and MERVA Extended Connectivity this value does not define an actual limit on data sizes. You can tune this value in case of performance problems.

[7] SLUNAME=F39VU11

This parameter defines the secondary logical unit name; it is the name used by MERVA Extended Connectivity running on the communication controller under NCP. This name is logically connected to an SVC on a specific physical line to the X.25 network. This line must be defined as a switched line.

[8] SRUSIZE=8192

This parameter defines the request unit size for sending.

VTAM Definition for SWIFT X.25 Lines

Figure 42 shows how to define an application ID for a connection to the SWIFT network via X.25. The application ID is used to open a connection to the MERVA Extended Connectivity product running on a 37xx controller.

```
   VBUILD TYPE=APPL                                                    [1]
ID0AC381 APPL  AUTH=ACQ                                                [2]
```

Figure 42. Example of a VTAM Definition for MERVA Extended Connectivity

Notes:

[1] VBUILD TYPE=APPL

This statement instructs VTAM to build an application.

[2] ID0AC381 APPL AUTH=ACQ

This statement specifies that the application name ID0AC381 is defined.

Figure 43 shows how to define a LOGMODE entry to be used for the VTAM session to the MERVA Extended Connectivity product running on a 37xx controller. The LOGMODE entry is used to define the session characteristics of the connection when the parameter LOGMODE is specified in the DWSVLINE macro. Specify a LOGMODE entry only when you want to select a class-of-service (COS) for the session.

```
SCS3790  MODEENT LOGMODE=SCS3790,FMPROF=X'03',TSPROF=X'03',           * [1]
               PRIPROT=X'B1',SECPROT=X'B0',COMPROT=X'3080',           *
               RUSIZES=X'8585',PSERVIC=X'010000000000000000000000'
```

Figure 43. Example of a LOGMODE Entry for MERVA Extended Connectivity

Notes:

[1] MODEENT LOGMODE=SCS3790

This is the name of the LOGMODE entry, and must be specified in the LOGMODE parameter of the DWSVLINE macro. The parameter RUSIZES specifies the maximum RU sizes used by the secondary and primary logical unit:

- The leftmost 2 digits specify the value for the secondary logical unit and correspond to the RRUSIZE parameter of the DWSVLINE macro.
- The rightmost 2 digits specify the value for the primary logical unit and correspond to the SRUSIZE parameter of the DWSVLINE macro.

The following table shows the allowed values for the leftmost and rightmost 2 digits of the RUSIZES parameter, and the corresponding RU sizes:

RU Size	**64**	**128**	**256**	**512**	**1024**	**2048**	**4096**	**8192**
RUSIZES left 2 digits (RRUSIZE)			85	86	87	88		
RUSIZES right 2 digits (SRUSIZE)	83	84	85	86	87	88	89	8A

Note: Do not change any of the other parameters of the LOGMODE entry! For more information about the DWSVLINE macro, refer to the *MERVA for ESA Macro Reference*. For more information about RU sizes, refer to the *VTAM Resource Definition Reference*.

Defining Logical Terminals for the SWIFT Network

The SWIFT Link Logical Terminal Table defines the logical terminals used in a computer based terminal (CBT) as agreed with S.W.I.F.T. These definitions are used by the SWIFT Link Program DWSDGPA which is the main program of the SWIFT Link for the SWIFT network.

DWSDGPA loads the logical terminal table with the name defined in DWSPRM. The name of the Logical Terminal Table can also be specified as the start parameter in the NPT entry definition, or in the operator start command. If multiple instances of the SWIFT Link are used, each instance needs its own Logical Terminal Table. If you use a name other than DWSLTT, you must define this to CICS. The routing tables defined in the logical terminal table are also loaded and must be defined to CICS.

The DWSLT macro can be used by user-written programs to map the entries of DWSLTT. User-written programs link-edited to DSLNUC can access DWSLTT via

DSLCOM. The field COMDSNL points to an address list that contains the address of DWSPRM in the third fullword when the SWIFT Link (that is, the program DWSDGPA) is started. User-written programs not link-edited to DSLNUC must load DWSLTT if they need it.

Figure 44 shows how to define the master and synonym logical terminals, ready queues, routing tables, and other information. These definitions are used in the examples for the routing logic starting with "Examples of Function Table Entries for the SWIFT Link" on page 7.

Details of the parameters of the DWSLT macro are given in the *MERVA for ESA Macro Reference*.

Note: Names used in the function tables, routing tables, and logical terminal table must be consistent.

```
LTT      TITLE 'EXAMPLE FOR SWIFT LOGICAL TERMINAL TABLE'
DWSLTT   DWSLT TYPE=MAS,NAME=VNDEBET2A,LINE=6,                      * [1]
               ROUTIN=DWSL1IN,ROUTOUT=DWSL1OUT,                     *
               READYQ=(L1RGPAU),                                    *
               TFLAG=SLS,                                           * [2]
               USENAME=USEMERVA2,                                   *
               ROUTSK=DWSRTSK,                                      *
               SKEYQ=SLSGPA
         DWSLT TYPE=AP,NAME=FIN,                                    * [3]
               ROUTIN=DWSL1IN,ROUTOUT=DWSL1OUT,                     *
               READYQ=(L1RFINU,L1RFINN),                            *
               SKEYQ=SLSFIN,                                        *
               DELSUB=(SYSTEM,URGENT,NORMAL)
         DWSLT TYPE=MAS,NAME=VNDOBET2A,LINE=7,                      * [4]
               ROUTIN=DWSL2IN,ROUTOUT=DWSL2OUT,                     *
               READYQ=(L2RGPAU)
         DWSLT TYPE=SYN,NAME=VNDOSYN2A                                [5]
         DWSLT TYPE=AP,NAME=FIN,                                    *
               ROUTIN=DWSL2IN,ROUTOUT=DWSL2OUT,                     *
               READYQ=(L2RFINU,L2RFINN),                            *
               DELSUB=(SYSTEM,URGENT,NORMAL)
         DWSLT TYPE=MAS,NAME=VNDPBET2A,LINE=8,                      * [6]
               ROUTIN=DWSL3GPI,ROUTOUT=DWSL3GPO,                    *
               READYQ=(L3RGPAU)
         DWSLT TYPE=SYN,NAME=VNDPSY12A
         DWSLT TYPE=SYN,NAME=VNDPSY22A
         DWSLT TYPE=AP,NAME=FIN,                                    *
               ROUTIN=DWSL3FII,ROUTOUT=DWSL3FIO,                    *
               READYQ=(L3RFINU,L3RFINN),                            *
               DELSUB=(SYSTEM,URGENT,NORMAL)
         END
```

Figure 44. Coding Example of a SWIFT Link Logical Terminal Table

Notes:

[1] This is the first DWSLT macro, the label DWSLTT defines the name of the logical terminal table. This name must be specified in the LTT parameter of the DWSPARM macro used in the SWIFT Link customization parameter module DWSPRM.

The logical terminal VNDEBET2A is a master logical terminal (TYPE=MAS) with the routing tables DWSL1IN for input messages from a ready queue, and DWSL1OUT for output messages received from SWIFT.

The READYQ=(L1RGPAU) parameter defines the ready queue name for the General Purpose Application messages. The queue L1RGPAU must be defined in the function table DSLFNTT.

Line 6 (LINE=6) is used for this master logical terminal. The line used can be changed by the SWIFT Link operator using the **setlt** command.

[2] The master logical terminal VNDEBET2A uses the SWIFT secure login/select (SLS) technology as defined with the TFLAG=SLS parameter. As a consequence, either one or both of the following must be defined also:

- For single session key retrieval from the workstation where the SWIFT USE functions run (USE workstation), the name of the USE workstation used for routing of the session key request to the MERVA Link send queue for sending to this USE workstation (USENAME parameter). Also, the name of a routing table used for this routing must be specified then (ROUTSK parameter).

  Using single session key retrieval requires the USE workstation being connected to MERVA ESA during login and select.
- For session key pre-generation and preloading, the name of the queue that is to contain the session keys (SKEYQ parameter). The session keys are generated at the USE workstation, sent to MERVA ESA via MERVA Link, and stored in the session key queue by the program DWSDLSK.

  You can define a different session key queue for each entry of DWSLTT, this eases the supervision and maintenance of the session keys (see the *MERVA for ESA Operations Guide* for details).

  Using preloaded session keys requires the USE workstation being connected to MERVA ESA only during the transmission of the session keys from the USE workstation to MERVA ESA. It is not necessary to have the USE workstation connected during login and select then.

If both ways are provided, the SWIFT Link tries to get a session key from the queue first, and if it is not found there, a session key request is sent to the USE workstation.

The TFLAG and USENAME values can be changed using the **setlt** command (see *MERVA for ESA Operations Guide* for details).

[3] This entry defines the financial application for the first master logical terminal (TYPE=AP,NAME=FIN).

The routing tables DWSL1IN for financial input messages from a ready queue and DWSL1OUT for financial output messages received from SWIFT are used.

The READYQ=(L1RFINU,L1RFINN) parameter defines the ready queue names for the Financial Application messages:

- L1RFINU for urgent priority and system messages
- L1RFINN for normal priority messages

The ready queues are processed by DWSDGPA in the order in which they appear in the READYQ parameter. Both queues must be defined in the function table DSLFNTT.

SKEYQ=SLSFIN defines the session key queue for the FIN application (see the explanation of this parameter for the master logical terminal). The

parameters TFLAG, USENAME and ROUTSK cannot be specified for the FIN application as they cannot be different from the values specified for the master logical terminal.

DELSUB=(SYSTEM,URGENT,NORMAL) defines the default FIN delivery subsets used in the select message sent to the SWIFT network. These delivery subsets can be changed with the **dds** or **select** command (see the *MERVA for ESA Operations Guide* for details). The FIN delivery subsets must be created with a General Purpose Application message type 047, as otherwise the select message is rejected by the SWIFT network.

[4] The logical terminal VNDOBET2A is a master logical terminal (TYPE=MAS parameter) with the routing tables DWSL2IN for input messages from a ready queue and DWSL2OUT for output messages received from the SWIFT network. The ready queue name L2RGPAU is defined. The LINE=7 parameter specifies that this master logical terminal uses line 7.

This master logical terminal and its FIN application use paper table technology for login and select as the TFLAG parameter is not specified and therefore defaulted to TFLAG=PT.

[5] The logical terminal VNDOSYN2A is a synonym logical terminal (TYPE=SYN) for the master logical terminal VNDOBET2A.

The synonym logical terminal uses the same facilities as the master logical terminal and its FIN application, for example, the same ready queues and routing tables.

[6] A third master logical terminal VNDPBET2A and its two synonym logical terminals VNDPSY12A and VNDPSY22A are defined with the macros that follow.

The LINE=8 parameter defines that this master logical terminal and its synonyms use line 8.

SWIFT Link Parallel Processing

You can split MERVA ESA installations and the corresponding message volume to achieve more throughput. The disadvantage of this method is that the messages and the infrastructure must be split also and it is complicated to integrate these messages afterwards in an offline process.

The SWIFT Link parallel processing allows the activation of multiple SWIFT Link servers in the same MERVA ESA nucleus. This way, the journal and queue data set are not split over multiple systems, but are integrated within one file, data set, or database.

However, the logical terminals and the logical X.25 lines must be distributed between the SWIFT Link servers. One physical X.25 line can transport more than one logical line (SVC) at the same time, therefore this is not a real limitation. On the other hand, when only a single logical terminal with a high throughput requirement is logged in, multiple SWIFT Link servers are not useful.

The parallel processing performance option should be considered only for large MVS installations with a high SWIFT message volume. If this option is chosen, MERVA ESA nucleus should not run as a CICS transaction but as an MVS batch program or as an IMS BMP. In CICS installations, the MERVA ESA nucleus can run in a separate batch region. Transactions in the CICS region are started by the CICS batch interface via the MERVA ESA transaction table.

Before multiple servers of the SWIFT Link are activated, a detailed trade-off analysis should be performed. The overall throughput of a system fulfilling the prerequisites can be expected to be higher than with a single SWIFT Link server. On the other hand, the logical terminals must be distributed over different logical terminal tables (DWSLTT). The operating and controlling of multiple SWIFT Links within one MERVA ESA system is more complicated. The SWIFT Link command router has to be used for operator commands to address the correct SWIFT Link server.

The operation can be done from a single point of control, a CMD, or an MSC function. This is probably simpler than with multiple and split MERVA ESA systems where the command operations have to be done from different sessions.

The following sample tables show a scenario using the base SWIFT Link server and three additional servers running in parallel. The servers must be defined in the nucleus program table.

```
⋮

* For DWSDGPA up to 152 ECBs can be specified, supporting 30 lines.
* DWSDGPA needs 5 ECBs for each line plus 2 ECBs for the program.
* A minimum of 12 ECBs must be specified to support two active lines.
         DSLNPT TYPE=PGM,NAME=DWSDGPA,ECB=152,STRTREQ=4,STOPREQ=8,     *
               ECBREQ=0,AUTO=YES,PRTY=8,DESC=SWIFTII
         DSLNPT TYPE=PGM,NAME=DWSDGPA,ECB=12,STRTREQ=4,STOPREQ=8,      *
               DESC=SWIFTIIA,PARM=DWSLTTA
         DSLNPT TYPE=PGM,NAME=DWSDGPA,ECB=12,STRTREQ=4,STOPREQ=8,      *
               DESC=SWIFTIIB,PARM=DWSLTTB
         DSLNPT TYPE=PGM,NAME=DWSDGPA,ECB=12,STRTREQ=4,STOPREQ=8,      *
               DESC=SWIFTIIC,PARM=DWSLTTC

⋮
```

Figure 45. Example of DWSNPTTC Definitions for Multiple SWIFT Link Servers

The first additional server has the name SWIFTIIA. It must be started manually by an operator. This is done by entering the command:

```
S SWIFTIIA
```

This server requires its own logical terminal table DWSLTTA. The logical terminal names defined in this table must not be specified in any of the other logical terminal tables. Each server must also be defined as a subtask in the nucleus server table DSLNSVT (SERVER=TASK).

```
⋮

* The SWIFT link program group
 DSLNSV NAME=SWIFTII,MODNAME=DWSDGPA,SERVER=TASK
         DSLNSV NAME=SWIFTAUT,MODNAME=DWSAUTIN
         DSLNSV NAME=DWSDCMD
         DSLNSV NAME=DWSDCMR
         DSLNSV NAME=DWSDIVA
         DSLNSV NAME=DWSXTRCE
         DSLNSV NAME=DWSAUTP
         DSLNSV NAME=DWSAUTIN
 DSLNSV NAME=SWIFTIIA,MODNAME=DWSDGPA,SERVER=TASK
         DSLNSV NAME=DWSDCMDA,MODNAME=DWSDCMD
 DSLNSV NAME=SWIFTIIB,MODNAME=DWSDGPA,SERVER=TASK
         DSLNSV NAME=DWSDCMDB,MODNAME=DWSDCMD
 DSLNSV NAME=SWIFTIIC,MODNAME=DWSDGPA,SERVER=TASK
         DSLNSV NAME=DWSDCMDC,MODNAME=DWSDCMD

⋮
```

Figure 46. Example of DSLNSVT Definitions for Multiple SWIFT Link Servers

All commands directed to this SWIFT Link server must be prefixed with the command **SWIFTTII**. For this command, the synonym **SW** is defined; refer to *MERVA for ESA Operations Guide* for more details. For example, to open a line, issue a login to SWIFT, and select the FIN application for an LT, the following command sequence must be entered:

```
SW A,SETLT VNDEBET2A,9
SW A,LI VNDEBET2A
SW A,SE VNDEBET2A
```

Session Keys Received from the USE Workstation

If several SWIFT Link servers are used and session keys are received from the USE workstation individually and on demand (that is, during login or select), each SWIFT Link server needs it own single session key receive queue. The name of the queue is defined with the DWSPRM parameter RSKQUE. The default name of the queue is SLSRECV.

A SWIFT Link server identified with a descriptive name of the form **SWIFTII*x*** (where ***x*** is its letter identifier) creates the name of the receive queue for the single session keys by appending its letter identifier to the name specified for the RSKQUE parameter. These queues must be defined in the DSLFNTT, for example:

```
 *        RECEIVED SINGLE SWIFT SESSION KEYS (SWIFTIIA)
          DSLFNT NAME=SLSRECVA,                                  +
                DESCR='Received SWIFT Session Keys (SWIFTIIA)',  +
                DQFILL=NO,                                       +
                KEY1=(SWBHLT,12),                                +
                NEXT=USEERROR,                                   +
                QUEUE=YES
*
```

Furthermore, the routing modules that refer to the queue must be updated to route the messages containing the single session keys to the appropriate queues. In the session-key routing module DWSRTCT or EKARTTXU, the statement for the target queue for received single session queues must be modified to support multiple receive queues. The original statement looks like this:

```
*        ROUTE SINGLE SESSION KEYS RECEIVED FROM THE USE PS/2
RECV     DSLROUTE TYPE=SET,TARGET=('SLSRECV'),GOTO=END
```

It must be changed as described here:

- First alternative: Route the session key to all possible receive queues or SWIFT Link servers. Unsuitable session keys are ignored by any other SWIFT Link servers not handling the LT. For example:

```
RECV     DSLROUTE TYPE=SET,TARGET=('SLSRECV')
         DSLROUTE TYPE=SET,TARGET=('SLSRECVA')
         DSLROUTE TYPE=SET,TARGET=('SLSRECVB')
         DSLROUTE TYPE=SET,TARGET=('SLSRECVC'),GOTO=END
```

- Second alternative: Route the session keys beginning with certain characters to the proper queue, depending on the name in the LT field. All others go into a standard SLSRECV queue. For example:

```
RECV    DSLROUTE TYPE=TEST,COND=(LT,'IBMA',EQ,SHORT),TRUE=RECVA
        DSLROUTE TYPE=TEST,COND=(LT,'IBMB',EQ,SHORT),TRUE=RECVB
        DSLROUTE TYPE=TEST,COND=(LT,'IBMC',EQ,SHORT),TRUE=RECVC
        DSLROUTE TYPE=SET,TARGET=('SLSRECV'),GOTO=END
RECVA   DSLROUTE TYPE=SET,TARGET=('SLSRECVA'),GOTO=END
RECVB   DSLROUTE TYPE=SET,TARGET=('SLSRECVB'),GOTO=END
RECVC   DSLROUTE TYPE=SET,TARGET=('SLSRECVC'),GOTO=END
```

The Currency Codes

There are two methods in MERVA ESA to store the currency codes:

- The optional SWIFT link currency code file
- The SWIFT link currency code table

When a currency code file is installed, the currency codes can be updated using the MERVA ESA file maintenance function.

The SWIFT Link currency code table defines all currency codes used in SWIFT messages. The table is supplied as an assembler copy code DWSCURT that is used by the program DWSMCCRT and the help panel MCB DWSHCUR. The copy code can be changed when currency codes change. DWSMCCRT and DWSHCUR must be assembled, and DWSMU141 and DWSHCUR must be link-edited.

The currency codes are defined using the DWSCUR macro. For more information about the parameters of the DWSCUR macro, refer to the *MERVA for ESA Macro Reference*.

Figure 47 shows examples of how to code a currency code.

```
DWSCUR  CUR=ADP,NUM=0,COM='Andorran Peseta'                  [1]

DWSCUR  CUR=CHF,NUM=2,COM=('Swiss Franc',                  * [2]
       'Liechtenstein',                                    *
       'Switzerland')
```

Figure 47. Example of two Entries in the Currency Code Table

Notes:

[1] The first example shows the currency code ADP in the CUR parameter. This currency allows the specification of no digits after the decimal comma (NUM=0), and the descriptor for the help panel shows that this is the "Andorran Peseta" (COM parameter).

[2] The second example shows the currency code CHF in the CUR parameter. This currency allows the specification of two digits after the decimal

comma (NUM=2), and the descriptor for the help panel shows that this is the "Swiss Franc" (COM parameter). Two additional subparameters of the COM parameter (there can be more additional subparameters) show that this currency is used in Liechtenstein and Switzerland.

The Central Institutions Table

Use the DWSCI macro to define the central institutions participating in the PREMIUM or FIN-Copy services offered by SWIFT The macro parameters are described in detail in the *MERVA for ESA Macro Reference*. The name of the DWSCIT table has to be specified in DWSPRM using the CIT parameter of the DWSPARM macro.

PREMIUM Service

Figure 48 shows how to define a central institution for a SWIFT PREMIUM service with a single macro statement.

```
DWSCIT   DWSCI   CI=COPYCCLL,                                  * [1]
                 FIELD=SW53,                                  * [2]
                 HOME=VNDOBET2,                               * [3]
                 MSG=(S100,S202)                                [4]
```

Figure 48. Example of a Central Institution of a SWIFT PREMIUM Service

Notes:

[1] CI=COPYCCLL

The name of the central institution.

[2] FIELD=SW53

The name of the message field containing the central institution.

[3] HOME=VNDOBET2

The name of the home LT.

[4] MSG=(S100,S202)

A list of message identifications that are part of the PREMIUM set for this central institution.

If the home LT, the name of the central institution in the specified field, and the message type of a SWIFT input message match the DWSCI PREMIUM service definitions, a PAC trailer is generated.

To calculate the PAC you have to exchange authentication keys with the central institution. The Authenticator-Key File has to contain a record for the specified home LT and CI.

FIN-Copy Service

Figure 49 on page 142 shows the definitions for a SWIFT FIN-Copy service. The DWSCIT table entry for each service consists of two parts:

1. The service characteristics are defined with the DWSCI TYPE=DEF macro statement.
2. Each SWIFT message type that is included in the service has to be defined with a DWSCI TYPE=MSG macro statement.

The TYPE=MSG definitions must follow the TYPE=DEF statement immediately. The DWSCIT table can contain more than one FIN-Copy service.

```
DWSCIT   DWSCI   TYPE=DEF,FINCOPY=YCP,                          *  [1]
               CI=COPYLLCC,                                    *  [2]
               HOME=VNDOBET2,                                  *  [3]
               CURR=IEP,                                       *  [4]
               AUTH=2,                                         *  [5]
               FULLCOPY=N                                         [6]
         DWSCI   TYPE=MSG,                                     *  [7]
               MT=S100,                                        *  [8]
               FIELDS=(20,32A),                                *  [9]
               CURRFLD=SW32CUR                                   [10]
```

Figure 49. Example of a SWIFT FIN-Copy Service Definition

Notes:

[1] TYPE=DEF,FINCOPY=YCP

The name of the FIN-Copy service is defined by SWIFT. The message field 103 must contain this 3-character service code.

[2] CI=COPYLLCC

The name of the central institution.

[3] HOME=VNDOBET2

The name of the home LT.

[4] CURR=IEP

Only messages with this currency code are included in the service.

[5] AUTH=2

The service uses double authentication (MAC and PAC calculation).

[6] FULLCOPY=N

Not the whole message but only a partial copy will be sent to the central institution for this service.

[7] TYPE=MSG

The definition of the FIN-Copy Service must be immediately followed by at least one SWIFT message type specification.

[8] MT=S100

The SWIFT message type that is to be included in the FIN-Copy service. It must be defined in the message type table (DSLMTTT).

[9] FIELDS=(20,32A)

This parameter gives a list of up to 30 SWIFT field tags, including the option character. These are the fields that will be copied to the central institution, as FULLCOPY=N is specified for this service.

[10] CURRFLD=SW32CUR

The message field SW32CUR contains the currency code. The name of this field is required as the parameter CURR=IEP has been specified for the FIN-Copy service.

CURRFLD must be defined in the field definition table (DSLFDTT). If the currency code is only a part of the field contents, the subfield name must be used. If the field occurs more than once in the message, the first occurrence will be used to select the message for FIN-Copy.

A SWIFT input message is included in the FIN-Copy service, if the home LT, the service code in field 103, the message type and (optionally) the currency code match the DWSCI FIN-Copy service definitions.

To calculate the PAC you have to exchange authentication keys with the central institution. The authenticator-key has to contain a record for the specified home LT and CI.

If you want to add field 103 to the messages or to perform extra validations required by the central institution, you can route the outgoing FIN-Copy messages to a special queue where you have installed your own transaction to perform these functions. You can use a MERVA ESA routing module to select only messages that are potentially for FIN-Copy.

Chapter 4. Setting Up a Central Institution to Calculate PACs

If your bank is a central institution that uses double authentication when processing FIN-Copy service messages of type MT096 (incoming) or MT097 (outgoing), it authenticates the imbedded text block and the message authorization code (MAC) of the original message with a proprietary authorization code (PAC).

When the central institution receives an MT096 message, regardless of whether it uses partial copy or full-copy service, the FIN-Copy program groups all the imbedded fields of that message into a single field named SF97COPY, and stores them in the MERVA ESA tokenized format (TOF) buffer. An application program can use TOF services to retrieve the contents of this field, or can scan the imbedded text block. When scanning, the application program starts with the tag of the first copied field, not with the begin-of-block tag ({**4:**), and ends with the final hyphen. In either case, the application program must store the contents of SF97COPY while the MT096 message is being processed, and must insert them into the MT097 message.

MT096 PAC Calculation

The PAC is generated by the sender of the original financial message on the fields copied to the central institution and the original MAC. The buffer to authenticate with the PAC is located in the MT096 message by the FIN-Copy program DWSFCPY. It consists of the imbedded text block (this is not the first occurrence of a text block as in other SWIFT messages) and of the MAC in the imbedded trailer.

Authenticator keys have been exchanged between the sender of the original message and the central institution. They are stored in the authenticator-key file and can be retrieved using a home LT and a correspondent LT. The sender of the original message can be located in the imbedded basic header. That is, the correspondent LT in the authenticator-key file. For MT096 messages, the receiving LT in the basic header is not the home LT, as it is in other SWIFT message types. Therefore two LTs have to be defined in the DWSCIT table:

HOME=servccll
: FIN-Copy server destination. This is the receiving LT in the basic header of the MT096 and of the sender of the MT097.

CI=copyccll
: Central institution destination. This LT has exchanged authenticator keys with the sender and the receiver of the original financial message.

Both LTs may be identical, but usually they are different.

MT097 PAC Calculation

Authenticator keys have been exchanged between the central institution and the receiver of the original financial message. They are stored in the authenticator key file. The PAC has to be calculated on the fields, and the MAC has to be copied into the MT096 message and, if present, into field 115. The copied fields and the MAC are not sent back by the central institution in the MT097, but they must be added to the message, otherwise the PAC cannot be calculated by the authentication program (DWSAUTP). The LT of the receiver, which has to be used as correspondent LT in the authenticator-key file, is also not contained in the MT097.

Therefore the application of the central institution must insert the following fields into the MT097 message when using the external line format U:

SF97COLT Receiver of the original message in the MRF trailer of the MT096.

SF97MAC MAC of the original message.

SF97COPY When the central institution receives the MT096 message, it puts all the imbedded fields of the original message together into this field and stores it in tokenized format (TOF). An application program can then use TOF services to retrieve the contents of this field.

The application program can scan the imbedded text block using other means as well. It can start with the tag of the first copied field (not with the beginning of block tag 4), and end with the final hyphen, and store this into SF97COPY. This has to be done during processing the MT096 message, and this data has to be inserted into the MT097 message as well.

These fields must be defined in the MCB DWSF097, as shown here:

```
DWSF097  DSLLMCB COPR=2000
*************************************************************************
* This MCB is used for  APPL  APDU-ID  MT                               *
*                        F      01     097                              *
*************************************************************************
MESSAGE  DSLLDEV  TYPE=MESSAGE
         DSLLCOND O1=(TEST=DSLCOND),EQ=NO,O2='NO',GOTO=MSG10
         DSLLEXIT IMBED=SBHEAD
         DSLLEXIT IMBED=SAHEAD
MSG10    DSLLCOND ,
GRN005   DSLLGRP  GRPNUM=5
SW103    DSLLMFLD MAND=YES
SW109    DSLLMFLD MAND=YES
SW451    DSLLMFLD MAND=YES
SW432    DSLLMFLD
SW114    DSLLMFLD
SW115    DSLLMFLD
SF97COLT DSLLMFLD DAMAX=1,LENGTH=(8,,F)
SF97MAC  DSLLMFLD DAMAX=1,LENGTH=(8,,F)
SF97COPY DSLLMFLD DAMAX=1,LENGTH=U
COPY     DWSMTRL
.
.
.
LINEX    DSLLDEV  TYPE=NET,ID=X,SEP=X'0D25',LIKE=LINEY
LINES    DSLLDEV  TYPE=NET,ID=S,SEP='}',LIKE=LINEW
*
*        LINE U to insert the fields of MT096 into MT097:
*        SF97COLT,  SF97MAC , SF97COPY
*
LINEU    DSLLDEV  TYPE=NET,ID=U,SEP='}'
         DSLLCOND O1=(TEST=DSLCOND),EQ=NO,O2='NO',GOTO=NETU10
         DSLLEXIT IMBED=SBHEAD
         DSLLEXIT IMBED=SAHEAD
         DSLLNFLD TAG='{4:',SEP=''
NETU10   DSLLCOND ,
         DSLLGRP  GROUP=GRN005
         DSLLNFLD FLD=SW103,TAG='{103:'
         DSLLNFLD FLD=SW109,TAG='{109:'
         DSLLNFLD FLD=SW451,TAG='{451:'
         DSLLNFLD FLD=SW432,TAG='{432:'
         DSLLNFLD FLD=SW114,TAG='{114:'
         DSLLNFLD FLD=SW115,TAG='{115:'
         DSLLNFLD TAG='}',SEP=''
         DSLLEXIT IMBED=STRAIL,TAG='{5:',SEP='}'
```

```
         DSLLNFLD FLD=SF97COLT,TAG='{COLT:'
         DSLLNFLD FLD=SF97MAC,TAG='{MAC:'
         DSLLNFLD FLD=SF97COPY,TAG='{COPY:'

GEN      DSLLGEN
END
```

Figure 50 shows the definitions for the central institution in the DWSCIT table. The other parameters of the DWSCI MACRO, for example CURR and CURRFLD, can be omitted, as they are not used for MT096 or MT097 messages.

```
DWSCIT   DWSCI  TYPE=DEF,FINCOPY=service,       * [1]
                HOME=servccll,                  * [2]
                CI=copyccll,                    * [3]
                AUTH=2,                         * [4]
                FULLCOPY=Y,                     * [5]
                SERVER=Y                          [6]
         DWSCI  TYPE=MSG,                       * [7]
                MT=S096,                        * [8]
         DWSCI  TYPE=MSG,                       * [7]
                MT=S097,                        * [9]
```

Figure 50. Example of a Central Institution That Uses SWIFT FIN-Copy Service

Notes:

[1] TYPE=DEF,FINCOPY=service

The name of the FIN-Copy service. This is the receiving LT in the application header of the MT096 and the sending LT in the basic header of the MT097.

[2] HOME=servccll

FIN-Copy server destination. This LT receives the MT096 and sends the MT097.

[3] CI=copyccll

The LT of the central institution which has exchanged authentication keys with the sender and receiver of the financial messages. This may be the same LT as the FIN-Copy server destination.

[4] AUTH=2

Double authentication is used. The MT096/MT097 messages contain PAC trailers.

[5] FULLCOPY=Y

The MT096 contains a full copy of the financial message.

[6] SERVER=Y

This parameter indicates a FIN-Copy server entry. The LT of the central institution is used as home LT in the authenticator-key file to calculate the PAC in the MT096/MT097.

[7] TYPE=MSG

Each S.W.I.F.T. message type that is included in the FIN-Copy service must be defined with a DWSCI TYPE=MSG macro statement.

[8] MT=S096

This parameter tells the authentication program that a PAC has to be calculated for MT096 messages.

[9] MT=S097

This parameter tells the authentication program that a PAC has to be calculated for MT097 messages.

Chapter 5. The Telex Link

In MERVA ESA there are two ways to communicate with the public telex network:

- The Telex Link via workstation
- The Telex Link via a fault-tolerant system using the Telex Interface Program

The Telex Link via workstation is described in the book *MERVA Workstation Based Functions*.

This chapter describes the customizing that must be carried out to adapt the Telex Link via a fault-tolerant system to meet the requirements of your installation and organization.

Customizing Parameters ENLPRM

The ENLPARM macro is used to generate the Telex Link customizing parameter module ENLPRM. You must modify ENLPRM according to your environment and requirements, then assemble and link-edit it before starting the Telex Link.

Details of the parameters of the ENLPARM macro are given in the *MERVA for ESA Macro Reference*.

Note: Names used in the function tables, routing tables and Telex Link customizing parameter module must be consistent.

```
ENLPARM                                                 *
      AUTO=NO,                                          * [1]
      BUFSIZE=6000,                                     * [2]
      LINES=120,                                        * [3]
      NRMQ=TXNRM,                                       * [4]
      URGQ=TXURG,                                       *
      RTRCV=ENLRTHCF,                                   * [5]
      RTSND=ENLRTHCF,                                   *
      TESTKEY=ETS6000                                     [6]
END
```

Figure 51. Coding Example of a Telex Link Customizing Parameter Module

Notes:

[1] AUTO=NO specifies that the session with the S/1 TXIP is not to be signed on automatically when the Telex Link (the program ENLSTP) is started. The Telex Link operator must start the connection manually with the **txon** command.

[2] BUFSIZE=6000 specifies the size of the buffer used for the communication with TXIP, that is, the maximum number of characters that a telex message can have.

The carriage-return and line-feed characters after each line are included in this number, but not the shift characters (alpha to numeric and numeric to alpha).

The maximum number is 32767. You must not specify a size greater than TXIP can handle.

If you use a buffer size (BUFSIZE) that is greater than 6000, you must also increase the values for the NICBUF, JRNBUF, and TOFSIZE parameters in the MERVA ESA customizing parameter module DSLPRM:

- Set NICBUF to a value at least 2000 bytes larger than the value of BUFSIZE.
- Increase the values of JRNBUF and TOFSIZE by the same amount you increased NICBUF.
- When working with a dynamic TOF, set MAXBUF to a value 2000 times the size of BUFSIZE.

If the record size of the journal data set is not large enough to allow the whole buffer to be stored in one record, you must segment the journal records. This is described in "DSLPRM Module Sample" on page 87.

[3] LINES=120 specifies the maximum number of text lines that an outgoing telex message can have. If you change this number you must change the DACNT parameter in the MCB for the message type TELEX, the message type for outgoing telex messages. The name of the MCB is ENLMTSND.

[4] NRMQ=TXNRM specifies the 1- to 8-character name of the queue that contains the telex messages that are ready for transmission to TXIP and have the telex type N (normal), T (timed), or P (print only). Telex messages contained in this queue are sent to TXIP only when the queue for urgent telex messages (defined with the URGQ parameter) is empty. The ready queues of the Telex Link must be specified in the function table DSLFNTT.

[5] RTRCV specifies the 1 to 8 character name of the MERVA ESA routing table used for routing telex messages and invalid data received from TXIP. RTSND specifies the 1 to 8 character name of the MERVA ESA routing table used for routing outgoing telex messages.

[6] TESTKEY=ETS6000 specifies the 8 character module name of the test-key processing program to be invoked for the **test-key** command. ETS6000 is the test-key program for the STELEX** product.

Customizing the Telex Message Text

The text area provides text lines with up to 69 characters per line on several pages. The maximum number of lines and the total size of the telex message, including a carriage-return and a line feed character for each transmitted line, are defined during the customization of the Telex Link with the LINES and BUFSIZE parameters of the ENLPARM macro defined in the *MERVA for ESA Macro Reference*.

The Telex Link is supplied with a sample of 120 lines and a buffer size of 6000.

Specifying General Test-Key Requirements

The Telex Link Test-Key Requirements Table (ENLTKRQT) specifies, for specific message types and groups of message types, whether messages of these types must be protected by the addition of a test key. When you create a telex message, the Telex Link determines the test-key requirement and:

- Writes the value (YES or NO) to the field ENLTKIND
- Displays the value in the telex header area after the label **Test** when displaying the telex message on a screen terminal or printing it on a terminal printer

When the telex header is displayed for the first time:

1. The Telex Link sets the value of ENLTKIND to NO.
2. The Telex Link checks whether the Test-Key Requirements Table ENLTKRQT specifies a test-key for this message type. If so, the Telex Link changes the value of ENLTKIND to this value.
3. The Telex Link checks whether there was a record for the sender in the Telex Correspondents File, and whether it specified a test-key for this message type. If so, the Telex Link changes the value of ENLTKIND to this value.

If you enter the correspondent's identification for the first time:

1. The Telex Link again checks whether the Test-Key Requirements Table ENLTKRQT specifies a test-key for this message type. If so, the Telex Link changes the value of ENLTKIND to this value.
2. The Telex Link checks whether there was a record for the correspondent in the Telex Correspondents File, and whether it specified a test-key for this message type. If so, the Telex Link changes the value of ENLTKIND to this value.

You can type over the value determined by the Telex Link.

The Telex Link Test-Key Requirements Table is generated by a sequence of ENLTKREQ macros described in the *MERVA for ESA Macro Reference*:

- TYPE=INITIAL
- TYPE=ENTRY
- TYPE=FINAL

The first macro must be a TYPE=INITIAL macro. The TYPE=ENTRY macros specify the test-key requirements for a particular message type or for a group of message types. The last macro in the sequence must be TYPE=FINAL, and must be followed by the END assembler statement.

When creating a telex message, the Test-Key Requirements Table is searched sequentially for existing messages with the same specification.

An asterisk in any position in the table is always "equal"; and matches any character found in the same position of the message type.

The first entry found in the table determines, by the value of the TK= parameter, the test-key requirement for all message types that satisfy that specification. Therefore, if specific message types and groups of message types overlap, the more specific one must be coded first in the sequence.

For example, in the sample table shown in Figure 52 on page 152, the message type S730, for which a test key is not required, is coded before the more general specification S7** that requires test keys.

The Telex Link provides the sample Test-Key Requirements Table shown in Figure 52 on page 152.

```
ENLTKRQT ENLTKREQ TYPE=INITIAL
         ENLTKREQ MT=S0**,TK=NO
         ENLTKREQ MT=S100,TK=YES
         ENLTKREQ MT=S2**,TK=YES
         ENLTKREQ MT=S3**,TK=NO
         ENLTKREQ MT=S400,TK=YES
         ENLTKREQ MT=S410,TK=NO
         ENLTKREQ MT=S412,TK=YES
         ENLTKREQ MT=S42*,TK=NO
         ENLTKREQ MT=S430,TK=YES
         ENLTKREQ MT=S50*,TK=YES
         ENLTKREQ MT=S51*,TK=NO
         ENLTKREQ MT=S520,TK=YES
         ENLTKREQ MT=S53*,TK=NO
         ENLTKREQ MT=S57*,TK=NO
         ENLTKREQ MT=S580,TK=NO
         ENLTKREQ MT=S730,TK=NO
         ENLTKREQ MT=S7**,TK=YES
         ENLTKREQ MT=S880,TK=NO
         ENLTKREQ MT=S***,TK=NO
         ENLTKREQ MT=TRCV,TK=NO
         ENLTKREQ MT=TSND,TK=NO
         ENLTKREQ MT=****,TK=NO
         ENLTKREQ TYPE=FINAL
         END
```

Figure 52. Sample Table for General Test-Key Requirements

Modifying the SWIFT Link for Telex Processing

When the SWIFT Link is installed together with the Telex Link, you can create SWIFT messages and send them to the receiver either via the SWIFT network or via the telex network. You can also forward messages received from the SWIFT network (SWIFT output messages) to another receiver via the telex network. To do this, the MCBs supplied by the SWIFT Link for each SWIFT message type must be modified as described in the following:

- The MCB of each message type for which the command **test-key extract** is used must be modified.
- Optionally the MCB of each message type can be modified to make the SWIFT message more readable in the telex line format shown in Figure 53 on page 153.

The Telex Link provides example routing tables to combine the functions TX... for telex processing with the functions L2... of the SWIFT Link for processing SWIFT messages. In the routing table ENLRTAI0 (see "Examples of Routing Tables for the Telex Link" on page 66), a test is made to determine whether the telex header information is available:

- If the telex header information is available, the message is routed to the TXTKC, TXURG, or TXNRM queue (depending on the test-key requirements and telex type) for transmission via the telex network.
- If the telex header information is not available, the message is routed to the L2RFINU or L2RFINN queue (depending on message priority) for transmission via the SWIFT network (see "Function Table Entries for Example 2" on page 19).

To prepare SWIFT messages for telex transmission and test-key processing, you modify their MCBs as follows:

- A DSLLDEV TYPE=NET,ID=K section must be available in the MCB of each SWIFT message type for which the Telex Link command **test-key extract** is used to extract currency code, amount and other fields from a SWIFT message for test-key calculation.
- A DSLLDEV TYPE=NET,ID=T section can be used to add extended field tags to a SWIFT message in the telex network format processed with Telex Link via S/1 for better readability.
- A DSLLDEV TYPE=NET,ID=N section can be used to add extended field tags to a SWIFT message in the telex network format processed with Telex Link via workstation for better readability.

A SWIFT message for transmission to the telex network is formatted according to the definitions in the MCBs supplied by the Telex Link and the SWIFT Link:

- The telex information is formatted from the data of the telex header area and the test-key input area.
- The SWIFT message is formatted according to the value specified for the LFMID parameter of the ENLPARM macro in ENLPRM:
  - If LFMID=S is used, the SWIFT messages are formatted using the MCBs supplied by the Telex Link without modification. The SWIFT message is sent in the SWIFT format, especially if the field tags are the same.
  - If LFMID=T is used (which is the default of the LFMID parameter), the SWIFT messages are formatted using the MCBs supplied by the Telex Link. You can use extended field tags for better readability.

Figure 53 shows an example of a SWIFT message type 100 (MT S100) formatted for the telex network with extended field tags.
The first line (starting with "TX") contains transmission instructions for Headoffice

```
TX,185275ABCD
FROM:CASEBANK, 12 RIVER ST.
LONDON, W1
TO  :HALLBANK, MR. BROWN
MANCHESTER, M20
DATE:901115
REF :1234-87
TEST:02992828

::100 CUSTOMER TRANSFER

PLEASE PAY

:20 SENDERS REF:1234-87
:32A VALUE DATE, AMOUNT:901115USD5000,
:50 ORDERING CUSTOMER:FRANZ HOLZAPFEL
:59 BENEFICIARY:H J JANSSEN
```

Figure 53. SWIFT Message Type 100 Customer Transfer Formatted for Telex Transmission

Telex on a fault-tolerant system.

The telex message that is transmitted via the telex network begins on the second line. You can use the Headoffice Telex on a fault-tolerant system's printer to check the format of this telex message.

Defining the Extraction Fields for Test-Key Calculation

If you want to use the Telex Link command **test-key extract** to extract currency code and amount fields from the text of a SWIFT message for test-key calculation, you must include a DSLLDEV TYPE=NET,ID=K section in the MCB of this SWIFT message type.

The Telex Link supplies copy code samples for the SWIFT MTs 100 (ENLS100), 201 (ENLS201), and the common group messages x92 (Request for Cancellation, ENLSX92). During processing of the command **test-key extract**, the fields are copied from the text of the SWIFT message into the fields of the test-key input area.

Figure 54 shows the extraction fields for the SWIFT MT 100 as coded in the copy code ENLS100.

```
LINEK    DSLLDEV  TYPE=NET,ID=K,SEP=X'0D25'                    [1]
         DSLLGRP  GROUP=GRP001                                 [2]
         DSLLNFLD FLD=SW32CUR,TAG=(':TKC:')                    [3]
         DSLLNFLD FLD=SW32AMNT,TAG=(':TKA:')                   [4]
```

Figure 54. SWIFT MT 100 Extraction Fields

Notes:

[1] Defines the net section with the ID=K for the extraction of data for test-key calculation.

[2] Is required by the MERVA ESA MCB logic to define a group.

[3] Defines the currency code subfield SW32CUR of the SWIFT field 32 for extraction. The TAG parameter ':TKC:' identifies the field type to the Telex Link.

The following field types are available:

':TKA:' Extracts an amount field for test-key calculation.

':TKC:' Extracts a currency code field for test-key calculation.

':TKD:' Extracts a date field for test-key calculation.

':TKL:' Extracts data for the letter field for test-key calculation.

':TKT:' Extracts data for the correspondents field for test-key calculation.

[4] Defines the amount subfield SW32AMNT of the SWIFT field 32 for extraction with the TAG parameter ':TKA:'.

Defining Extended Field Tags for the Telex Line Format

A DSLLDEV TYPE=NET,ID=T section can be used to add extended field tags to a SWIFT message in the telex network format for better readability. The Telex Link supplies a sample in the copy code ENLS100 for the SWIFT message type 100 shown in Figure 55 on page 155.

```
LINET    DSLLDEV  TYPE=NET,ID=T,SEP=X'0D25'                              [1]
         DSLLGRP  GROUP=GRP001                                           [2]
         DSLLNFLD TAG=('::S100'),SEP=(X'0D25',ALWAYS)                    [3]
         DSLLNFLD TAG=(' '),SEP=(X'0D25',ALWAYS)                         [4]
         DSLLNFLD TAG=(' CUSTOMER TRANSFER'),SEP=(X'0D25',ALWAYS)        [5]
         DSLLNFLD TAG=(' '),SEP=(X'0D25',ALWAYS)
         DSLLNFLD TAG=(' PLEASE PAY'),SEP=(X'0D25',ALWAYS)               [6]
         DSLLNFLD TAG=(' '),SEP=(X'0D25',ALWAYS)
         DSLLNFLD FLD=SW20,TAG=(':20 SENDERS REF:')                      [7]
         DSLLNFLD FLD=SW32,TAG=(':32 VALUE DATE AMOUNT:')
         DSLLNFLD FLD=SW50,TAG=(':50 ORDERING CUSTOMER:')
         DSLLNFLD FLD=SW52,TAG=(':52 ORDERING BANK:')
         DSLLNFLD FLD=SW53,TAG=(':53 SENDER CORR BANK:')
         DSLLNFLD FLD=SW54,TAG=(':54 RECEIVER CORR BANK:')
         DSLLNFLD FLD=SW57,TAG=(':57 ACCOUNT WITH BANK:')
         DSLLNFLD FLD=SW59,TAG=(':59 BENEFICIARY:')
         DSLLNFLD FLD=SW70,TAG=(':70 DETAILS OF PAYMENT:')
         DSLLNFLD FLD=SW71A,TAG=(':71 DETAILS OF CHARGES:')
         DSLLNFLD FLD=SW72,TAG=(':72 BANK TO BANK INFORMATION:')
```

Figure 55. ENLS100 Copy Code Sample for SWIFT MT 100

Notes:

[1] Defines the net section with the ID=T for the extended field tags.

[2] Is required by the MERVA ESA MCB logic to define a group.

[3] Defines the tag to identify the telex message as a SWIFT MT 100. This tag is always present.

[4] Defines an empty line with one blank.

[5] Defines the title line for the message type 100. Another empty line follows this title line.

[6] Defines another title line. Another empty line follows this title line.

[7] Defines an extended tag for SWIFT field 20 (sender's reference). This tag is only defined if the field SW20 is not empty. The definitions for the other SWIFT fields of MT 100 follow.

Note: The SWIFT message trailer is omitted, as it is only meaningful when sending the message via the SWIFT network.

Interface to the Test-Key Processing Program

When you enter the **test-key** command with one of the parameters **calculate**, **verify**, or **test**, the Telex Link invokes the test-key processing program specified in the Telex Link customizing parameters ENLPRM, passing the buffer shown as a COBOL declaration in Figure 56 on page 156. Assembler attributes are shown in the right-hand column.

```
01  ENLTKBUF.
      02 TKBUFLEN        PIC  9(4) COMP.                 H                  [1]
      02 TKBUFID         PIC X(16).                     CL16               [2]
      02 TKFUNCT         PIC  9(4) COMP.                H                  [3]
      02 TKRETCOD        PIC  9(4) COMP.                H                  [4]
      02 TKERRMSG        PIC X(66).                     CL66               [5]
      02 TKREF           PIC X(16).                     CL16               [6]
      02 TKTSTKEY        PIC X(16).                     CL16               [7]
      02 TKMISSES        PIC  9(4) COMP.                H                  [8]
      02 FILLER          PIC X(20).                     CL20
      02 TKINPUT.
         03 TKUSER       PIC  X(8).                     CL8                [9]
         03 FILLER       PIC  X(4).                     CL4
         03 TKCORR       PIC X(12).                     CL12              [10]
         03 FILLER       PIC  X(2).                     CL2
         03 TKCURAMT     OCCURS 6.
            04 TKCURRCY  PIC X(3).                      CL3               [11]
            04 TKAMOUNT  OCCURS 3 PIC S9(15)V999 COMP-3.  PL10,PL10,   PL10 [12]
                     (other 5 occurrences in assembler:) 165X
         03 TKLETTRS     PIC X(4).                      CL4               [13]
         03 FILLER       PIC X(6).                      CL6
         03 TKDDMMYY     PIC 9(6).                      CL6               [14]
```

Figure 56. Declaration of the Test-Key Processing Interface Buffer

Notes:

[1] The total length of the interface buffer.

[2] The control block identifier set by the Telex Link.

[3] The code for the requested function:

1. To request test-key calculation
2. To request test-key verification
3. To request a test of the test-key calculation
4. To request a test of the test-key verification

[4] The return code of the test-key processing program. It has one of the following values:

0 The invocation was successful.

4 The verification was successful, but gaps in the sequence number were detected.

8 The verification of the specified test key failed.

12 The test-key calculation failed, for example, because of an overflow when adding values.

16 Incorrect input values are passed in the interface buffer.

[5] Contains an error message of the test-key processing program if the return code is not zero.

[6] A reference identification that can be used for journaling test-key processing requests and results. The Telex Link moves the telex-reference identification into this field.

[7] The area where the calculated test key is returned after test-key calculation requests. For test-key verification requests (including the test request), this field contains the test key verified by the test-key processing program.

[8] The number of gaps detected in the sequence of telex messages received from the correspondent for which a test key is verified. A return code of 4 indicates the gaps.

[9] The user ID used for checking the authorization and logging the request.

[10] The correspondent's ID for which the test key is calculated or verified.

[11] The currency code (ISO standard) or XXX for the up to three following amounts. The set of one currency code and three amounts is available six times.

[12] Is used for up to three amounts for each of the six currency codes.

[13] A field where letters are passed that are used, if applicable, in the test-key algorithm.

[14] The field for the date of the test key. For test-key calculation, the test-key processing program returns the current system date in this field. For test-key verification, the Telex Link passes the specified date (default being the system date) to the test-key processing program.

Automatic Test-Key Facility

Telex Link provides a copy code sample for automatically calling the test-key facility. The automatic test-key facility (ATK) can be used to:

- Automatically extract the test-key fields as described in the test-key MCB for SWIFT telex messages
- Automatically access the test-key facility and test-key calculation using the extracted data
- Automatically route messages using the default routing module, or a user written test-key routing module
- Modifiable exception processing

Use the following entry, shown in Figure 57, into DSLMU023 to activate the sample facility with a call to module ENLMU397.

```
DSLMU023 DSLMFS  MF=START,TYPE=USER,MODNUM=023,OPT=EXTTS
⋮
         DSLMFS TYPE=USER,MODNUM=397,MF=(E,MFSTLIST)
         MVC   MFSLREAS-MFSL(2,R7),MF$LREAS-MFSTLIST(R1)
         LTR   R15,R15                  RETURN OK?
         BNZ   MFSERWNG                 NO, SET WARNING RETURN
         B     MFSGOOD
⋮
```

Figure 57. Activate ATK in User Exit DSLMU023

Note: When the DSLCXT program (DSLX transaction) is used in other functions of your installation, ensure that the user exit 397 is called only in the function TXTKCA. The current function name can be found in the field TUCNAME in the Terminal and User Control Block (TUCB).

Table 8 on page 158 describes how ATK is activated in the Telex Link.

Table 8. Activating ATK in the Telex Link

Involved Modules		Status	Compile	Link
ENLFNTTC	COPY	Function TXTKCA is prepared to call transaction DSLCXT.	DSLFNTT	DSLFNTT
ENLMPTTC	COPY	Entries for ENLMU397 ENLMU398 are prepared.	DSLMPTT	DSLMMFS
ENLMU397	OBJ	ENLMU397 performs the field extraction and invokes the call to the test-key calculation program as specified in ENLPRM.	--------	ENLMU397
ENLMU398	ASSEMBLE	Source code supplied.	ENLMU398	DSLMMFS
DSLMU023	ASSEMBLE	To be modified as indicated above.	DSLMU023	DSLMU023
ENLRTDE0	ASSEMBLE	To be modified on customers requirements.	ENLRTDE0	ENLRTDE0

Installation Notes:

[1] Test-key Functions.

The following two Test-key Calculation functions are recommended:

- DSLCXT transaction for automatic call
- Manual test-key calculation

It is also recommended that the routing module that routes messages to the test-key facility, tests messages initially for their message type. This is to determine whether the text of the message is in a SWIFT format, or a free formatted Telex. If the message is in SWIFT format, routing to the automatic test-key function can take place. If the message is a free formatted Telex, it should be routed to the manual test-key function.

The SWIFT Link MCBs must be prepared to support test-key extraction. A network device description for ID=K must be added for all message types that require automatic test-key calculation. Ensure that the program ENLMU397 is link-edited correctly, and resides in the MERVA load library. This program is loaded on demand by the message format service.

[2] Operator Authorization for the Test-key Calculation Facility.

Since there is no actual operator associated with the ATK function, you need to add the Auto Test-key Function name as a test-key operator to the test-key facility. You can restrict the use of the ATK function, with the special test-key function, so that only users with access to a certain DATA ENTRY function have Telex messages routed to them.

[3] In the IMS environment, the INQUIRY=(YES,NORECOV) parameter must be removed from the transaction definition used for ATK.

[4] In the IMS environment, the PSBGEN of the PSB used for ATK must be extended by the PSBs required by the ATK software. For details, refer to the documentation of the ATK software used.

Sample Description

The transaction DSLCXT is automatically started after a SWIFT type message with a telex header is created and routed to the ATK function TXTKCA. As part of DSLCXT the User Exit DSLMU023 is used and ENLMU397, as the actual automatic test-key program is started if the required modifications have been made. Test-key data can be extracted and presented to the Test-key Program.

Test-Key Facility Processing Exceptions

When a processing exception in the test-key facility occurs the following processing is performed:

- If there is a DSLxxx or ENLxxx message, the message is routed to your error function and processing continues.
- If there is an exception message from the test-key calculation program, the User Exit ENLMU398, containing a table of all exception message numbers that cause Automatic Test-key processing to stop, obtains control. The only message number provided in the default user exit ENLMU308 is ETSS281. The error message is then displayed on the console log list and you can use the **dm last** command to display the MERVA ESA command panel. When the problem is resolved, use the SF (**start function**) command to continue processing. If the exceptional message is not specified in this table, processing is performed in outline for DSLxxx and ENLxxx messages.

Use either the **hold** or the **sf** command to manually control the test-key function.

Disabling Telex Link Long Answerback (LAB)

The Telex Link provides customizable copy codes and source codes that you can use to disable the long answerback (LAB) checking method, for example, if your Telex Link does not support LAB.

LAB provides separate data entry fields in the Telex header and the Telex Correspondents File that enable answerback codes which are independent of the dial-up number to be entered. This means that the answerback code can be any valid Telex character string. To disable LAB, follow these customization steps:

1. Edit the following copy and source codes (following the instructions in their comment sections):
   - Source code ENLTCOR
   - Source code ENLLTXIP
   - Copy code ENLTX
   - Copy code ENLXL
   - Copy code ENLXLN
   - Copy code ENLXLX
   - Copy code ENLXMSG
2. Assemble and link-edit the following sources:
   - ENLLTXIP
   - ENLMTINV
   - ENLMTRCV
   - ENLMTSND
   - ENLTCOR
   - ENLTXHD

Note: These copy and source codes must be edited in accordance with the assembler language rules.

Telex Link Additional Transmit Data

Telex Link via a fault-tolerant system provides an open interface in order to receive additional transmit data from the Headoffice Telex on a fault-tolerant system, which is written into TOF field ENLXMEXT.

The Headoffice Telex on a fault-tolerant system used must support additional transmit data.

The open interface is implemented as follows:

- A received Telex is checked for the tag '#' directly after the Telex Header (refer to Appendix C in *MERVA for ESA Installation Guide*).

  If this tag is not found, normal processing takes place.

  If this tag is found, all data (including the tag) up to the separator X'0D25' is written into TOF field ENLXMEXT. If more than 69 bytes are found, the data between the tag '#' and the separator X'0D25' is split into data areas of ENLXMEXT.
- After the separator, normal processing of the received Telex takes place.
- Any required parts of ENLXMEXT can be defined as subfields of field ENLXMEXT.
- MERVA ESA performs no further processing on the additional transmit data stored in ENLXMEXT.

To send additional transmit data to Headoffice Telex on a fault-tolerant system the net section of the ENLLTXIP MCB may be modified to include any additional tags, data fields and separators in the transmitted Telex. In order to do this, Telex Link via a fault-tolerant system must support this additional transmitted data.

Telex Link Sample Code

The MERVA ESA sample library contains the part DSLBT01A, which is a collection of code parts to enhance the functions of Telex Link.

Chapter 6. The MERVA Link for CICS and IMS

This chapter describes how to adapt the MERVA Link functions executing in the MERVA CICS and MERVA IMS environments to meet the requirements of your installation and organization. The customization of the MERVA Link functions executing in the OS/390 UNIX System Services (USS) environment are described in "Chapter 7. The MERVA Link for Unix System Services (USS)" on page 243.

Defining Partner Table ASP and MTP Entries (Samples)

Use the MERVA Link partner table (PT) to customize MERVA Link in the CICS and IMS environments. The PT contains the definitions of all the MERVA Link processes that support the exchange of messages between partner systems. It consists of a header and a number of different entries, such as entries to define the parameters of an application support process (ASP) or entries to define the parameters of a message transfer process (MTP).

The parameters of an ASP are, for example:

- The ASP names and the address of the partner ASP
- The name of the send queues and the ASP control queue
- Message transmission format information
- The name of the MTP associated with this ASP

The parameters of an MTP are, for example:

- The MTP names (internal, external) and the name of the partner MTP
- Identifier of the partner system or a gateway system
- The name of the ASP associated with this MTP

The following samples support the symmetric exchange of messages between the participating partner systems. Sending and receiving systems can exchange their roles. Any local system can be seen as the remote system and any remote system can be seen as the local system.

The sample MERVA Link network consists of a number of MERVA Link nodes. These nodes are numbered and characterized as follows:

Node 1 (C1) The first MERVA CICS system with the CICS TS VTAM APPLID (LU Name) CTS1LUNM.

Node 2 (C2) The second MERVA CICS system with the CICS TS VTAM APPLID (LU Name) CTS2LUNM.

Node 3 (I1) The first MERVA IMS system with the APPC/MVS System Base LU name MVS1LUNM and the APPC/IMS Base LU name IA01LUNM.

Node 4 (I2) The second MERVA IMS system with the APPC/MVS System Base LU name MVS2LUNM and the APPC/IMS Base LU name IA02LUNM.

Node 9 (U1) A MERVA Link USS Gateway with the APPC/MVS System Base LU name MVS9LUNM.

Sample 1: Interconnecting Two MERVA Link CICS Systems

The sample PT generation statements below are for two interconnected MERVA ESA CICS systems called C1 and C2:

```
        EKAPT TYPE=INITIAL,NODE=C1,   MERVA LINK NODE 1 (CICS)        *    [1]
              TRACE=EKAT              CONVERSATION TRACE DS IDENTIFIER    [2]
        EKAPT TYPE=ASP,               APPLICATION SUPPORT PROCESS     *
              NAME=A2A,               ASP NAME                        *    [3]
              SENDQC=(EKA2AS1,EKA2AS2,EKA2AS3), SEND QUEUE CLUSTER    *    [4]
              DEST=(C2,A1A),          PARTNER ASP ADDRESS             *    [5]
              MTP=T2A,                NAME OF APPLICABLE MTP          *    [6]
              CONTROL=(EKA2ACQ,5),    CTRL QUEUE NAME AND WINDOW SIZE *    [7]
              MFSEXIT=7010,           USER EXIT NUMBER                *    [8]
              FORMAT=QUEUE,           TRANSMIT IN MERVA QUEUE FORMAT  *    [9]
              IRROUTE=(ACK,EKAAWQ,CTLQ) CORRELATE RECEIPT REPORTS          [10]
        EKAPT TYPE=MTP,               MESSAGE TRANSFER PROCESS        *   [11]
              NAME=(T2A,X12A),        INTERNAL AND EXTERNAL MTP NAME  *   [12]
              LINK=(APPC,CA02),       REMOTE SYSTEM INFORMATION       *   [13]
              PARTNER=(X21A,EKAR),    REMOTE PROCESS INFORMATION      *   [14]
              ASP=A2A                 NAME OF APPLICABLE ASP              [15]
*       EKAPT TYPE=MTP,               MTP USING A GATEWAY NODE        *   [16]
              NAME=T2A,               INTERNAL MTP NAME               *   [17]
              DEST=(MVS9,EKAR1),      GATEWAY SYSTEM ISC INFORMATION  *   [18]
              ASP=A2A                 NAME OF APPLICABLE ASP
        EKAPT TYPE=FINAL
        END
```

Figure 58. PT Sample 1: Node 1 (CICS System C1)

```
        EKAPT TYPE=INITIAL,NODE=C2,   MERVA LINK NODE 2 (CICS)        *    [1]
              TRACE=EKAT              CONVERSATION TRACE DS IDENTIFIER    [2]
        EKAPT TYPE=ASP,               APPLICATION SUPPORT PROCESS     *
              NAME=A1A,               ASP NAME                        *    [3]
              SENDQC=(EKA1AS1,EKA1AS2,EKA1AS3), SEND QUEUE CLUSTER    *    [4]
              DEST=(C1,A2A),          PARTNER ASP ADDRESS             *    [5]
              MTP=T1A,                NAME OF APPLICABLE MTP          *    [6]
              CONTROL=(EKA1ACQ,5),    CTRL QUEUE NAME AND WINDOW SIZE *    [7]
              MFSEXIT=7010,           USER EXIT NUMBER                *    [8]
              FORMAT=QUEUE,           TRANSMIT IN MERVA QUEUE FORMAT  *    [9]
              IRROUTE=(ACK,EKAAWQ,CTLQ) CORRELATE RECEIPT REPORTS          [10]
        EKAPT TYPE=MTP,               MESSAGE TRANSFER PROCESS        *   [11]
              NAME=(T1A,X21A),        INTERNAL AND EXTERNAL MTP NAME  *   [12]
              LINK=(APPC,CA01),       REMOTE SYSTEM INFORMATION       *   [13]
              PARTNER=(X12A,EKAR),    REMOTE PROCESS INFORMATION      *   [14]
              ASP=A1A                 NAME OF APPLICABLE ASP              [15]
*       EKAPT TYPE=MTP,               MTP USING A GATEWAY NODE        *   [16]
              NAME=T1A,               INTERNAL MTP NAME               *   [17]
              DEST=GATEWAY9,          GATEWAY PARTNER NAME            *   [18]
              ASP=A1A                 NAME OF APPLICABLE ASP
        EKAPT TYPE=FINAL
        END
```

Figure 59. PT Sample 1: Node 2 (CICS System C2)

The node C1 is the local node; C2 is the partner node. Because of the symmetry of MERVA Link connections, the role of the local system or the partner system can always be exchanged.

Notes:

[1] The local node name is C1. It must be specified by the partner ASP as the

partner node name (see [5]). The partner node name is C2. It must be specified by the local ASP as the partner node name (see [5]).

[2] The MERVA Link conversation trace is written to the CICS Transient Data queue EKAT.

[3] The local ASP name is A2A, an ASP with a partner located in node 2. The partner ASP name is A1A, an ASP with a partner located in node 1. ASP Free Form Names are not used in this sample.

[4] Messages can be passed to MERVA Link by a MERVA ESA application in one of the 3 send queues for being transferred to the partner application.

[5] The address of the partner ASP consists of the partner node name and the name of the partner ASP in that node.

[6] The MTP associated with ASP A2A is named T2A within the local node (internal MTP name). MTP T2A is defined in the following EKAPT macro. The MTP associated with ASP A1A is named T1A within the partner node (internal MTP name). MTP T1A is defined in the following EKAPT macro.

[7] A MERVA Link ASP uses a MERVA ESA queue for internal control purposes. This queue must be reserved exclusively for this ASP. EKA2ACQ is the control queue of ASP A2A. EKA1ACQ is the control queue of ASP A1A. The window size of the ASPs is 5.

[8] A MERVA ESA MFS user exit is associated with these ASPs. It controls the messages during a sending and during a receiving process.

[9] The ASPs transfer messages in the MERVA ESA internal queue format.

[10] Incoming acknowledgment messages (status reports) must be correlated and merged with the reported message in the queue EKAAWQ (Ack Wait Queue). The acknowledged message must be routed as specified by the routing table associated with the control queue of this ASP.

[11] The active MTP definition interconnects the two partner systems directly. An alternative is shown starting at [16].

[12] The internal and external names of the local MTP are T2A and X12A, respectively. The partner MTP refers only to the external MTP name X12A. The internal and external names of the partner MTP are T1A and X21A, respectively. The local MTP refers only to the external MTP name X21A.

[13] The APPC connection to the partner system is defined in CICS 1 under the connection name CA02. The APPC connection to the local system is defined in CICS 2 under the connection name CA01.

[14] The external name of the partner MTP is X21A. The transaction code of the APPC back-end process (receiving MTP) in the partner system is EKAR. The external name of the local MTP is X12A. The transaction code of the APPC back-end process (receiving MTP) in the local system is EKAR.

[15] The ASP associated with the local MTP is named A2A. The ASP associated with the partner MTP is named A1A.

[16] An alternate MTP definition for the interconnection of two MERVA Link CICS systems is shown to provide an example for the connection to a MERVA Link USS Gateway.

[17] External MTP names are not used by the cooperating MTPs in this sample. External MTP names must, however, be used for connections to MERVA Link ESA Version 3 (and earlier).

[18] The intersystem connection parameters (partner system ID and partner TP name) can be provided in the DEST parameter instead of the LINK and PARTNER parameters. These parameters can be specified directly (see NODE=C1), or indirectly by referring to a CICS Partner definition (see NODE=C2).

The destination of outbound conversations from C1 and C2 is a MERVA Link USS Gateway (sample MERVA Link node 9). This gateway routes the conversation to the intended destination CICS node (C2 or C1). Interconnecting two MERVA Link CICS systems via a MERVA Link Gateway makes not much sense because they can communicate directly. This sample may, however, be used as a reference for a connection to a MERVA Link gateway to interconnect a MERVA Link CICS node and a partner MERVA Link node using TCP/IP.

Sample 2: Interconnecting MERVA Link CICS and IMS Systems

The sample PT generation statements below are for two interconnected MERVA ESA systems: one MERVA ESA CICS system (C1) and one MERVA ESA IMS system (I1).

```
        EKAPT TYPE=INITIAL,NODE=C1,   MERVA LINK NODE 1 (CICS)        *    [1]
              TRACE=EKAT              CONVERSATION TRACE DS IDENTIFIER     [2]
        EKAPT TYPE=ASP,               APPLICATION SUPPORT PROCESS     *
              NAME=A3A,               ASP NAME                        *    [3]
              SENDQC=(EKA3AS1,EKA3AS2,EKA3AS3), SEND QUEUE CLUSTER    *    [4]
              DEST=(I1,A1A),          PARTNER ASP ADDRESS             *    [5]
              MTP=T3A,                NAME OF APPLICABLE MTP          *    [6]
              CONTROL=(EKA3ACQ,10),   CTRL QUEUE NAME AND WINDOW SIZE *    [7]
              MFSEXIT=7010,           USER EXIT NUMBER                *    [9]
              FORMAT=QUEUE,           TRANSMIT IN MERVA QUEUE FORMAT  *   [10]
              IRROUTE=(ACK,EKAAWQ,CTLQ) CORRELATE RECEIPT REPORTS          [11]
        EKAPT TYPE=MTP,               MESSAGE TRANSFER PROCESS        *   [12]
              NAME=(T3A,X13A),        INTERNAL AND EXTERNAL MTP NAME  *   [13]
              LINK=(APPC,MVS1,EKAPROF), REMOTE SYSTEM INFORMATION     *   [14]
              PARTNER=(X31A,EKAR),    REMOTE PROCESS INFORMATION      *   [15]
              ASP=A3A                 NAME OF APPLICABLE ASP              [16]
*       EKAPT TYPE=MTP,               MTP USING A GATEWAY NODE        *   [17]
              NAME=T3A,               INTERNAL MTP NAME               *   [18]
              DEST=(MVS9,EKAR1),      GATEWAY SYSTEM ISC INFORMATION  *   [19]
              ASP=A3A                 NAME OF APPLICABLE ASP
*
        EKAPT TYPE=FINAL
        END
```

Figure 60. PT Sample 2: Node 1 (CICS System C1)

```
         EKAPT TYPE=INITIAL,NODE=I1    MERVA LINK NODE 3 (IMS 1)       *    [1]
         EKAPT TYPE=ASP,               APPLICATION SUPPORT PROCESS     *
               NAME=A1A,               ASP NAME                        *    [3]
               SENDQC=(EKA1AS1,EKA1AS2,EKA1AS3), SEND QUEUE CLUSTER    *    [4]
                DEST=(C1,A3A),          PARTNER ASP ADDRESS             *   [5]
               MTP=T1A,                NAME OF APPLICABLE MTP          *    [6]
               CONTROL=(EKA1ACQ,20),   CTRL QUEUE NAME AND WINDOW SIZE *    [7]
               TRAN=EKASA1,            SENDING ASP TRANSACTION ID      *    [8]
               MFSEXIT=7010,           USER EXIT NUMBER                *    [9]
               FORMAT=QUEUE,           TRANSMIT IN MERVA QUEUE FORMAT  *   [10]
               IRROUTE=(ACK,EKAAWQ,CTLQ) CORRELATE RECEIPT REPORTS         [11]
         EKAPT TYPE=MTP,               MESSAGE TRANSFER PROCESS        *   [12]
               NAME=(T1A,X31A),        INTERNAL AND EXTERNAL MTP NAME  *   [13]
               LINK=(APPC,CTS1LUNM,APPCLU62), REMOTE SYSTEM INFO       *   [14]
               PARTNER=(X13A,EKAR),    PARTNER MTP INFORMATION         *   [15]
               ASP=A1A                 NAME OF APPLICABLE ASP              [16]
*        EKAPT TYPE=MTP,               MTP USING A GATEWAY NODE        *   [17]
               NAME=T1A,               INTERNAL MTP NAME               *   [18]
               DEST=(MVS9,EKAR1),      GATEWAY SYSTEM ISC INFORMATION  *   [19]
               ASP=A1A                 NAME OF APPLICABLE ASP
         EKAPT TYPE=FINAL
         END
```

Figure 61. PT Sample 2: Node 3 (IMS System I1)

MERVA ESA node C1 is the local node. MERVA ESA node I1 is the partner node. The role of the local system or the partner system can always be exchanged because of the symmetry of MERVA Link connections.

Notes:

[1] The local node name is C1. It must be specified by the partner ASP as the partner node name. The partner node name is I1. It must be specified by this ASP as the partner node name.

[2] The MERVA Link conversation trace is written to the CICS Transient Data queue EKAT. The external conversation trace is not supported in the MERVA Link IMS environment.

[3] The local ASP name is A3A, an ASP with a partner located in node 3. The partner ASP name is A1A, an ASP with a partner located in node 1. ASP Free Form Names are not used in this sample.

[4] Messages can be passed to MERVA Link by a MERVA ESA application in one of the 3 send queues for being transferred to the partner application.

[5] The address of the partner ASP consists of the partner node name and the name of the partner ASP in that node.

[6] The MTP associated with ASP A3A is named T3A within the local node (internal MTP name). MTP T3A is defined in the following EKAPT macro. The MTP associated with ASP A1A is named T1A within the partner node (internal MTP name). MTP T1A is defined in the following EKAPT macro.

[7] A MERVA Link ASP uses a MERVA ESA queue for internal control purposes. This queue must be reserved exclusively for this ASP. EKA3ACQ is the control queue of ASP A3A. The window size of this ASP is 10. EKA1ACQ is the control queue of ASP A1A. The window size of this ASP is 20.

[8] The transaction code of the sending ASP defaults to EKAS in the local (CICS) system. In the partner (IMS) system, the transaction code of the

sending ASP is EKASA1. In MERVA Link CICS, all ASPs can have the same transaction code. In MERVA Link IMS, each ASP must have a unique transaction code.

[9] A MERVA ESA MFS user exit is associated with each ASP. It controls the messages during a sending and during a receiving process.

[10] The ASPs exchange messages in the MERVA ESA internal queue format.

[11] Incoming acknowledgment messages (status reports) must be correlated and merged with the reported message in the queue EKAAWQ (Ack Wait Queue). The acknowledged message must be routed as specified by the routing table associated with the control queue of this ASP.

[12] The active MTP definition interconnects the two partner systems directly. An alternative is shown starting at [16].

[13] The internal and external names of the local MTP are T3A and X13A, respectively. The partner MTP refers only to the external MTP name X13A. The internal and external names of the partner MTP are T1A and X31A, respectively. The local MTP refers only to the external MTP name X31A.

[14] The connection to the partner system is defined in CICS under the connection name MVS1. The type of this connection is APPC. The name of the APPC profile is EKAPROF. In the IMS APPC/MVS environment the connection to a partner system is directly specified. The connection identifiers are the VTAM LU name and the name of the VTAM Logon Mode table entry.

[15] The external name of the partner MTP is X31A. The transaction code of the APPC back-end process (receiving IMS MTP) in the partner system is EKAR. EKAR is the name of an APPC/MVS TP profile in the partner system. The external name of the local MTP is X13A. The transaction code of the APPC back-end process (receiving CICS MTP) in the local system is EKAR.

[16] The ASP associated with the local MTP is named A3A. ASP A3A is defined in the previous EKAPT macro.

The ASP associated with the partner MTP is named A1A. ASP A1A is defined in the previous EKAPT macro.

[17] An alternate MTP definition for the interconnection of MERVA Link CICS and IMS systems is shown to provide an example for the connection to a MERVA Link USS Gateway.

[18] External MTP names are not used by the cooperating MTPs in this sample. External MTP names must, however, be used for connections to MERVA Link ESA Version 3 (and earlier).

[19] The intersystem connection parameters are provided in the DEST parameter instead of the LINK and PARTNER parameters. In a MERVA Link CICS system, these parameters are Partner System ID and Partner TP Name. In a MERVA Link IMS system, the DEST parameter specifies a Symbolic Destination defined in APPC/MVS Side Information. The sample symbolic destination name is MVS9.

The destination of outbound conversations from C1 and I1 is a MERVA Link USS Gateway (sample MERVA Link node 9). This gateway routes the conversation to the intended destination nodes I1 and C1. Interconnecting two MERVA Link ESA systems via a MERVA Link Gateway makes not much sense because they can communicate directly. This sample can,

however, be used as a reference for a connection to a MERVA Link gateway to interconnect a MERVA Link CICS or IMS node and a partner MERVA Link node using TCP/IP.

Sample 3: Interconnecting Two MERVA Link IMS Systems

This PT sample shows a connection of two MERVA ESA IMS systems I1 and I2.

- In I1, the receiving MTP runs in an APPC/MVS initiator.
- In I2, the receiving MTP runs in an IMS MPR.

The sample PT generation statements for both systems are shown in Figure 62 and in Figure 63.

```
EKAPT TYPE=INITIAL,NODE=I1    MERVA LINK NODE 3 (IMS 1)             [1]
EKAPT TYPE=ASP,               APPLICATION SUPPORT PROCESS     *
      NAME=A4A,               ASP NAME                        *    [2]
      SENDQC=EKA4AS1,         SEND QUEUE CLUSTER              *    [3]
      DEST=(I2,A3A),          PARTNER ASP ADDRESS             *    [4]
      MTP=T4A,                NAME OF APPLICABLE MTP          *    [5]
      CONTROL=(EKA4ACQ,2),    CTRL QUEUE NAME AND WINDOW SIZE *    [6]
      TRAN=EKASA4,            SENDING ASP TRANSACTION ID      *    [7]
      IRROUTE=(ACK,EKAAWQ,CTLQ) CORRELATE RECEIPT REPORTS          [8]
EKAPT TYPE=MTP,               MESSAGE TRANSFER PROCESS        *
      NAME=(T4A,X34A),        INTERNAL AND EXTERNAL MTP NAME  *    [9]
      LINK=(APPC,IA02LUNM,APPCLU62), REMOTE SYSTEM INFO       *   [10]
      PARTNER=(X43A,EKARI510), PARTNER EXTERNAL MTP NAME      *   [11]
      ASP=A4A                 NAME OF APPLICABLE ASP              [12]
EKAPT TYPE=FINAL
END
```

Figure 62. PT Sample 3: Node 3 (IMS System I1)

```
EKAPT TYPE=INITIAL,NODE=I2    MERVA LINK NODE 4 (IMS 2)             [1]
EKAPT TYPE=ASP,               APPLICATION SUPPORT PROCESS     *
      NAME=A3A,               ASP NAME                        *    [2]
      SENDQC=EKA3AS1,         SEND QUEUE CLUSTER              *    [3]
      DEST=(I1,A4A),          PARTNER ASP ADDRESS             *    [4]
      MTP=T3A,                NAME OF APPLICABLE MTP          *    [5]
      CONTROL=(EKA3ACQ,50),   CTRL QUEUE NAME AND WINDOW SIZE *    [6]
      TRAN=EKASA3,            SENDING ASP TRANSACTION ID      *    [7]
      IRROUTE=(ACK,EKAAWQ,CTLQ) CORRELATE RECEIPT REPORTS          [8]
EKAPT TYPE=MTP,               MESSAGE TRANSFER PROCESS        *
      NAME=(T3A,X43A),        INTERNAL AND EXTERNAL MTP NAME  *    [9]
      LINK=(APPC,MVS1LUNM,APPCLU62), REMOTE SYSTEM INFO       *   [10]
      PARTNER=(X34A,EKAR),    PARTNER EXTERNAL MTP NAME       *   [11]
      ASP=A3A                 NAME OF APPLICABLE ASP              [12]
EKAPT TYPE=FINAL
END
```

Figure 63. PT Sample 3: Node 4 (IMS System I2)

MERVA ESA node I1 is the local node. MERVA ESA node I2 is the partner node. The role of the local system or the partner system can always be exchanged because of the symmetry of MERVA Link connections.

Notes:

[1] The local node name is I1. It must be specified by the partner ASP A3A as the partner node name. The partner node name is I2. It must be specified by the local ASP A4A as the partner node name.

[2] The local ASP name is A4A, an ASP with a partner located in node 4. The

partner ASP name is A3A, an ASP with a partner located in node 3. ASP Free Form Names are not used in this sample.

[3] Messages can be passed to MERVA Link by a MERVA ESA application in one send queue for being transferred to the partner application.

[4] The address of the partner ASP consists of the partner node name and the name of the partner ASP in that node.

[5] The MTP associated with ASP A4A is named T4A within the local node (internal MTP name). MTP T4A is defined in the following EKAPT macro. The MTP associated with ASP A3A is named T3A within the partner node (internal MTP name). MTP T3A is defined in the following EKAPT macro.

[6] A MERVA Link ASP uses a MERVA ESA queue for internal control purposes. This queue must be reserved exclusively for this ASP. EKA4ACQ is the control queue of ASP A4A. The window size of this ASP is 2. EKA3ACQ is the control queue of ASP A3A. The window size of this ASP is 50.

[7] The transaction code of the sending ASP is EKASA4 in the local system. In the partner system, the transaction code of the sending ASP is EKASA3.

In MERVA Link IMS, each sending ASP must have a unique transaction code.

[8] Incoming acknowledgment messages (status reports) must be correlated and merged with the reported message in the queue EKAAWQ (Ack Wait Queue). The acknowledged message must be routed as specified by the routing table associated with the control queue of this ASP.

[9] The internal and external names of the local MTP are T4A and X34A, respectively. The partner MTP refers only to the external MTP name X34A. The internal and external names of the partner MTP are T3A and X43A, respectively. The local MTP refers only to the external MTP name X43A.

[10] The identifier of the partner system and the Logon Mode Table entry name are directly specified in the LINK parameter in the APPC/MVS environment. The local system I1 uses LU MVS1LUNM, an APPC/MVS LU associated with the APPC/MVS transaction scheduler ASCH, to handle inbound conversations. The partner system I2 uses LU IA02LUNM for that purpose. IA02LUNM is an an APPC/MVS LU associated with the APPC/IMS transaction scheduler I510. The Logon Mode Table entry name is APPCLU62 in both systems.

[11] The external name of the partner MTP is X43A. The transaction code of the APPC back-end process (receiving MTP) in the partner system is EKARI510. EKARI510 is the name of an APPC/IMS TP profile in the partner system. The receiving MTP in node 4 runs in an IMS MPR.

The external name of the local MTP is X34A. The transaction code of the APPC back-end process (receiving MTP) in the local system is EKAR. EKAR is the name of an APPC/MVS TP profile in the local system. The receiving MTP in node 3 runs in an APPC/MVS initiator.

[12] The ASP associated with the local MTP is named A4A. ASP A4A is defined in the previous EKAPT macro. The ASP associated with the partner MTP is named A3A. ASP A3A is defined in the previous EKAPT macro.

Defining Partner Table SCP Entries (Samples)

MERVA Link supports the partner MERVA system control function (PMSC), which allows the MERVA MSC function to be run in a partner MERVA ESA system. A partner system control process (SCP) must be defined in the partner tables of both the client and the server MERVA Link systems to enable the PMSC function.

The PMSC function can be enabled between all sample MERVA Link nodes 1 to 4. It is, however, not supported by MERVA Link USS (sample node 9). SCP definitions for the sample MERVA Link nodes 1 to 4 are shown in the following. These samples use the DEST parameter to identify the PMSC server in the partner system. The legacy LINK and PARTNER parameters of the EKAPT TYPE=SCP statement can still be used for that purpose as an alternative to the DEST parameter.

Sample SCPs for Node 1

This PT sample shows SCP definitions that provide for controlling sample MERVA Link nodes 2 to 4 at node 1.

```
         EKAPT TYPE=SCP,               SYSTEM CONTROL PROCESS          *    [1]
               NAME=(NODE2,C2),        PARTNER NODE NAMES              *    [2]
               DEST=(CA02,EKAC),       PARTNER SYSTEM ISC INFORMATION  *    [3]
               LOPER=ALL,              AUTHORIZED LOCAL OPERATORS      *    [4]
                POPER=ALL               AUTHORIZED PARTNER OPERATORS         [5]
*
         EKAPT TYPE=SCP,               SYSTEM CONTROL PROCESS          *    [1]
               NAME=(NODE3,I1),        PARTNER NODE NAMES              *    [2]
               DEST=(MVS1,EKAC),       PARTNER SYSTEM ISC INFORMATION  *    [3]
               LOPER=ALL,              AUTHORIZED LOCAL OPERATORS      *    [4]
               POPER=ALL               AUTHORIZED PARTNER OPERATORS         [5]
*
         EKAPT TYPE=SCP,               SYSTEM CONTROL PROCESS          *    [1]
               NAME=(NODE4,I2),        PARTNER NODE NAMES              *    [2]
               DEST=(IA02,EKACI510),   PARTNER SYSTEM ISC INFORMATION  *    [3]
               LOPER=ALL,              AUTHORIZED LOCAL OPERATORS      *    [4]
               POPER=ALL               AUTHORIZED PARTNER OPERATORS         [5]
```

Figure 64. Sample SCP Definitions for Node 1

Notes:

[1] An SCP entry in the PT provides for controlling a partner MERVA Link system.

[2] The NAME parameter specifies a nickname and the MERVA Link node name for the applicable partner MERVA ESA system.

[3] The DEST parameter provides the intersystem connection parameters for the PMSC function (or refers to a CICS PARTNER definition). The connections and partners defined in CICS for the Message Transfer function can also be used for the PMSC function. A proprietary partner transaction identifier must, however, be specified for the PMSC function.

[4] The LOPER parameter controls the authorization of local MERVA ESA operators to access the applicable partner system.

[5] The POPER parameter controls the authorization of MERVA ESA operators in the partner system to access the local MERVA ESA system.

Sample SCPs for Node 2

This PT sample shows SCP definitions that provide for controlling sample MERVA Link nodes 1, 3, and 4 at node 2.

```
        EKAPT TYPE=SCP,               SYSTEM CONTROL PROCESS          *    [1]
              NAME=(NODE1,C1),        PARTNER NODE NAMES              *    [2]
              DEST=(CA01,EKAC),       PARTNER SYSTEM ISC INFORMATION  *    [3]
              LOPER=ALL,              AUTHORIZED LOCAL OPERATORS      *    [4]
              POPER=ALL               AUTHORIZED PARTNER OPERATORS         [5]
*
        EKAPT TYPE=SCP,               SYSTEM CONTROL PROCESS          *    [1]
              NAME=(NODE3,I1),        PARTNER NODE NAMES              *    [2]
              DEST=(MVS1,EKAC),       PARTNER SYSTEM ISC INFORMATION  *    [3]
              LOPER=ALL,              AUTHORIZED LOCAL OPERATORS      *    [4]
              POPER=ALL               AUTHORIZED PARTNER OPERATORS         [5]
*
        EKAPT TYPE=SCP,               SYSTEM CONTROL PROCESS          *    [1]
              NAME=(NODE4,I2),        PARTNER NODE NAMES              *    [2]
              DEST=(IA02,EKACI510),   PARTNER SYSTEM ISC INFORMATION  *    [3]
              LOPER=ALL,              AUTHORIZED LOCAL OPERATORS      *    [4]
              POPER=ALL               AUTHORIZED PARTNER OPERATORS         [5]
```

Figure 65. Sample SCP Definitions for Node 2

The comments for SCP sample 1 also apply to SCP sample 2.

Sample SCPs for Node 3

This PT sample shows SCP definitions that provide for controlling sample MERVA Link nodes 1, 2, and 4 at node 3.

```
        EKAPT TYPE=SCP,               SYSTEM CONTROL PROCESS          *    [1]
              NAME=(NODE1,C1),        PARTNER NODE NAMES              *    [2]
              DEST=(CA01,EKAC),       PARTNER SYSTEM ISC INFORMATION  *    [3]
              LOPER=ALL,              AUTHORIZED LOCAL OPERATORS      *    [4]
              POPER=ALL               AUTHORIZED PARTNER OPERATORS         [5]
*
        EKAPT TYPE=SCP,               SYSTEM CONTROL PROCESS          *    [1]
              NAME=(NODE2,C2),        PARTNER NODE NAMES              *    [2]
              DEST=(CA02,EKAC),       PARTNER SYSTEM ISC INFORMATION  *    [3]
              LOPER=ALL,              AUTHORIZED LOCAL OPERATORS      *    [4]
              POPER=ALL               AUTHORIZED PARTNER OPERATORS         [5]
*
        EKAPT TYPE=SCP,               SYSTEM CONTROL PROCESS          *    [1]
              NAME=(NODE4,I2),        PARTNER NODE NAMES              *    [2]
              DEST=(IA02,EKACI510),   PARTNER SYSTEM ISC INFORMATION  *    [3]
              LOPER=ALL,              AUTHORIZED LOCAL OPERATORS      *    [4]
              POPER=ALL               AUTHORIZED PARTNER OPERATORS         [5]
```

Figure 66. Sample SCP Definitions for Node 3

Notes:

[1] An SCP entry in the PT provides for controlling a partner MERVA Link system.

[2] The NAME parameter specifies a nickname and the MERVA Link node name for the applicable partner MERVA ESA system.

[3] The DEST parameter provides the intersystem connection parameters for the PMSC function. This sample uses APPC/MVS side information (SI) defined for the MERVA Link message transfer function. The sample

symbolic destination names are CA01, CA02, and IA02 (the same as the CICS connection definitions in SCP samples 1 and 2). The partner TP name in the sample SI profiles (EKAR) must, however, be dynamically replaced by the TP name of the PMSC function (EKAC or EKACI510).

[4] The LOPER parameter controls the authorization of local MERVA ESA operators to access the applicable partner system.

[5] The POPER parameter controls the authorization of MERVA ESA operators in the partner system to access the local MERVA ESA system.

Sample SCPs for Node 4

This PT sample shows SCP definitions that provide for controlling sample MERVA Link nodes 1 to 3 at node 4.

```
        EKAPT TYPE=SCP,              SYSTEM CONTROL PROCESS          *    [1]
              NAME=(NODE1,C1),       PARTNER NODE NAMES              *    [2]
              DEST=(CA01,EKAC),      PARTNER SYSTEM ISC INFORMATION  *    [3]
              LOPER=ALL,             AUTHORIZED LOCAL OPERATORS      *    [4]
              POPER=ALL              AUTHORIZED PARTNER OPERATORS         [5]
*
        EKAPT TYPE=SCP,              SYSTEM CONTROL PROCESS          *    [1]
              NAME=(NODE2,C2),       PARTNER NODE NAMES              *    [2]
              DEST=(CA02,EKAC),      PARTNER SYSTEM ISC INFORMATION  *    [3]
              LOPER=ALL,             AUTHORIZED LOCAL OPERATORS      *    [4]
              POPER=ALL              AUTHORIZED PARTNER OPERATORS         [5]
*
        EKAPT TYPE=SCP,              SYSTEM CONTROL PROCESS          *    [1]
              NAME=(NODE3,I1),       PARTNER NODE NAMES              *    [2]
              DEST=(MVS1,EKAC),      PARTNER SYSTEM ISC INFORMATION  *    [3]
              LOPER=ALL,             AUTHORIZED LOCAL OPERATORS      *    [4]
              POPER=ALL              AUTHORIZED PARTNER OPERATORS         [5]
```

Figure 67. Sample SCP Definitions for Node 4

The comments for SCP sample 3 also apply to SCP sample 4.

Customizing CICS for MERVA Link

A number of resources must be defined to CICS when MERVA Link is installed in the CICS environment. You must define the following types of CICS resources:

Program The MERVA Link partner table EKAPT, the MERVA Link programs, and all MERVA ESA MFS programs (MCBs and User Exits) supported by MERVA Link that are not linked to DSLMMFS must be defined to CICS.

Transaction The MERVA Link local transactions must be defined to CICS.

Profile APPC profiles may optionally be used and must then be defined to CICS.

Connection The connections to partner systems must be defined to CICS.

Session Each connection requires the definition of a set of sessions in a CICS session definition.

Partner Intersystem communication parameters provided by a CICS Partner definition can be used by MERVA Link.

Transient Data

The CICS Transient Data Facility is used to write the MERVA Link external conversation trace. The corresponding resources must be defined in CICS.

The definition of the programs, transactions, profiles, connections, and sessions are described in the following in CICS resource definition online (RDO) terms. The definition of the transient data destinations is described in CICS resource definition (macro) terms (DFHDCT macros).

Defining CICS Programs

Mandatory Resident Programs

The MERVA Link partner table EKAPT and the MERVA Link program EKAAI10 must be defined as resident Assembler programs to CICS. A sample CICS RDO screen for the definition of the MERVA Link partner table EKAPT is shown below. The MERVA Link program EKAAI10 must be defined with the same parameters.

```
  OVERTYPE TO MODIFY                                        CICS RELEASE = 0520
  CEDA  Alter
   PROGram        : EKAPT
   Group          : EKAGROUP
   DEscription  ==>
   Language     ==> Assembler          CObol | Assembler | Le370 | C | Pli
   REload       ==> No                 No | Yes
   RESident     ==> Yes                No | Yes
   USAge        ==> Normal             Normal | Transient
   USElpacopy   ==> No                 No | Yes
   Status       ==> Enabled            Enabled | Disabled
   RSl            : Public             0-24 | Public
   Cedf         ==> Yes                Yes | No
   DAtalocation ==> Any                Below | Any
   EXECkey      ==> Cics               User | Cics
:
```

Figure 68. CICS RDO Sample: Define the Program EKAPT

Mandatory Non-Resident Programs

The following MERVA Link programs and message control blocks (MCBs) must be defined as Assembler programs to CICS. These programs need not to be defined as resident.

EKAAM10 MERVA Link ASP monitor

EKAAR10 MERVA Link receiving ASP

EKAAS10 MERVA Link sending ASP

EKAAS11 MERVA Link SUBMIT.Request processor

EKAPMSC MERVA Link partner MERVA System Control Program

EKASP10 MERVA Link Message Transfer Service Processor (MTSP)

EKATM10 MERVA Link TP mirror program

EKATR10 MERVA Link receiving message transfer program

EKATS10 MERVA Link sending message transfer program

EKAACHP MERVA Link Control Facility HELP MCB

EKAHELP MERVA Link HELP MCB of the MERVA ESA help facility

EKAMCOV MERVA Link cover MCB displaying the MERVA Link control fields

A sample CICS RDO screen for the definition of the MERVA Link ASP monitor EKAAM10 is shown below. The other non-resident MERVA Link programs and MCBs must be defined with the same parameters.

```
OVERTYPE TO MODIFY                                        CICS RELEASE = 0520
CEDA  Alter
 PROGram        : EKAAM10
 Group          : EKAGROUP
 DEscription  ==>
 Language     ==> Assembler          CObol | Assembler | Le370 | C | Pli
 REload       ==> No                 No | Yes
 RESident     ==> No                 No | Yes
 USAge        ==> Normal             Normal | Transient
 USElpacopy   ==> No                 No | Yes
 Status       ==> Enabled            Enabled | Disabled
 RSl            : Public             0-24 | Public
 Cedf         ==> Yes                Yes | No
 DAtalocation ==> Any                Below | Any
 EXECkey      ==> Cics               User | Cics
⋮
```

Figure 69. CICS RDO Sample: Define the Program EKAAM10

Optional Non-Resident Programs

The following MERVA Link programs, routing tables, and message control blocks (MCBs) must be considered as MERVA Link samples. They must be defined as Assembler programs to CICS if they are used. For example, the MERVA Link installation verification scenario uses the following resources.

EKAAF10 MERVA Link sample application support filter for message authentication

EKAAF20 MERVA Link sample application support filter for message encryption

EKADEMO MERVA Link sample MCB for demonstration purposes

EKAMU010 MERVA Link sample MFS user exit

EKARTS MERVA Link sample unique routing table

EKARTSCQ MERVA Link sample control-queue routing table

The optional non-resident programs and MCBs are defined to CICS with the parameters shown in Figure 69.

Operator Message HELP MCBs

The operator message HELP MCBs must be defined as Assembler programs to CICS if the explanation of MERVA Link operator messages must be displayed online using the **show** command. The names of these MCBs are listed in Table 9 on page 174.

Table 9. MERVA Link Operator Message HELP MCBs

EKA701E	EKA718E	EKA764E	EKA777E
EKA702E	EKA719I	EKA765E	EKA778I
EKA703I	EKA720E	EKA766E	EKA779E
EKA704I	EKA721E	EKA767E	EKA780E
EKA710E	EKA722I	EKA768E	EKA781E
EKA711E	EKA730E	EKA770E	EKA782E
EKA712W	EKA731I	EKA771E	EKA783E
EKA713I	EKA732E	EKA772E	EKA784I
EKA714I	EKA760E	EKA773E	EKA790I
EKA715I	EKA761E	EKA774E	EKA791I
EKA716E	EKA762E	EKA775E	EKA792I
EKA717E	EKA763W	EKA776E	

The operator message HELP MCBs are defined to CICS with the parameters shown in Figure 69 on page 173.

Defining CICS Transactions

Mandatory Local Transactions

The following local MERVA Link transactions must be defined to CICS:

EKAC MERVA Link partner system control transaction (EKAPMSC)

EKAM MERVA Link ASP monitor transaction (EKAAM10)

EKAR MERVA Link receiving MTP transaction (EKATR10)

EKAS MERVA Link sending ASP transaction (EKAAS10)

A sample CICS RDO screen for the definition of the MERVA Link sending ASP transaction EKAS is shown below. The other mandatory transactions must be defined with the same parameters (except the PROGRAM parameter which specifies the applicable program name).

```
 OVERTYPE TO MODIFY                                          CICS RELEASE = 0520
  CEDA  Alter
   TRansaction    : EKAS
   Group          : EKAGROUP
   DEscription  ==>
   PROGram      ==> EKAAS10
   TWasize      ==> 00000             0-32767
   PROFile      ==> DFHCICST
   PArtitionset ==>
   Status       ==> Enabled           Enabled | Disabled
   PRIMedsize     : 00000             0-65520
   TASKDataloc  ==> Any               Below | Any
   TASKDATAKey  ==> Cics              User | Cics
   STOrageclear ==> No                No | Yes
   RUnaway      ==> System            System | 0-2700000
   SHutdown     ==> Disabled          Disabled | Enabled
   ISolate      ==> No                Yes | No
   Brexit       ==>
REMOTE ATTRIBUTES
   DYnamic      ==> No                No | Yes
   REMOTESystem ==>
   REMOTEName   ==>
   TRProfName   ==>
   Localq       ==>                   No | Yes
  SCHEDULING
   PRIOrity     ==> 001               0-255
   TClass       ==> No                No | 1-10
   TRANClass    ==> DFHTCL00
 ALIASES
   Alias        ==>
   TASKReq      ==>
   XTRanid      ==>
   TPName       ==>
                ==>
   XTPName      ==>
                ==>
                ==>
  RECOVERY
   DTimout      ==> No                No | 1-6800
   RESTart      ==> No                No | Yes
   SPurge       ==> No                No | Yes
   TPurge       ==> No                No | Yes
   DUmp         ==> Yes               Yes | No
   TRACe        ==> Yes               Yes | No
   COnfdata     ==> No                No | Yes
  INDOUBT ATTRIBUTES
   ACtion       ==> Backout           Backout | Commit
   WAIT         ==> Yes               Yes | No
   WAITTime     ==> 00, 00, 00        0-99 (Days,Hours,Mins)
   INdoubt        : Backout           Backout | Commit | Wait
  SECURITY
   RESSec       ==> No                No | Yes
   Cmdsec       ==> No                No | Yes
   Extsec         : No                No | Yes
   TRANsec        : 01                1-64
   Rsl            : 00                0-24 | Public


                                                              APPLID=CTS1LUNM

PF 1 HELP 2 COM 3 END            6 CRSR 7 SBH 8 SFH 9 MSG 10 SB 11 SF 12 CNCL
```

Figure 70. CICS RDO Sample: Define the Local Transaction EKAS

Defining CICS APPC Profiles

A MERVA Link MTP may be asked to specify an APPC profile when it allocates a session to a partner system. Any of these APPC profiles must be defined in CICS.

A sample CICS RDO screen for the definition of the APPC profile EKAAPPC that can be used for sessions to APPC/MVS partner systems is shown below. The MODENAME parameter must specify a valid VTAM Logon Mode Table entry.

```
OVERTYPE TO MODIFY                                      CICS RELEASE = 0520
 CEDA  Alter
  PROFile        : EKAPROF
  Group          : EKAGROUP
  DEscription  ==>
  Scrnsize     ==> Default            Default | Alternate
  Uctran       ==> No                 No | Yes
  MOdename     ==> APPCLU62
  Facilitylike ==>
  PRIntercomp  ==> No                 No | Yes
 JOURNALLING
  Journal      ==> No                 No | Yes
  MSGJrnl      ==> No                 No | INPut | Output | INOut
 PROTECTION
  MSGInteg     ==> No                 No | Yes
  Onewte       ==> No                 No | Yes
  PROtect      ==> No                 No | Yes
  Chaincontrol ==> No                 No | Yes
 PROTOCOLS
  DVsuprt      ==> All                All | Nonvtam | Vtam
  Inbfmh       ==> No                 No | All | Dip | Eods
  RAq          ==> No                 No | Yes
  Logrec       ==> No                 No | Yes
 RECOVERY
  Nepclass     ==> 000                0-255
  RTimout      ==> No                 No | 1-7000


                                                          APPLID=CTS1LUNM

PF 1 HELP 2 COM 3 END            6 CRSR 7 SBH 8 SFH 9 MSG 10 SB 11 SF 12 CNCL
```

Figure 71. CICS RDO Sample: Define the APPC Profile EKAPROF

Defining CICS Connections and Sessions

Each connection to a partner system must be defined in CICS in an intersystem connection definition. Each connection definition is associated with a set of sessions. The parameters of the connection and session definitions depend on the type of the partner system (CICS, APPC/MVS, IMS, or workstation).

If you use CICS APPC services for intersystem communication, you must specify ISC=YES either in the DFHSIT or as a CICS startup parameter. The CICS intersystem communication support is described in the *CICS/ESA Intercommunication Guide*.

APPC Connections to Another CICS

A sample CICS RDO screen for the definition of an APPC connection to another CICS system (CTS2LUNM) is shown below. The definition of the APPC sessions for this connection is shown below.

```
OVERTYPE TO MODIFY                                        CICS RELEASE = 0520
CEDA  Alter
 Connection     : CA02
 Group          : EKAISC
 DEscription  ==>
CONNECTION IDENTIFIERS
 Netname      ==> CTS2LUNM
 INDsys       ==>
REMOTE ATTRIBUTES
 REMOTESystem ==>
 REMOTEName   ==>
 REMOTESYSNet ==>
CONNECTION PROPERTIES
 ACcessmethod ==> Vtam              Vtam | IRc | INdirect | Xm
 Protocol     ==> Appc              Appc | Lu61
 Conntype     ==>                   Generic | Specific
 SInglesess   ==> No                No | Yes
 DAtastream   ==> User              User | 3270 | SCs | STrfield | Lms
 RECordformat ==> U                 U | Vb
 Queuelimit   ==> No                No | 0-9999
 Maxqtime     ==> No                No | 0-9999
OPERATIONAL PROPERTIES
 AUtoconnect  ==> Yes               No | Yes | All
 INService    ==> Yes               Yes | No
SECURITY
 SEcurityname ==>
 ATtachsec    ==> Local             Local | Identify | Verify | Persistent
                                    | Mixidpe
 BINDPassword ==>                   PASSWORD NOT SPECIFIED
 BINDSecurity ==> No                No | Yes
 Usedfltuser  ==> No                No | Yes
RECOVERY
 PSrecovery   ==> Sysdefault        Sysdefault | None
 Xlnaction    ==> Keep              Keep | Force


                                                          APPLID=CTS1LUNM

PF 1 HELP 2 COM 3 END            6 CRSR 7 SBH 8 SFH 9 MSG 10 SB 11 SF 12 CNCL
```

Figure 72. CICS RDO Sample: Define an APPC Connection to CTS2LUNM

```
OVERTYPE TO MODIFY                                        CICS RELEASE = 0520
 CEDA  Alter
  Sessions       : CA02S
  Group          : EKAISC
  DEscription  ==>
 SESSION IDENTIFIERS
  Connection   ==> CA02
  SESSName     ==>
  NETnameq     ==>
  MOdename     ==> APPCLU62
 SESSION PROPERTIES
  Protocol     ==> Appc              Appc | Lu61 | Exci
  MAximum      ==> 004 , 002         0-999
  RECEIVEPfx   ==>
  RECEIVECount ==>                   1-999
  SENDPfx      ==>
  SENDCount    ==>                   1-999
  SENDSize     ==> 30720             1-30720
  RECEIVESize  ==> 30720             1-30720
  SESSPriority ==> 000               0-255
  Transaction    :
 OPERATOR DEFAULTS
  OPERId         :
  OPERPriority   : 000               0-255
  OPERRsl        : 0                                              0-24,...
  OPERSecurity   : 1                                              1-64,...
 PRESET SECURITY
  USERId       ==>
 OPERATIONAL PROPERTIES
  AUtoconnect  ==> Yes               No | Yes | All
  INService    ==>                   Yes | No
  Buildchain   ==> Yes               Yes | No
  USERArealen  ==> 000               0-255
  IOarealen    ==> 0000 , 0000       0-32767
  RELreq       ==> No                No | Yes
  DIscreq      ==> No                No | Yes
  NEPclass     ==> 000               0-255
 RECOVERY
  RECOVOption  ==> Sysdefault        Sysdefault | Clearconv | Releasesess
                                     | Uncondrel | None
  RECOVNotify  ==> None              None | Message | Transaction

                                                          APPLID=CTS1LUNM

PF 1 HELP 2 COM 3 END            6 CRSR 7 SBH 8 SFH 9 MSG 10 SB 11 SF 12 CNCL
```

Figure 73. CICS RDO Sample: Define APPC Sessions for CA02

If the APPC sessions are to be automatically acquired by CICS when CICS is started, `AUtoconnect ==> Yes` must be specified in both the connection and session definitions. Otherwise, the APPC sessions must be automatically acquired by other means.

The CICS master operator can acquire the APPC sessions by issuing one of the following commands when both CICS systems are active:

```
CEMT SET CONN(CA02) INS ACQ
CEMT SET CONN ALL INS ACQ
```

A sending MERVA Link CICS MTP can be asked to acquire a session dynamically before it reports a session allocation failure. `AUtoconnect ==> No` can be specified in this case.

The session definition parameters SENDSIZE 30720 and RECEIVESIZE 30720 provide an optimum performance if MERVA ESA large messages are to be

transferred using this connection. You can, however, use smaller values if your SNA network is not capable of handling request units of this size.

APPC Connections to APPC/MVS or APPC/IMS

A connection to a remote APPC/MVS system or to a remote APPC/IMS system is defined in the same way as a connection to another CICS system (shown in Figure 72 on page 177 and in Figure 73 on page 178).

For a connection to an APPC/MVS system, you must specify the name of any APPC/MVS LU that is associated with the APPC/MVS transaction scheduler ASCH in the Netname parameter of the connection definition. The sample APPC/MVS LU name is MVS1LUNM.

For a connection to an APPC/IMS system, you must specify the name of the APPC/IMS Base LU that is associated with the APPC/IMS transaction scheduler in the NETNAME parameter. The sample APPC/IMS LU name is IA02LUNM.

Parameter modifications for the connection to an APPC/MVS system are shown in Figure 74 and in Figure 75.

```
OVERTYPE TO MODIFY                                      CICS RELEASE = 0520
CEDA  Alter
 Connection     : MVS1
 Group          : EKAISC
 DEscription  ==>
CONNECTION IDENTIFIERS
 Netname      ==> MVS1LUNM
 INDsys       ==>
```

Figure 74. CICS RDO Sample: Define an APPC Connection to MVS1LUNM

```
OVERTYPE TO MODIFY                                      CICS RELEASE = 0520
CEDA  Alter
 Sessions       : MVS1S
 Group          : EKAISC
 DEscription  ==>
SESSION IDENTIFIERS
 Connection   ==> MVS1
 SESSName     ==>
 NETnameq     ==>
 MOdename     ==> APPCLU62
SESSION PROPERTIES
 Protocol     ==> Appc              Appc | Lu61 | Exci
 MAximum      ==> 004 , 002         0-999
 RECEIVEPfx   ==>
 RECEIVECount ==>                   1-999
 SENDPfx      ==>
 SENDCount    ==>                   1-999
 SENDSize     ==> 30720             1-30720
 RECEIVESize  ==> 30720             1-30720
 SESSPriority ==> 000               0-255
 Transaction    :
```

Figure 75. CICS RDO Sample: Define APPC Sessions for MVS1

Defining CICS Partners

A MERVA Link MTP can refer to a CICS PARTNER definition that provides information about a specific partner MTP. If more than 4 characters are specified by the MTP as the partner system identifier, this identifier is interpreted as a CICS PARTNER name.

A CICS PARTNER definition (that can be compared to an APPC/MVS side information profile) contains the parameters:

- Netname of the partner LU
- Session profile name
- Partner TP name

The corresponding MTP parameters need not be specified if the MTP refers to a CICS partner definition.

A sample CICS RDO screen for the definition of the Partner GATEWAY9 is shown below.

```
OVERTYPE TO MODIFY                                        CICS RELEASE = 0520
 CEDA  Alter PARTNer ( GATEWAY9 )
  PARTNer        : GATEWAY9
  Group          : EKAISC
  DEscription  ==>
 REMOTE LU NAME
  NETName      ==> MVS9LUNM
  NETWork      ==>
 SESSION PROPERTIES
  Profile      ==> EKAPROF
 REMOTE TP NAME
  Tpname       ==> EKAR1
               ==>
  Xtpname      ==>
               ==>
               ==>


                                                        APPLID=CTS2LUNM

PF 1 HELP 2 COM 3 END            6 CRSR 7 SBH 8 SFH 9 MSG 10 SB 11 SF 12 CNCL
```

Figure 76. CICS RDO Sample: Define Partner GATEWAY9

Defining CICS Transient Data Destinations

The CICS Transient Data facility is used by MERVA Link CICS to support the MERVA Link External Conversation Trace. An external conversation trace is written by a MERVA Link MTP to a CICS transient data queue if it is asked to do so by parameters in the partner table. The sample TD queue identifier for the conversation trace is EKAT.

A CICS transient data definition for the MERVA Link conversation trace can be provided either in DFHDCT statements or via RDO.

Defining Extrapartition Transient Data Queue (DFHDCT)

The sample destination of the conversation trace (EKAT) is defined as an extrapartition transient data queue in CICS. The corresponding entries are shown in Figure 77 on page 181.

Note: There are specific CICS rules for the location of a DFHDCT TYPE=SDSCI entry in your DCT generation source code. For more information refer to *CICS Resource Definition (Macro).*

```
*-------------------------------------------------------------------*
*       MERVA LINK CONVERSATION TRACE
*-------------------------------------------------------------------*
        DFHDCT TYPE=SDSCI,                                          *
               DSCNAME=EKACTRC,                                     *
               BLKSIZE=6144,                                        *
               TYPEFLE=OUTPUT,                                      *
               RECFORM=VARBLK,                                      *
               RECSIZE=4096
        DFHDCT TYPE=EXTRA,                                          *
               DESTID=EKAT,                                         *
               DSCNAME=EKACTRC
```

Figure 77. Define DFHDCT Entries for the Conversation Trace

Defining Extrapartition Transient Data Queue (RDO)

The sample destination of the conversation trace (EKAT) is defined as an extrapartition transient data queue in CICS. A sample CICS RDO screen for the definition of this TD queue is shown below.

```
OVERTYPE TO MODIFY                                      CICS RELEASE = 0520
 CEDA  ALter TDqueue( EKAT )
  TDqueue        : EKAT
  Group          : EKAISC
  DEscription  ==>
  TYPE         ==> Extra             Extra | INTra | INDirect
 EXTRA PARTITION PARAMETERS
  DAtabuffers  ==> 001               1-255
  DDname       ==> EKACTRC
  DSname       ==>
  Sysoutclass  ==>
  Erroroption  ==> Ignore            Ignore | Skip
  Opentime     ==> Initial           Initial| Deferred
  REWind       ==>                   Leave | Reread
  TYPEFile     ==> Output            Input | Output | Rdback
  RECORDSize   ==> 04096             1-32767
  BLOCKSize    ==> 06144             1-32767
  RECORDFormat ==> Variable          Fixed | Variable
  BLOCKFormat  ==> Blocked           Blocked | Unblocked
  Printcontrol ==>                   A | M
  DIsposition  ==> Shr               Shr | Old | Mod
 INTRA PARTITION PARAMETERS
  .
  .
  .
                                                        APPLID=CTS1LUNM

PF 1 HELP 2 COM 3 END           6 CRSR 7 SBH 8 SFH 9 MSG 10 SB 11 SF 12 CNCL
```

Figure 78. CICS RDO Sample: Define a TD Queue for the Conversation Trace

Allocating MERVA Link Conversation Trace Data Set

The MERVA Link external conversation trace is written to a sequential data set when this service is requested.

This sequential data set must be allocated before the conversation trace service can be used. The sample MERVA Link conversation trace data set used in the CICS environment can be allocated with the following parameters:

```
DATA SET NAME:        EKA.CTS120.CTRC1
Organization:         PS
Record Format:        VB
Record Length:        4096
Block Size:           6144
First Extent Tracks   5
Secondary Tracks      5
```

Customizing the CICS Startup Job

Depending on your installation and customization decisions, you must add the definition of the MERVA Link conversation trace data set to the CICS startup job control statements. In addition, you can add a job step to your CICS startup job that prints the conversation trace data set after CICS shutdown.

The MERVA Link sample CICS startup job statements for the MVS environment are shown in Figure 79. The MERVA Link sample CICS startup job statements to print the conversation trace data set in the VSE environment are shown in Figure 80.

```
//*-------------------------------------------------------------------*
//*         MERVA LINK CONVERSATION TRACE DATA SET
//*-------------------------------------------------------------------*
//EKACTRC   DD DSN=EKA.CTS120.CTRC1,DISP=SHR

//*-------------------------------------------------------------------*
//*         ADDITIONAL JOB STEP: PRINT CONVERSATION TRACE
//*-------------------------------------------------------------------*
//LSTCTRC   EXEC PGM=IDCAMS,PARM=GRAPHICS(CHAIN(TN))
//EKACTRC   DD DSN=EKA.CTS120.CTRC1,DISP=SHR
//SYSPRINT  DD SYSOUT=*
//SYSIN     DD *
            PRINT INFILE(EKACTRC)
/*
```

Figure 79. Sample Additional CICS TS Startup MVS JCL Statements

```
// *------------------------------------------------------------------*
// *        ADDITIONAL JOB STEP: PRINT CONVERSATION TRACE
// *------------------------------------------------------------------*
// ASSGN SYS025,DISK,VOL=volser,SHR
// DLBL SDSKIN,'EKA.CTS120.CTRC1',0,SD
// EXTENT SYS025,volser,1,0,xxxx,yy
// UPSI 1
// EXEC DITTO,SIZE=512K
$$DITTO SET DUMP=ACROSS
$$DITTO SDP INPUT=SYS025,MODE=V
$$DITTO EOJ
/&
```

Figure 80. Sample Additional CICS/VSE Startup JCL Statements

Customizing IMS for MERVA Link

MERVA Link IMS requires a set of resources to be built and defined to the IMS system.

IMS PSB and ACB

The MERVA Link program EKAAS10 needs an IMS PSB to be defined and assembled, and the corresponding ACB to be built. The PSB definition is shown below.

```
*-------------------------------------------------------------------*
*      MERVA LINK SENDING ASP                                       *
*-------------------------------------------------------------------*
       PCB     TYPE=TP,MODIFY=YES            MODIFY=YES IS MANDATORY
       PSBGEN  PSBNAME=EKAAS10,LANG=ASSEM
       END
```

Figure 81. IMS PSB Definition

Do not change the default parameters of the PCB macros:

- ALTRESP=NO
- SAMETRM=NO
- EXPRESS=NO

The input for the required ACB generation is:

```
  BUILD        PSB=EKAAS10
```

These definitions are part of the MERVA ESA installation jobs DSLEIPSB and DSLEIACB.

The receiving message transfer program EKATPI1, which supports both APPC/MVS and APPC/IMS, need not be defined as an IMS PSB.

Defining IMS Applications

The PSB for EKAAS10 must be defined to IMS as an application together with the transaction codes used to run this program. You must define a unique transaction code for program EKAAS10 for every sending ASP.

The definitions must be included in the stage 1 input code of the IMS generation. The definitions are part of the MERVA ESA IMS application definition copy book DSLIMSAP.

Sample application and transaction definitions are shown in Figure 82. The five transaction codes for the program (PSB) EKAAS10 allow five ASPs to be defined in the partner table.

```
*--------------------------------------------------------------------
*  MERVA LINK IMS APPLICATION DEFINITIONS
*--------------------------------------------------------------------
*
       APPLCTN PSB=EKAAS10,            SENDING ASP                  *
               SCHDTYP=PARALLEL
      TRANSACT CODE=(EKAS,EKAS1,EKAS2,EKAS3,EKAS4),                 *
               MODE=SNGL,                                           *
               MSGTYPE=(SNGLSEG,,24),                               *
               PARLIM=1,MAXRGN=1
```

Figure 82. IMS Application Definitions

The parameters PARLIM=1 and MAXRGN=1 in the TRANSACT macro for the sending ASP are required. These parameters ensure that a sending ASP does not run in two message processing regions at the same time.

The transaction code for the receiving MTP EKATPI1 supporting APPC/MVS is defined in an APPC/MVS TP profile. It does need not to be defined in an IMS application definition.

Customizing the IMS Message Processing Region Startup Job

The startup job for the message processing regions that run the MERVA Link programs must be extended by a data definition statement for the MERVA Link SNAP dumps. An example of such a statement is:

```
//EKASNAP   DD SYSOUT=*
```

Customizing APPC/MVS for MERVA Link

An APPC/MVS TP profile for the APPC/MVS transaction scheduler ASCH must be defined for MERVA Link IMS APPC connections if the receiving transaction must run in an APPC/MVS initiator. A TP profile describes a MERVA Link IMS receiving process.

You can specify the intersystem parameters needed for a MERVA Link sending process in the PT. Therefore, you do not need to specify APPC/MVS Side Information in the MERVA Link APPC/MVS environment. Alternatively, the intersystem connection parameters in the PT can refer to APPC/MVS Side Information. In this case, the TP name in an SI profile can be dynamically overwritten by a PT parameter.

Details of the APPC/MVS Intersystem Communication Support are described in *MVS/ESA Planning: APPC Management*. Refer to this manual for the definition of APPC/MVS intersystem communication resources.

APPC/MVS TP Profile for the APPC/MVS Scheduler

An APPC/MVS TP profile is used by the APPC/MVS task scheduler to start an inbound transaction program (TP). A TP profile is defined for access by APPC/MVS in the APPC/MVS TPADD command. A TP profile is identified by the TP name (parameter TPNAME in the APPC/MVS TPADD command). The most important information in a TP profile are job control statements used to run the inbound TP in an APPC/MVS initiator (a specific MVS region).

A sample TP profile definition for the MERVA Link inbound TP EKATPI1 is shown below. The MERVA Link program EKATPI1 runs in an APPC/MVS initiator. It is the MERVA Link receiving MTP that supports APPC/MVS in the MERVA Link IMS environment. The APPC/MVS TP profile of the MERVA Link receiving MTP can be shared by receiving processes communicating with sending processes in all partner systems (CICS, APPC/MVS, and workstations).

```
//jobname  JOB (....,.),'programmer',
//          MSGLEVEL=(1,1)
//*****************************************************************
//* DEFINE A SYSTEM-LEVEL STANDARD TP-PROFILE
//*****************************************************************
//DEFTP    EXEC PGM=ATBSDFMU
//SYSPRINT DD   SYSOUT=*
//SYSSDOUT DD   SYSOUT=*
//SYSSDLIB DD   DSN=SYS1.APPCTP,DISP=SHR
//SYSIN    DD   DATA,DLM=XX
     TPDELETE
          TPNAME(EKAR)
          SYSTEM
     TPADD
          TPNAME(EKAR)
          SYSTEM
          ACTIVE(YES)
          TPSCHED_DELIMITER(##)
          TPSCHED_TYPE(STANDARD)
          JCL_DELIMITER(END_OF_JCL)
//jobname  JOB (....,.),'programmer',MSGCLASS=X,
//          MSGLEVEL=(1,1)
//EKAR    EXEC PGM=EKATPI1
//STEPLIB   DD DSN=merva.SDSLLODI,DISP=SHR
//          DD DSN=merva.SDSLLODB,DISP=SHR
//SYSUDUMP  DD SYSOUT=X
//DSLSNAP   DD SYSOUT=X
//EKASNAP   DD SYSOUT=X
END_OF_JCL
##
XX
```

Figure 83. APPC/MVS EKATPI1 TP Profile Definition Sample

A sample TP profile definition for the inbound TP EKAPMSC is shown below. The MERVA Link program EKAPMSC runs in an APPC/MVS initiator. It is the MERVA System Control Facility server program that supports APPC/MVS in the MERVA Link IMS environment.

```
⋮

//SYSIN    DD   DATA,DLM=XX
     TPDELETE
          TPNAME(EKAC)
          SYSTEM
     TPADD
          TPNAME(EKAC)
          SYSTEM
          ACTIVE(YES)
          TPSCHED_DELIMITER(##)
          TPSCHED_TYPE(STANDARD)
          JCL_DELIMITER(END_OF_JCL)
//jobname  JOB (....,.),'programmer',MSGCLASS=X,
//           MSGLEVEL=(1,1)
//EKAR     EXEC PGM=EKAPMSC
//STEPLIB    DD DSN=merva.SDSLLODI,DISP=SHR
//           DD DSN=merva.SDSLLODB,DISP=SHR
//SYSUDUMP   DD SYSOUT=X
//DSLSNAP    DD SYSOUT=X
END_OF_JCL
##
XX
```

Figure 84. APPC/MVS EKAPMSC TP Profile Definition Sample

APPC/MVS SI Profile

An APPC/MVS SI profile defines a Symbolic Destination for a receiving process in a partner system. The parameters of an SI profile are the TP name, the APPC session mode name, and the partner LU name.

A sample SI profile that can be used in sample node 3 to connect to sample node 1 (CICS 1) is shown below.

```
⋮

//DEFTP    EXEC PGM=ATBSDFMU
//SYSPRINT DD   SYSOUT=*
//SYSSDOUT DD   SYSOUT=*
//SYSSDLIB DD   DSN=SYS1.APPCSI,DISP=SHR
//SYSIN    DD   *
     SIDELETE
          DESTNAME(CA01)
     SIADD
          DESTNAME(CA01)
          TPNAME(EKAR)
          MODENAME(APPCLU62)
          PARTNER_LU(CTS1LUNM)
/*
```

Figure 85. APPC/MVS CA01 SI Profile Definition Sample

The sample SI profile definition used in sample node 3 to connect to the MERVA Link USS Gateway (sample node 9) is shown below.

```
⋮

//DEFTP    EXEC PGM=ATBSDFMU
//SYSPRINT DD   SYSOUT=*
//SYSSDOUT DD   SYSOUT=*
//SYSSDLIB DD   DSN=SYS1.APPCSI,DISP=SHR
//SYSIN    DD   *
     SIDELETE
          DESTNAME(MVS9)
     SIADD
          DESTNAME(MVS9)
          TPNAME(EKAR1)
          MODENAME(APPCLU62)
          PARTNER_LU(MVS9LUNM)
/*
```

Figure 86. APPC/MVS MVS9 SI Profile Definition Sample

APPC/MVS Inbound TP Security Considerations

An APPC/MVS inbound TP runs in an APPC/MVS initiator in a similar way as an MVS batch job. A TP security environment on MVS is established by APPC/MVS. This security environment depends on the security type of the inbound allocate request and the security data contained in this request.

The security type is specified and the security information can be provided by the partner system. Security types defined in LU 6.2 are NONE, SAME, and PROGRAM. The APPC partner systems are CICS, APPC/MVS, APPC/IMS, and Communications Server on a workstation.

CICS Outbound Security in an Allocate Request

CICS supports security type SAME as a fix parameter in the mapping of CICS APPC commands to LU 6.2 verbs (see CICS Intercommunication Guide). A CICS application program has no option to modify this parameter.

The user ID of the CICS security environment is passed to APPC/MVS in the allocate request (FMH5) with an indicator that the user ID has already been verified. The APPC/MVS inbound TP can access any data or resources that this user is allowed to access.

The APPC/MVS LU must be told to accept an already verified user ID by the parameter **SECACPT=ALREADYV** in the VTAM definition of the APPC/MVS LU.

The user ID in CICS: A CICS task, which is started by a CICS START command rather than by an operator at a terminal, inherits the user ID from the task that issued the START command. This applies also for a sequence of tasks.

The first task in this chain must, however, be started either by an operator from a terminal after a successful SIGNON, or through definitions in the CICS PLT during CICS system startup. In case of an operator start, the operator identifier, the operator password, and the user identifier are defined in an entry of the CICS Signon Table (DFHSNT). In case of the automatic start during CICS startup, CICS itself is the 'operator' and the CICS system identifier specified in the SYSIDNT parameter of the CICS SIT is the user identifier associated with the started task.

If the first task is started by an operator at a terminal without sign on, no user identifier is associated with the sequence of started tasks.

What this means in the MERVA Link environment: In the MERVA ESA CICS environment, the sequence of tasks is DSL (MERVA ESA startup), DSLN (MERVA ESA Nucleus task providing the central MERVA ESA services), and EKAS (MERVA Link sending task). Task EKAS is associated with the user ID of task DSLN, and task DSLN is associated with the user ID of task DSL. The user ID of the operator who enters DSL at a terminal after a successful signon is the user ID which is passed to APPC/MVS to setup the inbound TP security environment.

If the MERVA ESA nucleus is started by program DSLCAS specified in the CICS PLT, the CICS system identifier specified in the SYSIDNT parameter of the CICS SIT (default system identifier is 'CICS') is associated as user ID with the tasks DSLN and EKAS, and it is passed to APPC/MVS.

This user ID must be authorized in the MVS system housing APPC/MVS to access the resources used by the inbound TP. These resources, defined in the TP profile, are, for example, the MERVA ESA program libraries.

How a security violation is reported in MERVA Link CICS: If task DSL is started by an operator who has not signed on, no user ID is passed to APPC/MVS. Unless universal access has been defined for the inbound TP resources, a security violation will be detected and reported to the sending system. It appears as a remote process abend (RP ABE) in MERVA Link CICS.

If task DSL is started by a signed on operator or using the PLT, a user ID is passed to APPC/MVS. If this user ID is not defined in the MVS system housing APPC/MVS or has insufficient access authority, again, a security violation will be detected and reported to the CICS system.

CICS/MVS and CICS/ESA handle that report of a security violation differently.

CICS/MVS abends the MERVA Link sending task EKAS in this case with abend code AISS. This CICS task abend is intercepted by MERVA Link and appears as a local process abend (LP ABE) in MERVA Link.

CICS/ESA reports the security violation as a terminal error with an error code of X'080F6051' in EIBERRCD. This CICS error code is translated by MERVA Link and appears as 'security not valid' (SEC NV) in MERVA Link.

IMS Outbound Security in an Allocate Request

The outbound allocate request is issued by the MERVA Link sending MTP named EKATPO1 in the IMS APPC environment. EKATPO1, which runs in an IMS MPP, specifies security type SAME as a fix parameter of the outbound allocate request. An option to modify this parameter is not provided.

The user ID of the IMS MPP security environment is passed to APPC/MVS in the allocate request. The APPC/MVS inbound TP can access any data or resources that this user is allowed to access.

How a security violation is reported in MERVA Link IMS: APPC/MVS reports the security violation with ATB_Return_Code 6 (Security_not_valid). This return code is translated by MERVA Link and appears as 'security not valid' (SEC NV) in MERVA Link.

EKAPMSC Security Considerations

EKAPMSC is the inbound TP of the MERVA System Control Facility. If EKAPMSC runs in the APPC/MVS environment, the outbound TP (EKAEMSC within the MERVA EUD program) must provide security information. In the IMS environment, this security information is provided by the security environment of the IMS MPP that houses the EUD transaction.

In the CICS environment, this security information is provided by the security environment of the terminal used for communication with the MERVA EUD. In CICS/MVS you must sign on at the terminal before you can operate a partner MERVA system using your local MERVA EUD. And you must be authorized in the partner MVS system to access the required resources.

In CICS/ESA the CICS system administrator can define a default user which becomes applicable if you do not sign on at your terminal. This user must be authorized in the partner MVS system to access the required resources. If no default user is defined, a signon is required as for CICS/MVS, and you must be authorized in the partner MVS system to access the required resources.

How a security violation is reported: In the CICS environment, the MSC front end transaction program checks whether a user ID is associated with the EUD task. Operator message EKA767E asks you to sign on if no user ID is available.

In the IMS environment, a user ID is always available. The IMS MPP security environment applies in this case.

If the user is, however, not authorized by the partner system, the command transfer request is rejected at the APPC level, and an 'invalid security information' error is reported. The 'security not valid' error code is translated by MERVA Link and appears as SEC NVAL in the MERVA System Control Facility screen AC03.

Connecting Trusted and Untrusted Partner Systems

A data communication system which employs an access control facility (for example, RACF) that is operated by a security administrator is considered to be a trusted system. A user identifier received from a trusted system has already been verified by that system, and need not be verified again by the local APPC/MVS system. CICS and APPC/MVS, for example, are considered to be trusted systems.

Workstations are considered to be untrusted systems because they can be under the full control of one person.

Some options to connect an APPC/MVS system to trusted partner systems, to untrusted partner systems, and to both, trusted and untrusted partner systems, for inbound conversations are discussed in the following sections. APPC/MVS outbound conversations are always handled by the APPC/MVS Base LU. The SECACPT parameter is irrelevant for outbound conversations.

For more details about these options, refer to *MERVA for ESA Advanced MERVA Link* and RACF documentation.

Connecting Trusted Partner Systems

A trusted partner system sends an already verified user ID at the beginning of an outbound conversation. An APPC/MVS LU accepts inbound conversations with the AV indicator if it is defined with SECACPT=ALREADYV. Therefore, if you define the APPC/MVS Base LU with SECACPT=ALREADYV, you can connect trusted partner systems to this LU for outbound and inbound conversations.

A single CICS CONNECTION definition is required if both, the outbound and the inbound conversations to and from CICS use the APPC/MVS Base LU. If the APPC/MVS Base LU cannot be used by CICS for CICS outbound conversations, a second CONNECTION to APPC/MVS must be defined in CICS. It must specify the name of another APPC/MVS LU. The system identifier of that connection must be specified in the applicable MTP entry of the MERVA Link partner table in the MERVA ESA CICS system.

Connecting Untrusted Partner Systems

An untrusted partner system sends a user ID and password at the beginning of an outbound conversation. An APPC/MVS LU accepts inbound conversations with user ID and password if it is defined with SECACPT=CONV. It rejects inbound conversations with the AV indicator in this case. You can connect untrusted partner systems to this LU for outbound and inbound conversations.

Connecting Both Trusted and Untrusted Partner Systems

There are two major options with VTAM to support both trusted and untrusted partner systems. The first option uses multiple APPC/MVS LUs, the second option uses a single APPC/MVS LU, the APPC/MVS Base LU, with temporary modification of the value specified in the SECACPT keyword of the VTAM APPL statement.

Using multiple APPC/MVS LUs: If you choose the first option, you must define the APPC/MVS Base LU with SECACPT=CONV and another APPC/MVS LU with SECACPT=ALREADYV. Untrusted partner systems are connected for outbound and inbound conversations to the Base LU. Trusted partner systems are connected for APPC/MVS outbound conversations to the APPC/MVS Base LU, the only option supported by MERVA Link.

For APPC/MVS inbound conversations, the trusted partner systems are connected to the APPC/MVS LU defined with SECACPT=ALREADYV. This is achieved by appropriate definitions in the (sending) partner system (CICS or APPC/MVS).

- In a partner MERVA ESA CICS system another CICS CONNECTION must be defined for that APPC/MVS LU, and that connection must be specified in the MTP entry of the MERVA Link PT for the local APPC/MVS system.
- In a partner MERVA ESA IMS (APPC/MVS) system you must specify that APPC/MVS LU in the MTP entry of the MERVA Link PT for the local APPC/MVS system.

This option may require the definition of RACF profiles of the classes APPL or APPCPORT to prevent an untrusted partner system from connecting to the APPC/MVS LU defined with SECACPT=ALREADYV.

Using a single APPC/MVS LU, the Base LU: If you choose the second option, you must define the APPC/MVS Base LU with SECACPT=CONV and VERIFY=OPTIONAL, and ask for dynamic modification of this parameter for all connections to trusted partner systems. RACF profiles of the class APPCLU are used for that purpose. The definition of RACF profiles of the classes APPL or APPCPORT may not be necessary if you choose this option.

Customizing APPC/IMS for MERVA Link

An APPC/MVS TP profile for the APPC/IMS transaction scheduler must be defined for MERVA Link IMS APPC connections if the receiving transaction must run in an IMS Message Processing Region (MPR). A TP profile describes a MERVA Link IMS receiving process.

All identifiers of the partner process that are required in a MERVA Link sending process are specified in the PT. This is why APPC/MVS Side Information is not applicable in the MERVA Link APPC/IMS environment.

Details of the APPC/IMS Intersystem Communication Support are described in:

- *OS/390 MVS Planning: APPC/MVS Management*
- *IMS/ESA Administration Guide: Transaction Manager*

Refer to these manuals for the definition of APPC/IMS intersystem communication resources.

APPC/MVS TP Profile for the APPC/IMS Scheduler

An APPC/MVS TP profile is used by the APPC/IMS scheduler to start an inbound transaction program (TP) in an IMS MPR. A TP profile is defined for access by APPC/IMS in the APPC/MVS TPADD command. A TP profile is identified by the TP name (parameter TPNAME in the APPC/MVS TPADD command). The most important information in an APPC/IMS TP profile is the name of the transaction program and the message class that specifies the IMS MPRs that can run the TP.

A sample TP profile definition for the MERVA Link inbound TP EKATPI1 is shown below.

```
//jobname  JOB (....,.),'programmer',
//          MSGLEVEL=(1,1)
//*****************************************************************
//* DEFINE A SYSTEM-LEVEL STANDARD TP-PROFILE
//*****************************************************************
//DEFTP    EXEC PGM=ATBSDFMU
//SYSPRINT DD   SYSOUT=*
//SYSSDOUT DD   SYSOUT=*
//SYSSDLIB DD   DSN=SYS1.APPCTP,DISP=SHR
//SYSIN    DD   DATA,DLM=XX
     TPDELETE
          TPNAME(EKARI510)
          SYSTEM
     TPADD TPSCHED_EXIT(DFSTPPE0)
          TPNAME(EKARI510)
          SYSTEM
          ACTIVE(YES)
          TPSCHED_DELIMITER(##)
            TRANCODE=EKATPI1
            CLASS=24
            MAXRGN=1
            RACF=FULL
            ##
XX
```

Figure 87. APPC/MVS EKATPI1 TP Profile Definition Sample

The MERVA Link program EKATPI1 runs in an IMS MPR that supports message class 24. EKATPI1 is the MERVA Link receiving Message Transfer Program that supports APPC/MVS and APPC/IMS in the MERVA Link IMS environment. The APPC/MVS TP profile of the MERVA Link receiving TP can be shared by receiving processes communicating with sending processes in all partner systems (CICS, APPC/MVS, and workstations).

A sample TP profile definition for the inbound TP EKAPMSC is shown below.

```
⋮

//SYSIN    DD   DATA,DLM=XX
     TPDELETE
          TPNAME(EKACI510)
          SYSTEM
     TPADD TPSCHED_EXIT(DFSTPPE0)
          TPNAME(EKACI510)
          SYSTEM
          ACTIVE(YES)
          TPSCHED_DELIMITER(##)
            TRANCODE=EKAPMSC
            CLASS=23
            MAXRGN=1
            RACF=FULL
            ##
XX
```

Figure 88. APPC/MVS EKAPMSC TP Profile Definition Sample

The MERVA Link program EKAPMSC runs in an IMS MPR that supports message class 23. EKAPMSC is the MERVA System Control Facility server program that supports APPC/MVS and APPC/IMS in the MERVA Link IMS environment.

APPC/IMS Inbound TP Security Considerations

An APPC/IMS inbound TP runs in an IMS Message Processing Region (MPR). The TP security environment has been established by the MPR startup job. The security information in the inbound FMH5 is not used by APPC/IMS to build the inbound TP security environment.

The access of an inbound TP by a remote user can be controlled by RACF definitions (resource class APPCTP).

APPC/IMS Inbound TP Scheduling Considerations

APPC/MVS schedules inbound TPs in one of its initiators (MVS address spaces controlled by APPC/MVS). APPC/IMS schedules inbound TPs in one of its Message Processing Regions (MPRs). The following considerations apply to APPC/IMS because of its TP scheduling technique.

IMS MPR Availability

An IMS message class is specified in the TP scheduling data parameter *CLASS* of an APPC/IMS TP profile. An IMS MPR supporting this message class must be active to serve an inbound request for that TP.

If there is no IMS MPR active for that message class, APPC/MVS keeps the inbound conversation request in a message queue. The client application (outbound conversation) in the partner system enters a wait state until an appropriate IMS MPR is started and the queued request is processed. A timeout is not applicable in this situation. This means that the sending application may wait a long time for the response from the partner system.

Partner MSC Response Time

EKAPMSC can return its command response much faster if it is scheduled in an IMS MPR rather than in an APPC/MVS initiator. An IMS MPR has the majority of the resources required by EKAPMSC already allocated when the command server transaction is scheduled.

All resources must be allocated by APPC/MVS for each command server transaction if EKAPMSC runs in an APPC/MVS initiator. This is why you must wait for the command response significantly longer in this environment.

Customizing a Synchronous Back-to-Back Test Environment

The MERVA Link Back-to-Back (BTB) Test Environment provides a means to run a sending ASP and its partner receiving ASP in the same CICS or IMS system. User-written MERVA ESA MFS User Exits and application support filters can be used in this environment. BTB provides a convenient environment to test user exits and ASFs. The MERVA Link Installation Verification, for example, is run in the MERVA Link Back-to-Back Test Environment.

Synchronous TP Mirror EKATM10

The MERVA Link Back-to-Back Test Environment is based on the MERVA Link TP mirror program EKATM10. All sending and receiving MERVA Link functions are run in the same MERVA Link transaction (the sending MERVA Link task) when the TP Mirror is called at the MERVA Link TP layer.

A MERVA Link sending transaction that is started to handle the outbound messages of an ASP in the Back-to-Back Test Environment handles all messages in the send queue cluster in one transaction. The sending and receiving functions are run sequentially for each message. Therefore, the transfer rate in a TP Mirror environment can be smaller than the transfer rate in a real APPC environment where sending and receiving functions run in parallel in separate processing regions.

Back-to-Back Sample Customization

The MERVA Link sample customization for the Back-to-Back Test Environment is independent of the DC environment (CICS or IMS). It defines one ASP and one MTP entry in the PT. The ASP entry specifies its own address (local node and ASP name) as the address of the partner application in the DEST parameter. The MTP must identify itself and provide a pointer to its associated ASP.

Sample PT generation statements for the MERVA Link Back-to-Back Test Environment that is used for the MERVA Link installation verification are shown in Figure 89.

```
         EKAPT TYPE=INITIAL,NODE=N1          MERVA LINK NODE N1
*
         EKAPT TYPE=ASP,                     APPL SUPPORT PROCESS      *
               NAME=(A1I,'BACK-TO-BACK WITHIN NODE N1'),               *
               MFSEXIT=7010,                 MFS USER EXIT NUMBER      *
               SENDQC=EKA1IS1,               SEND QUEUE                *
               DEST=(N1,A1I),                PARTNER ASP ADDRESS       *
               CONTROL=EKA1ICQ,              CONTROL QUEUE NAME        *
               IRROUTE=(ACK,EKAAWQ,CTLQ),    REC REPORT CORRELATION    *
               MTP=BTB                       NAME OF APPLICABLE MTP
*
         EKAPT TYPE=MTP,NAME=BTB,            MSG TRANSFER PROCESS      *
               DEST=BTB,                     MTP TYPE IS BTB           *
               ASP=A1I                       NAME OF APPLICABLE ASP
         EKAPT TYPE=FINAL
         END
```

Figure 89. PT Sample for the Back-to-Back Test Environment

The keyword **BTB** specified in the DEST parameter identifies the MTP as a Back-to-Back MTP. The default message transfer program used by an MTP of the type BTB is EKATM10. You do not need to specify the name of this program in an MTP parameter.

Customizing the MERVA System Control Facility

You can customize the following within the MERVA System Control Facility:

- The color of the MERVA command response display
- The color of the ASP and SCP list lines
- The color of the ASP and SCP list frame data
- The color of information and error messages
- The PF key allocation
- The command names and their alias names

Customizing the Display Panels

Customizing the Main Menu

The Main Menu of the MERVA System Control Facility and the MSC Local Help information is defined in the MCB EKAACMM that is part of the DSLMMFS interface program.

You can modify the contents of the main menu and the local help text according to your requirements.

The PF keys are defined in the MERVA ESA MFS PF-key table DSLMPF00 (PF key group 42).

Customizing the Display of a MERVA Command Response

The display of a MERVA Command response is defined in the MCB EKAAC00 that is part of the DSLMMFS interface program. The colors of the command response lines are set in an MFS editing user exit named EKAME012.

The PF keys are defined in the MERVA ESA MFS PF-key table DSLMPF00 (PF key group 42).

Customizing the Display of an ASP List

The display of the ASP list is defined in the MCB EKAAC01 that is part of the DSLMMFS interface program. The colors of the ASP list lines are set in an MFS editing user exit named EKAME010. You cannot modify the structure of an ASP list line.

The PF keys are defined in the MERVA ESA MFS PF-key table DSLMPF00 (PF key group 41).

Customizing the Display of Specific ASP/MTP Parameters

The display of specific ASP and MTP parameters is defined in the MCB EKAAC02 that is part of the DSLMMFS interface program. The color of the ASP list line is specified in the MFS editing user exit EKAME010. You cannot modify the structure of the ASP list line in this screen.

The PF keys are the same as in the ASP-list display.

Customizing the Display of an SCP List

The display of the SCP list is defined in the MCB EKAAC03 that is part of the DSLMMFS interface program. The colors of the SCP list lines are set in an MFS editing user exit named EKAME011. You cannot modify the structure of an SCP list line.

The PF keys are the same as in the ASP-list display.

Customizing the PT Header Display

The display of PT Header data is defined in the MCB EKAAC04 that is part of the DSLMMFS interface program.

The PF keys are the same as in the ASP-list display.

Customizing the Explanation Panels

The screens of the MERVA System Control Facility explanations displayed in response to the command EXPLAIN or XPL, are defined in the MCB EKAACHP. The identifier of the screen (ACMM, AC00, AC01, AC02, AC03, or AC04), from which the explanation is called, is provided in field EKAMTYPE. The contents of field EKAMTYPE determines the text displayed.

You can modify the explanation text according to your requirements.

Customizing the Command Names

The MERVA Link commands, the SWIFT Link DDS command, and the MERVA ESA queue test commands supported by the MERVA System Control Facility are defined in the MERVA Link command table EKAMSCMT. You can change the commands (labels of the DSLNCM macros) according to your requirements. You must, however, not modify the command codes or other parameters of the DSLNCM macros.

The characteristic of a MERVA Link command, whether it is a privileged command or whether it can be used by any MSC operator, cannot be customized.

The MERVA ESA Base commands and the other commands of the SWIFT Link and TELEX Link components are not defined in EKAMSCMT. These commands are defined in the MERVA ESA command table DSLNCMT.

Application Support Filter

A MERVA Link application support filter (ASF) is a customer-written program. It is associated with a specific MERVA Link application in the corresponding application support process (ASP) entry in the PT and is called by the MERVA Link when messages for this application are processed.

The purpose of an ASF is to control data that is passed across the MT layer boundary (boundary between the ASL and the MTL). It does not control MERVA ESA messages or the routing of messages within MERVA ESA.

Note: An ASF does not have access to the MERVA ESA services. For all activities dealing with MERVA ESA refer to the MERVA ESA MFS User Exit supported by the MERVA Link.

The MERVA Link supports up to three ASFs associated with a specific ASP. In the layered representation of the MERVA Link message handling system, an ASF is

located between the AS Layer and the MT Layer. Depending on the specific ASF, it can be a lower boundary extension of the ASL or an upper boundary extension of the MTL.

An ASF references and modifies internal data areas of the MERVA Link. You must carefully design and test your ASF definition because MERVA Link programs are not protected against the malfunction of an ASF.

The following describes the purpose of a MERVA Link ASF and the support provided by MERVA Link to develop an ASF. You also find an explanation of the ASF samples provided by the MERVA Link.

Note: All explanations also apply to service primitive filters. A service primitive filter differs from an ASF only in the set of events it must handle:

- An ASF is called only for a SUBMIT.Request and a DELIVER.Indication.
- A service primitive filter is called for all events at the MTL boundary.

A set of service primitive filters is specified in the partner table parameter FILTER with the keyword ALL.

ASF Called for a SUBMIT.Request

An ASF that is called for a SUBMIT.Request checks or modifies the request. If it does not reject the request, it passes the request to the next program, which is either another ASF or the Message Transfer Service Processor (MTSP). If the ASF rejects the request, it immediately returns to the calling program and provides appropriate error information, including the identifier of the new service primitive SUBMIT.Confirmation.

Either an application message (IM-ASPDU) or a receipt report (SR-ASPDU) is passed by an ASP in a SUBMIT.Request to the MTSP. An ASF must check what kind of protocol data unit (PDU) is applicable and acts appropriately.

An ASF can authenticate and encrypt a message. If both authentication and encryption are applied in this sequence, the encryption and decryption process are verified by the authentication.

An ASF can also add customer defined data elements to the message heading.

ASF Called for a DELIVER.Indication

An ASF that is called for a DELIVER.Indication must check or modify this request, and either pass this DELIVER.Indication to the next program in the applicable sublayer structure to the ASP or reject it.

If the ASF rejects this request, it immediately returns to the calling program and provides appropriate error information (including the identifier of the new service primitive DELIVER.Response).

Either an application message (IM-ASPDU) or a receipt report (SR-ASPDU) is passed by the MTSP in a DELIVER.Indication to an ASP. An ASF must check what kind of PDU is applicable and acts appropriately.

If the authentication of a received message fails, the ASF can reject the delivery or pass this message to the next program to the ASP. In this case the ASF provides appropriate error control information (including the identifier of the new service

primitive DELIVER.Response when the delivery is rejected). If the received message is encrypted, the ASF must decrypt the message.

An ASF can also process customer defined data elements contained in the message heading.

ASF Programming Interface

The ASF programming interface comprises four parts. It is the program entry and the return to the caller (as in a user exit). The ASF programming interface, however, comprises two more parts, the call of the next program and the return from that program.

ASF Program Entry

An ASF, called by MERVA Link or another ASF, gets the message transfer parameter list EKAMTPL as a CICS Commarea in a CICS environment and a pointer to the address of the EKAMTPL as input parameter list in the IMS environment. Among other information, the EKAMTPL contains:

- The service primitive identifier
- A pointer to the P2 PDU (IM-ASPDU or SR-ASPDU)
- Pointers to the PT header and the applicable ASP entry
- Pointers to P1 originator and recipient information

Call the Next Program

When the ASF has completed its activity for a specific event, it calls the “next program”. This “next program” is determined by the service primitive and its own identity.

Any ASF is a member of a program list that consists of three, four, or five programs. A MERVA Link ASP is located at the top of the list and the MERVA Link MTSP is located at the bottom of the list. One, two, or three ASFs can be between the ASP and MTSP programs. In the MERVA Link sample, this list reads:

- EKAAS10 or EKAAR10
- EKAAF10
- EKAAF20
- EKASP10

The next program of EKAAF10 for a SUBMIT.Request is EKAAF20, the lower boundary ASF. The next program of EKAAF10 for a DELIVER.Indication is EKAAR10, the MERVA Link receiving ASP. The next program of EKAAF20 for a SUBMIT.Request is EKASP10, the MERVA Link MTSP. The next program of EKAAF20 for a DELIVER.Indication is EKAAF10, the upper boundary ASF.

The names of the applicable ASFs are contained in the ASP entry of the PT.

Return from the Next Program

The “next program” returns to the calling ASF with an updated parameter list (EKAMTPL). The service primitive identifier field contains the identifier of the secondary service primitive (SUBMIT.Confirmation or DELIVER.Response) that corresponds to the applicable primary service primitive (SUBMIT.Request or DELIVER.Indication) and return information. The latter information tells whether the SUBMIT.Request was successfully processed by the lower layer programs, or whether the DELIVER.Indication was successfully processed by the upper layer programs.

Return to the Caller

The ASF returns to the calling program. If it does not call the “next program” it must provide the identifier of the secondary service primitive (SUBMIT.Confirmation or DELIVER.Response) that corresponds to the applicable primary service primitive (SUBMIT.Request or DELIVER.Indication), and it must provide information describing the reason for this short cut.

In any case, the ASF can check and modify the information contained in the message transfer parameter list.

ASF Samples

The MERVA Link sample contains two ASFs named EKAAF10 and EKAAF20:

EKAAF10 Authenticates an outgoing message and verifies that an incoming message has not been altered.

EKAAF20 Encrypts an outgoing message and decrypts an incoming message.

These samples are provided in ASSEMBLE source code for your reference.

ASF Skeleton

An ASF skeleton used as the base for the sample MERVA Link ASFs, EKAAF10, and EKAAF20, is shown below. It is an executable ASF without specific activity.

This skeleton indicates the places where the developer of an ASF must add code for the specific activity of that ASF. You should use this skeleton to develop your own ASFs.

```
*********************************************************************** 00001
*        EKAAF100 MERVA LINK APPLICATION SUPPORT FILTER SKELETON        00002
*********************************************************************** 00003
EKAAF100 EKAASFB PRINT=OFF             GENERATE ASF BEGIN CODE          00004
*---------------------------------------------------------------------- 00005
*        MODULE WORK AREA USER EXTENSION                                00006
*---------------------------------------------------------------------- 00007
*        ADD YOUR WORK AREA FIELD DEFINITIONS HERE                      00008
*====================================================================== 00009
*        PROGRAM START                                                  00010
*====================================================================== 00011
EKAAF100 CSECT                                                          00012
*/* SELECT SERVICE PRIMITIVE:                                          00013
AF10010S DS    0H                                                       00014
*********************************************************************** 00015
*/* WHEN SUBMIT REQUEST:                                               00016
*********************************************************************** 00017
AF10011W DS    0H                                                       00018
         CLC   TPLSPID,AF10SURQ        SUBMIT REQUEST ?                 00019
         BNE   AF10012W                NO, CHECK FOR OTHER SERV PRIM    00020
*        ADD YOUR SUBMIT.REQUEST PROCESSING CODE HERE                   00021
         B     AF10010X                                                 00022
*********************************************************************** 00023
*/* WHEN DELIVER INDICATION:                                           00024
*********************************************************************** 00025
AF10012W DS    0H                                                       00026
         CLC   TPLSPID,AF10DLIN        DELIVER INDICATION ?             00027
         BNE   AF10013W                NO, CHECK FOR OTHER SERV PRIM    00028
*        ADD YOUR DELIVER.INDICATION PROCESSING CODE HERE               00029
AF10013W DS    0H                                                       00030
AF10010X DS    0H                                                       00031
*/* ENDSELECT                                                          00032
         EKAASFC MYNAME=EKAAF10        CALL APPLICABLE NEXT PROGRAM     00033
         EKAASFE                       RETURN TO THE CALLER             00034
*/* ENDMODULE EKAAF100                                                 00035
AF10SURQ DC    CL4'SURQ'              SUBM.REQ SERVICE PRIMITIVE ID    00036
AF10DLIN DC    CL4'DLIN'              DELV.IND SERVICE PRIMITIVE ID    00037
         END                                                            00038
```

Figure 90. ASF Sample Skeleton

ASF User-Code Elements

One of the following elements is usually contained in the ASF user code:

- Check for message or receipt report
- Locate a data element in the message heading
- Locate the message text
- Add a data element to the message heading
- Report authentication failure
- Reject message delivery due to authentication failure

Commented examples for each of these elements are shown in the following figures (Figure 91 on page 200 to Figure 96 on page 203). These examples are from the MERVA Link sample ASF EKAAF10.

Check for Application Message: Each of the user-code elements starts with an instruction to point to the message transfer parameter list EKAMTPL. This instruction is not necessary if the contents of RMTP (general register 5) is not altered in the ASF. Initially, RMTP points to the EKAMTPL.

Some of the user-code elements continue with an instruction to point to the Body-Part Data Segment (L RPDU,TPAMCA). This instruction is not necessary if the contents of RPDU (general register 4) is not altered in the ASF. Initially, RPDU points to the Body-Part Data Segment.

```
*/* IF APPLICATION MESSAGE SUBMISSION IS REQUESTED:
         L     RMTP,WAMTPL             POINT TO THE EKAMTPL
         L     RPDU,TPAMCA             POINT TO THE P2 PDU
         CLC   PDUID,HSR1ASPH          IS IT AN ASPDU HEADING ?
         BNE   HSR1010A                NO, ...
*/* HANDLE APPLICATION MESSAGE.

⋮

HSR1010A DS    0H
*/* ENDIF

⋮

HSR1ASPH DC    AL2(@PIASPH)            IM-ASPDU HEADING ID
```

Figure 91. ASF User-Code Element Check for Application Message

Locate Data Element: The following code scans the level 1 data element 0120 (IM-ASPDU Heading) for a specific level 2 data element. Corresponding code applies if an implicit level 2 data element is to be scanned for a level 3 data element. Level 3 data elements are always explicit in the MERVA Link architecture.

```
*/* LOCATE AUTHENTICATION DATA IN THE P2 PDU.
         L     RMTP,WAMTPL             POINT TO THE EKAMTPL
         L     RPDU,TPAMCA             POINT TO THE P2 PDU
         LR    R1,RPDU                 POINT TO THE END OF THE HEADING
         AH    R1,PDULL                *
         LA    RPDU,PDUDB              POINT TO FIRST LOWER LEVEL DE
*/* DO WHILE AUTH DE NOT FOUND AND MESSAGE HEADING NOT EXHAUSTED:
HDI1020D DS    0H
         CR    RPDU,R1                 STILL WITHIN THIS HEADING DE ?
         BNL   HDI1020T                NO, AUTHENTICATION DE NOT FOUND
         CLC   PDUID,HDI1AUTH          IS IT THE AUTH DE ?
         BE    HDI1020T                YES, EXIT LOOP
*/* POINT TO NEXT DE IN THE MESSAGE HEADING.
         AH    RPDU,PDULL              POINT TO THE NEXT DE
         B     HDI1020D
HDI1020T DS    0H
*/* ENDDO
*/* IF AUTHENTICATION DATA ELEMENT FOUND:
         CLC   PDUID,HDI1AUTH          IS IT THE AUTHENTICATION DE ?
         BNE   HDI1030A                NO, ...

⋮

HDI1030A DS    0H
*/* ENDIF

⋮

HDI1AUTH DC    AL2(@PIAUTH)            AUTHENTICATION INFO DE ID
```

Figure 92. ASF User-Code Element Locate Data Element

Locate the Message Text: The representation of the message text at the MTL boundary has been modified in MERVA Link of MERVA ESA. You must modify an application support filter written for MERVA/370 V2 accordingly.

The message text is contained in a storage area. It has a standard MERVA ESA buffer format with two fullword length fields as the buffer prefix. The standard MERVA ESA buffer format with halfword length fields is not supported for a MERVA Link message text buffer at the MTL boundary.

The chain pointer in the body part header points to a body descriptor. The body descriptor consists of a single body part descriptor as the message body consists of a single body part only.

The data buffer address in the body part descriptor points to the MERVA ESA buffer containing the message text. The length of the message text may exceed 32KB.

Modified sample code to locate the message text at the MTL boundary is shown below. The modified statements are flagged by '@@@'.

```
*/* POINT TO THE MESSAGE TEXT.
         L     RMTP,WAMTPL             POINT TO THE EKAMTPL
         L     RPDU,TPAMCA             POINT TO THE P2 PDU
         AH    RPDU,PDULL              POINT TO THE BODY PART HEADER
         L     RPDU,PDUDDSP            POINT TO THE BODY DESCR.     @@@
         L     RPDU,BPDDBA             POINT TO THE BP DATA BUFFER  @@@
         L     R1,4(RPDU)              GET DATA BUFFER DATA LENGTH  @@@
         LA    R1,0(R1)                RESET HIGH ORDER BIT         @@@
*/* IF TEXT AVAILABLE:
         SH    R1,GAD1FLFL             DATA SEGMENT TEXT LENGTH     @@@
         BNP   GAD1050A                NO MESSAGE TEXT AVAILABLE

⋮

GAD1050A DS    0H
*/* ENDIF

⋮

GAD1FLFL DC    H'4'                    LENGTH OF A FW LENGTH FIELD  @@@
```

Figure 93. ASF User-Code Element Locate the Message Text

Add a Data Element to an Outgoing Message Heading: To add a user-defined data element to an outgoing message heading you must consider the following:

- The body-part header is appended to the message heading. It has a length of 12 bytes. Its most important information is the pointer to the body-part data segment that contains the message text. Before adding a data element to the message heading, the body-part header must be saved and appended to the extended message heading later.
- MERVA Link reserves space in the message heading buffer for up to 1012 bytes, which is the maximum heading length supported. 12 extra bytes are reserved for the body-part header.

User-defined data elements must have the format of a valid MERVA Link data element. The data element identifier must be a data element identifier reserved for customer use.

When a data element has been added to the message heading, the message heading data element length must be updated as specified by the length of the additional data element.

```
WABPHD   DS    XL12                   BODY PART HEADER SAVE AREA
WAAUTH   DS    XL16                   AUTHENTICATION DATA

⋮

*/* ADD AUTHENTICATION DATA TO THE MESSAGE HEADING.
         L     RMTP,WAMTPL            POINT TO THE EKAMTPL
         L     RPDU,TPAMCA            POINT TO THE P2 PDU
         AH    RPDU,PDULL             POINT TO THE BODY PART HEADER
         MVC   WABPHD,PDUDP           SAVE BODY PART HEADER
         MVC   PDULL,HDI1ADEL         AUTHENTICATION DATA ELEMENT LNGT
         MVC   PDUID,HDI1AUTH         AUTHENTICATION DATA ELEMENT ID
         MVC   PDUDB(16),WAAUTH       AUTHENTICATION DATA
         AH    RPDU,PDULL             SKIP THIS NEW DATA ELEMENT
         MVC   0(L'WABPHD,RPDU),WABPHD APPEND BODY PART HEADER
         L     RPDU,TPAMCA            POINT TO THE P2 PDU
         LH    R1,HDI1ADEL            LENGTH OF AUTHENTICATION DE
         AH    R1,PDULL               ADD OLD MSG HEADING LENGTH
         STH   R1,PDULL               SET NEW MESSAGE HEADING LENGTH

⋮

HDI1ADEL DC    AL2(20)                AUTHENTICATION INFO DE LENGTH
HDI1AUTH DC    AL2(@PIAUTH)           AUTHENTICATION INFO DE ID
```

Figure 94. ASF User-Code Element Add Data Element

Report Authentication Failure: The event return code and the event diagnostic code in this example are saved in the MERVA Link control fields EKADELRC and EKADELDC in the delivered message. These fields can be checked in a MERVA ESA routing table. Messages with authentication failure can be routed to an incoming message error queue using this information.

The message routed to the message error queue can be authenticated by a MERVA ESA operator (user) using standard MERVA ESA functions.

```
*/* REPORT AUTHENTICATION FAILURE.
         L     RMTP,WAMTPL            POINT TO THE EKAMTPL
         MVC   TPAERC,AF10ECWR        SET EVENT RETURN CODE TO WARNING
         MVC   TPAEDC,AF10DCAF        INDICATE AUTHENTICATION FAILED

⋮

         EKAASFC ,                    PASS EVENT TO THE NEXT PROGRAM

⋮

AF10ECWR DC    AL2(@ERCWR)            WARNING EVENT CODE FOR AUTH ERR
AF10DCAF DC    CL6'AUTH F'            DIAG CODE FOR AUTH FAILURE
```

Figure 95. ASF User-Code Element Report Authentication Failure

Reject Message Delivery: Store an ASF return code (16 ... 255) in TPARC to indicate a receiving process error. The event return code and the event diagnostic code stored in TPAERC and TPAEDC, respectively, are returned to the sending partner application. The message is not delivered. The effect of this code is that the

sending partner application is set inoperable (application status code 09, indicating that an application error has been reported by the partner system), and that the message transfer is stopped.

Message transfer can be resumed upon request by the MERVA ESA control operator at the sending side when the authentication problem has been fixed. This requires a cooperation of the MERVA Link system administrators at the sending and the receiving side.

```
*/* REJECT MESSAGE DELIVERY DUE TO AUTHENTICATION FAILURE.
         L     RMTP,WAMTPL             POINT TO THE EKAMTPL
         MVC   TPARC,AF10RCAU          AUTHENTICATE MSG TEXT FAILED
         MVC   TPAERC,AF10ECER         SET EVENT RETURN CODE TO ERROR
         MVC   TPAEDC,AF10DCAF         INDICATE AUTHENTICATION FAILED
         MVC   TPLSPID,AF10DLRS        SET DELIVER.RESP SERV PRIMITIVE

⋮

         EKAASFE                       RETURN TO CALLER

⋮

AF10RCAU DC    H'178'                  ASF RC: AUTHENTICATION ERROR
AF10ECER DC    AL2(@ERCER)             ERROR EVENT CODE FOR AUTH ERROR
AF10DCAF DC    CL6'AUTH F'             DIAG CODE FOR AUTH FAILURE
AF10DLRS DC    CL4'DLRS'               DELV.RSP SERVICE PRIMITIVE ID
```

Figure 96. ASF User-Code Element Reject Message Delivery

Support of the MFS User Exits

The MERVA Link supports MERVA ESA MFS user exits for different purposes. The general rules to develop MERVA ESA MFS user exits are described in the *MERVA for ESA System Programming Guide*. The following gives additional rules to develop MERVA ESA MFS user exits that are to be used in the MERVA Link environment.

If requested, MERVA Link calls an MFS user exit at the following (logical) places in an ASP:

- When an outgoing message is about to be processed the user exit can tell MERVA Link to reroute it immediately. This means, the user exit can close the ASP temporarily for that specific ready-to-send message.
- When an outgoing message is processed, and MERVA Link must know whether it is an application message or an acknowledgment message. The user exit uses a receipt return code to indicate this in the MERVA Link control field EKARECRC. A valid receipt return code (00, 04, or 08) indicates an acknowledgment message.
- When the transfer of an outgoing message has been confirmed, the user exit can identify this message as delivered to the recipient application in application specific terms. Both, application messages and acknowledgment messages can be controlled and modified by a user exit at this place.

  The class of a message at this place is CF or CA. A message with class CA is a confirmed application message that already contains acknowledgment control information, that is, a valid receipt return code. A valid receipt return code (00, 04, or 08) in a message of the class CF indicates an acknowledgment message. Otherwise, the confirmed message is an application message.
- When an incoming application message has been received the user exit can check and modify its contents. The user exit is called when the message is saved

in the TOF and shortly before the message is routed to the applicable MERVA ESA destination queue or queues.

- When an incoming acknowledgment message is received and, if applicable, merged with the original (reported) message, the user exit can check and modify its contents. The user exit is called when the message is saved in the TOF and shortly before the message is routed to the applicable MERVA ESA destination queue or queues.
- When a message is recovered because it cannot be transmitted using MERVA Link or cannot be successfully delivered, the user exit can check and modify its contents. The user exit is called when the message is saved in the TOF and shortly before the message is routed to the applicable MERVA ESA "recovered message" queue or queues.

  Message recovery can be requested by an operator using the RECOVER and IPRECOV commands of the MERVA Link Control Facility. A message can also be recovered automatically from a delivery error (IPRECOV=AUTO specified in the partner table ASP entry). The user exit is called in all these situations.

MFS User Exit Interface

The MFS user exit called by the MERVA Link obtains the following input parameters and data from the MERVA Link:

- Message in the TOF and a pointer to the TOF in MFSLTOF and in general register 11.
- A pointer to an 8-byte field containing the message type in MFSLMSG. The contents of this 8-byte field depends on the value of the FORMAT parameter of the ASP definition in the sending MERVA Link system.

  For FORMAT=QUEUE it is the content of the TOF field DSLEXIT.

  For FORMAT=NET it is the content of the TOF field specified as second subparameter of the FORMAT parameter (default is DSLEXIT).

  For FORMAT=MCB it is the MCB specified as second subparameter of the FORMAT parameter.

  When the user exit is called to handle a recovered message the message type is not provided as a user exit input parameter.
- Address of the external module communication-area EKAXCPL as input buffer (MFSLIBUF). This area contains the pointer to the partner table in CPLPTBA, the pointer to the applicable partner table ASP entry in CPLPTEA, and the user exit function in CPLMUXF. CPLMUXF contains:

  S For a ready-to-send message

  O For an outgoing message

  C For a confirmed message

  I For an incoming application message

  R For an incoming acknowledgment message (status report)

  V For a recovered or re-routed message

  The user exit must reference only those three fields in the XCPL. It must not reference any other field in this MERVA Link control area. It must not modify any data at all in this MERVA Link control area.
- Address of a work area in the format of a TOF field buffer as output buffer (MFSLOBUF). This buffer has a length of 256 bytes and a standard MERVA ESA buffer prefix of 8 bytes.

The user exit can use this buffer to “read” or “write” data from or to the TOF. If MERVA Link control fields are to be processed, the field names are to be obtained from the partner table header.

The general registers are initialized as follows:

R4 TOF field-buffer pointer

R5 MERVA ESA MFS permanent-storage pointer

R6 MERVA ESA MFS temporary-storage pointer

R7 MERVA ESA MFS parameter-list pointer

R8 Partner table base register

R9 Pointer to the MERVA Link communication area EKAXCPL

R10 User Exit base register

R11 Pointer to the TOF

R12 Pointer to the MERVA ESA communication area DSLCOM

Registers 0 to 3 must be used as work registers. The use of registers 13 to 15 is defined by operating system standards. Within these standards, the latter registers can be used as work registers.

Start MFS User Exit Macro EKAUXS

The MERVA Link MFS user exit start macro EKAUXS establishes the MERVA ESA environment for a MERVA ESA MFS User exit to be used with the MERVA Link. It should be the first noncomment statement in the source code used to generate a user exit in assembler language. For details refer to *MERVA for ESA Macro Reference*.

The user exit start macro copies the message type, which is pointed to by the field MFSLMSG, to the local user exit field UXMSGID. In addition, it initializes the local user exit TOF field name prefix UXFLDPF to 'EKA', the prefix of all MERVA Link control field names. The two local user exit fields UXMSGID and UXFLDPF must therefore be defined in any MERVA Link user exit that starts with the EKAUXS macro instruction.

The user exit start macro supports a user exit that contains CICS commands (EXEC CICS statements). Specific rules must be observed when CICS commands are issued in an MFS user exit. These rules are described later in this chapter.

MFS User Exit Sample

The MERVA Link provides a sample MFS user exit named EKAMU010. The activity performed by this user exit is described in detail in the sample code.

Figure 97 on page 206 shows an MFS user exit skeleton named EKAMU001 that was the base for the sample MFS user exit EKAMU010.

Note: EKAMU001 is a fully operational MFS user exit with zero activity as far as the messages are concerned. Use this skeleton when developing your own user exits.

```
********************************************************************** 00001
*        EKAMU001 MERVA LINK MFS USER EXIT SAMPLE SKELETON             00002
********************************************************************** 00003
EKAMU001 EKAUXS NUM=7001                                               00004
*/* SELECT USER EXIT FUNCTION:                                         00005
********************************************************************** 00006
*/* WHEN READY TO SEND MESSAGE IS TO BE PROCESSED:                     00007
********************************************************************** 00008
MU00011W DS    0H                                                      00009
         CLI   CPLMUXF,@UXFRTS         READY TO SEND MESSAGE ?         00010
         BNE   MU00012W                NO, CHECK FOR OTHER FUNCTION    00011
*/* INSERT YOUR CODE FOR READY TO SEND MESSAGES HERE.                  00012
         B     MU00010X                                                00013
********************************************************************** 00014
*/* WHEN OUTGOING MESSAGE IS TO BE PROCESSED:                          00015
********************************************************************** 00016
MU00012W DS    0H                                                      00017
         CLI   CPLMUXF,@UXFOBM         OUTBOUND MESSAGE ?              00018
         BNE   MU00013W                NO, CHECK FOR OTHER FUNCTION    00019
*/* INSERT YOUR CODE FOR OUTGOING MESSAGES HERE.                       00020
         B     MU00010X                                                00021
********************************************************************** 00022
*/* WHEN CONFIRMED MESSAGE IS TO BE PROCESSED:                         00023
********************************************************************** 00024
MU00013W DS    0H                                                      00025
         CLI   CPLMUXF,@UXFCFM         CONFIRMED MESSAGE ?             00026
         BNE   MU00014W                NO, CHECK FOR OTHER FUNCTION    00027
*/* INSERT YOUR CODE FOR CONFIRMED MESSAGES HERE.                      00028
         B     MU00010X                                                00029
********************************************************************** 00030
*/* WHEN INCOMING REPORT IS TO BE PROCESSED:                           00031
********************************************************************** 00032
MU00014W DS    0H                                                      00033
         CLI   CPLMUXF,@UXFIBR         INBOUND REPORT ?                00034
         BNE   MU00015W                NO, CHECK FOR OTHER FUNCTION    00035
*/* INSER YOUR CODE FOR ACKNOWLEDGED OR ACKNOWLEDGMENT MESSAGES HERE.  00036
         B     MU00010X                                                00037
********************************************************************** 00038
*/* WHEN INCOMING APPLICATION MESSAGE IS TO BE PROCESSED:              00039
********************************************************************** 00040
MU00015W DS    0H                                                      00041
         CLI   CPLMUXF,@UXFIBM         INBOUND MESSAGE ?               00042
         BNE   MU00016W                NO, CHECK FOR OTHER FUNCTION    00043
*/* INSERT YOUR CODE FOR INCOMING APPLICATION MESSAGES HERE.           00044
```

Figure 97. EKAMU001 MFS User Exit Sample Skeleton (Part 1 of 2)

```
*********************************************************************** 00045
*/* WHEN RECOVERED OR RE-ROUTED MESSAGE MUST BE PROCESSED:              00046
*********************************************************************** 00047
MU00016W DS    0H                                                       00048
         CLI   CPLMUXF,@UXFRCV         RECOVERED MESSAGE ?             00049
         BNE   MU01017W                NO, CHECK FOR OTHER FUNCTION    00050
*/* INSERT YOUR CODE FOR RECOVERED OR RE-ROUTED MESSAGES HERE.          00051
MU00017W DS    0H                                                       00052
MU00010X DS    0H                                                       00053
*/* ENDSELECT                                                          00054
         B     MFSGOOD                 RETURN TO MERVA LINK            00055
*/* ENDMODULE EKAMU001                                                 00056
         LTORG                                                         00057
*--------------------------------------------------------------------- 00058
*        USER EXIT WORK FIELDS                                         00059
*--------------------------------------------------------------------- 00060
MFSTS    DSECT                         MFS TEMP STORAGE (CONTINUED)    00061
         DS    0D                                                      00062
UXMSGID  DS    CL8                     MESSAGE ID                      00063
UXFLDNM  DS    0CL8                    TOF FIELD NAME                  00064
UXFLDPF  DS    CL3                     TOF FIELD NAME PREFIX           00065
UXFLDID  DS    CL5                     TOF FIELD NAME IDENTIFIER       00066
MFSTTSLL EQU   *-MFSTS                                                 00067
         END                                                           00068
```

Figure 97. EKAMU001 MFS User Exit Sample Skeleton (Part 2 of 2)

CICS Commands in an MFS User Exit

An MFS user exit called by MERVA Link in the CICS environment may request CICS services (issue EXEC CICS commands) if it follows a number of rules. These rules are explained in the following and shown in Figure 98 on page 208.

```
*ASM CICS (NOPROLOG NOEPILOG)                                          0001
********************************************************************** 0002
                                                                       0003
********************************************************************** 0004
         TITLE 'EKAMU002 - MERVA LINK USER EXIT WITH CICS COMMANDS      0005
********************************************************************** 0006
EKAMU002 EKAUXS NUM=7002,CICS=YES                                      0007
*/* SELECT USER EXIT FUNCTION:                                         0008
********************************************************************** 0009
*/* WHEN READY TO SEND MESSAGE IS TO BE PROCESSED:                     0010
********************************************************************** 0011
         SPACE                                                         0012
MU00011W DS    0H                                                     0013
         CLI   CPLMUXF,@UXFRTS         READY TO SEND MESSAGE ?        0014
         BNE   MU01012W                NO, CHECK FOR OTHER FUNCTION   0015
*/* SET REGISTERS FOR THE CICS INTERFACE.                              0016
         L     DFHEIBR,UXDFHEIB        POINT TO THE CICS EIB          0017
         L     DFHEIPLR,UXDFHEIS       POINT TO THE CICS EISTG        0018
*/* ISSUE CICS COMMANDS.                                               0019
         EXEC CICS ......                                              0020
         EXEC CICS ......                                              0021
*/* RESTORE EKAUXS REGISTERS.                                          0022
         L     R9,UXEKACPL             RESTORE PTR TO THE XCPL        0023
         L     R11,UXDSLTOF            RESTORE PTR TO THE TOF         0024
⋮
                                                                       0025
UXDFHEIB DS    A                       PTR TO THE CICS EIB            0026
UXDFHEIS DS    A                       PTR TO THE CICS EISTG          0027
UXEKACPL DS    A                       PTR TO THE EKAXCPL             0028
UXDSLTOF DS    A                       PTR TO THE TOF                 0029
MFSTTSLL EQU   *-MFSTS                                                 0030
         END                                                           0031
```

Figure 98. EKAMU002 MFS User Exit Elements Supporting CICS Commands

Notes:

1. An MFS user exit that contains CICS commands must be processed by the CICS Command Preprocessor before it is assembled. The CICS Command Preprocessor must, however, not add its prologue and epilogue code to the user exit source. It is asked to do so by a preprocessor statement at the begin of the user exit source code (see statement 0001 in Figure 98).
2. The setup of the CICS environment for the user exit code must be requested using the parameter CICS=YES in the EKAUXS macro (see statement 0007 in Figure 98).

   The CICS environment is represented by the addresses of the CICS Interface storage area (DFHEISTG) and the CICS EXEC Interface Control Block (DFHEIB). The code generated by EKAUXS with CICS=YES saves these addresses in the fields UXDFHEIS and UXDFHEIB, respectively. This is why these fields must be defined in the user exit (see statements 0026 and 0027 in Figure 98).
3. The code generated by EKAUXS with CICS=YES saves the addresses of the MERVA Link communication area EKAXCPL and the TOF in the fields UXEKACPL and UXDSLTOF, respectively, to prepare restoring the corresponding pointers in registers 9 and 11 after a CICS command. This is why these fields must be defined in the user exit (see statements 0028 and 0029 in Figure 98).

4. Before a CICS command can be issued in a user exit, the CICS environment must be enabled by loading the addresses of the CICS EIB and EISTG into the registers 9 and 11 (DFHEIBR and DFHEIPLR), respectively (see statements 0017 and 0018 in Figure 98 on page 208).

   The initial contents of registers 9 and 11 (pointers to the EKAXCPL and the TOF) can be easily restored after the CICS commands (see statements 0023 and 0024 in Figure 98 on page 208).

Link-Editing an MFS User Exit

An MFS user exit must be defined in the MERVA ESA MFS program table (DSLMPTT). The definition statement specifies whether the user exit must be part of (linked to) the MERVA ESA Message Format Services (MFS) interface (DSLMMFS), or whether it is a separate load module.

When a user exit is defined with LINK=YES in the DSLMPTT, the DSLMMFS interface program must be link-edited to activate a new or modified user exit. No specific measures apply in this case if the user exit issues CICS commands.

When a user exit is defined with LINK=NO in the DSLMPTT, the user exit is called as a separate load module by MERVA ESA. It is not linked to the MERVA ESA DSLMMFS interface program. No specific considerations apply in this case in the IMS environment.

In the CICS environment, however, any user exit that is not linked to the DSLMMFS interface program must be defined to CICS as a processing program (DFHPPT entry, or corresponding online definition).

If the user exit issues CICS commands, it must be linked with the CICS EXEC Interface Stub (DFHEAI) as specified by CICS for any processing program that issues CICS commands.

MERVA ESA Unique Message Reference

A user exit can ask the MERVA Link to use the MERVA ESA unique message reference (UMR) for acknowledgment correlation purposes. The MERVA Link uses the data contained in the field EKAAMSID (IAM Message Identifier) for these correlation purposes. MERVA Link generates correlation data if it is not provided in an outgoing application message. This default correlation data can be overwritten by a user exit with the MERVA ESA unique message reference (or any other unique data).

Sample code to override the available message identifier with the MERVA ESA unique message reference is shown below.

```
*-------------------------------------------------------------------- 00001
* PROVIDE MERVA UNIQUE MESSAGE REFERENCE AS MERVA LINK IAM MESSAGE      00002
* IDENTIFIER FOR AN OUTGOING APPLICATION MESSAGE.                      00003
*-------------------------------------------------------------------- 00004
         SPACE                                                        00005
*/* READ RECEIPT RETURN CODE FIELD FROM THE TOF.                      00006
         MVC   UXFLDID,PTHRECRC       RECEIPT RC FIELD NAME IDENTIFIER 00007
         XC    4(8,R4),4(R4)          RESET TOF FLD BUFFER DATA LENGTH 00008
         DSLTSV TYPE=READ,FDNAM=UXFLDNM,BUFFER=(R4),NESTID=MU01CFNI,   *00009
               TOF=(R11),PREFIX=TS$,MF=(E,MFSTSVL)                    00010
         SPACE                                                        00011
*/* IF APPLICATION MSG IN PROCESS (NO VALID RECEIPT RC FOUND):        00012
         LTR   R15,R15                READ TOF FIELD SUCCESSFUL ?     00013
         BNZ   MU01040P               NO, TOF FIELD NOT FOUND         00014
         CLC   MU01RC00,8(R4)         VALID RECEIPT RETURN CODE ?     00015
         BE    MU01040A               YES, NO APPLICATION MSG IN PROC 00016
         CLC   MU01RC04,8(R4)         VALID RECEIPT RETURN CODE ?     00017
         BE    MU01040A               YES, NO APPLICATION MSG IN PROC 00018
         CLC   MU01RC08,8(R4)         VALID RECEIPT RETURN CODE ?     00019
         BE    MU01040A               YES, NO APPLICATION MSG IN PROC 00020
*/* GET UNIQUE MESSAGE REFERENCE FROM TOF FIELD DSLUMRIN.             00021
MU01040P DS    0H                                                    00022
         DSLTSV TYPE=READ,FDNAM=MU01UMR,BUFFER=(R4),                  *00023
               TOF=(R11),PREFIX=TS$,MF=(E,MFSTSVL),NESTID=MU01XFNI    00024
         SPACE                                                        00025
*/* IF READ TOF FIELD SUCCESSFUL:                                     00026
         LTR   R15,R15                REQUEST SUCCESSFUL ?            00027
         BNZ   MU01050A               NO, INDICATE ERROR              00028
*/* PROVIDE UMR AS IAM MESSAGE ID.                                    00029
         MVC   UXFLDID,PTHAMSID       SET CONTROL FIELD NAME          00030
         DSLTSV TYPE=WRITE,FDNAM=UXFLDNM,NESTID=MU01CFNI,DAINDEX=1,   *00031
               BUFFER=(R4),TOF=(R11),PREFIX=TS$,MF=(E,MFSTSVL)        00032
MU01050A DS    0H                                                    00033
*/* ENDIF                                                           00034
MU01040A DS    0H                                                    00035
*/* ENDIF                                                           00036
*/* IGNORE ANY ERROR AT THIS PLACE.                                   00037
         SLR   R15,R15                INDICATE ALL OK                 00038
⋮
                                                                      00039
MU01UMR  DC    CL8'DSLUMRIN'          MERVA UMR FIELD NAME            00041
MU01XFNI DC    AL1(0)                 DSLUMRIN FIELD NESTING ID       00042
MU01CFNI DC    AL1(1)                 MERVA LINK CTRL FLD NESTING ID  00043
MU01RC00 DC    CL2'00'                VALID RECEIPT RETURN CODE       00044
MU01RC04 DC    CL2'04'                VALID RECEIPT RETURN CODE       00045
MU01RC08 DC    CL2'08'                VALID RECEIPT RETURN CODE       00046
⋮
                                                                      00047
UXFLDNM  DS    0CL8                   TOF FIELD NAME                  00049
UXFLDPF  DS    CL3                    TOF FIELD NAME PREFIX           00050
UXFLDID  DS    CL5                    TOF FIELD NAME IDENTIFIER       00051
```

Figure 99. Sample Code: Use MERVA ESA UMR for ACK Correlation

If unique data is provided for ACK correlation other than the MERVA ESA unique message reference, an additional rule applies for this other data:

- It must be unique within the time frame of pending acknowledgments.
- It must also be unique for a specific message.

Uniqueness for a specific message means that the user exit provides the same correlation data if the message is presented twice. A second presentation to the user exit may happen when the MERVA Link recovers in-process messages from the control queue after a processing failure.

The correlation of an incoming acknowledgment message with the original outgoing application message may fail if this rule is not observed. The MERVA ESA unique message reference is both unique within the time frame of pending acknowledgments and unique for each message that is presented twice to a user exit.

Additional User Exit Considerations

A user exit must take following situations into consideration.

- A ready to send message (user exit function 'S') may contain an IAM Message Identifier in the field EKAAMSID. A user exit can delete this field (or set its contents to blanks) at this place to ask MERVA Link to generate a new message identifier.

  When the user exit is called for an outbound message (user exit function 'O') it must not delete or blank out the field EKAAMSID.

  A user exit can put a valid message identifier into the field EKAAMSID at both places (user exit functions 'S' and 'O').
- An outbound message (user exit function 'O') may contain a receipt return code of 01, 05, or 09 in the field EKARECRC indicating an application message that has been acknowledged before the transfer confirmation was received. The user exit must process this message as an application message rather than as an acknowledgment message.

  The user exit must not modify or delete fields in the TOF that are related to the ACK if EKARECRC contains a receipt return code of 01, 05, or 09. These fields are EKARECDT, -RECRC, -RECDC, and -RDATA.
- An inbound report (user exit function 'R') may contain an acknowledged message (acknowledgment message correlated and merged with the reported message) with a (masked) receipt return code of 01, 05, or 09 in the field EKARECRC, and the message class IP in the field EKACLASS. This indicates an application message that has been acknowledged before the transfer confirmation was received (it is still 'in process').

  The user exit has to take into consideration that this IP message is rerouted to the application control queue. In particular, it must not change the masked receipt return code in the field EKARECRC.

  The masked receipt return code is unmasked to 00, 04, or 08 when the transfer of the message is confirmed and the message is routed with class CA to the appropriate next queue (ack wait queue or completed message queue dependent on the type of the ACK).

Connecting Two MERVA ESA Systems

How MERVA Link can be used to exchange data between **two MERVA ESA systems** is described here. The following assumptions are made:

- Each MERVA ESA system runs under CICS.
- MERVA Link uses an APPC connection (LU 6.2).
- MERVA A is the descriptive name of a MERVA ESA system **without a link** to an external network.
- MERVA B is the descriptive name of a MERVA ESA system **with a link** to one or two external networks.

The external networks supported are:

- SWIFT Link
- Telex Link via a fault-tolerant system

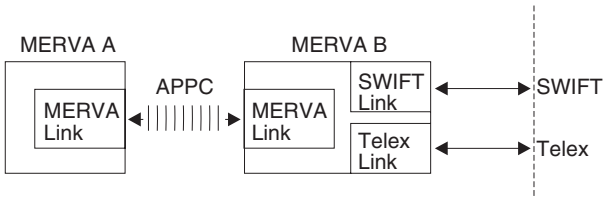

Figure 100. Connecting Two MERVA ESA Systems with MERVA Link

The following restrictions apply:

- SWIFT messages may be transferred by MERVA Link using either the MERVA ESA queue format or the SWIFT format. Either format is specified in the MERVA Link partner table.

  The MERVA ESA queue format is applicable only for the message transfer between two MERVA ESA systems.

  The SWIFT format can be used for the message transfer between two MERVA ESA systems and must be used for the message transfer between a MERVA ESA and MERVA running on a workstation.
- Telex messages can be sent to and received from the Telex network using the Telex Link via a fault-tolerant system. Telex message processing via workstation is not described here. With the appropriate customization, however, this type of telex message processing can be supported as well. Refer to *MERVA Workstation Based Functions* for more information.

The following deals with the message processing for SWIFT You need it only if messages are sent and received over the SWIFT network in your installation. If you are interested in the message processing for the Telex network only, see "Connecting MERVA A to MERVA B with Telex Link via a Fault-Tolerant System" on page 228.

Connecting MERVA A to MERVA B with the SWIFT Link

The following is a description of the message flow of SWIFT messages shown in Figure 100 on page 212:

- SWIFT input messages:
  1. In MERVA A input messages are created, transferred using MERVA Link to MERVA B, and sent over the SWIFT network using the attached SWIFT Link.
  2. The acknowledgments received from the network, are transferred with MERVA Link to MERVA A.
  3. In MERVA A queues, the acknowledged messages are available for further processing.

  In MERVAB, input messages can also be created and sent over the SWIFT network to the same master logical terminal, and acknowledgments can be received. The message creating/receiving process in MERVA B is independent of the process in MERVA A. The distribution of acknowledgments depends on where the input messages were created.
- SWIFT output messages:

  Output messages are received from the SWIFT network by the SWIFT Link in MERVA B, and transferred using the MERVA Link to MERVA A.
- SWIFT network controlling messages:

  The following network controlling messages and their acknowledgments are kept in MERVA B and are not transferred to MERVA A:

- LOGIN
- LOGOUT
- SELECT
- QUIT
- ABORT

Message Processing for SWIFT

Figure 101 shows in detail how messages move from one queue to another in MERVA A.

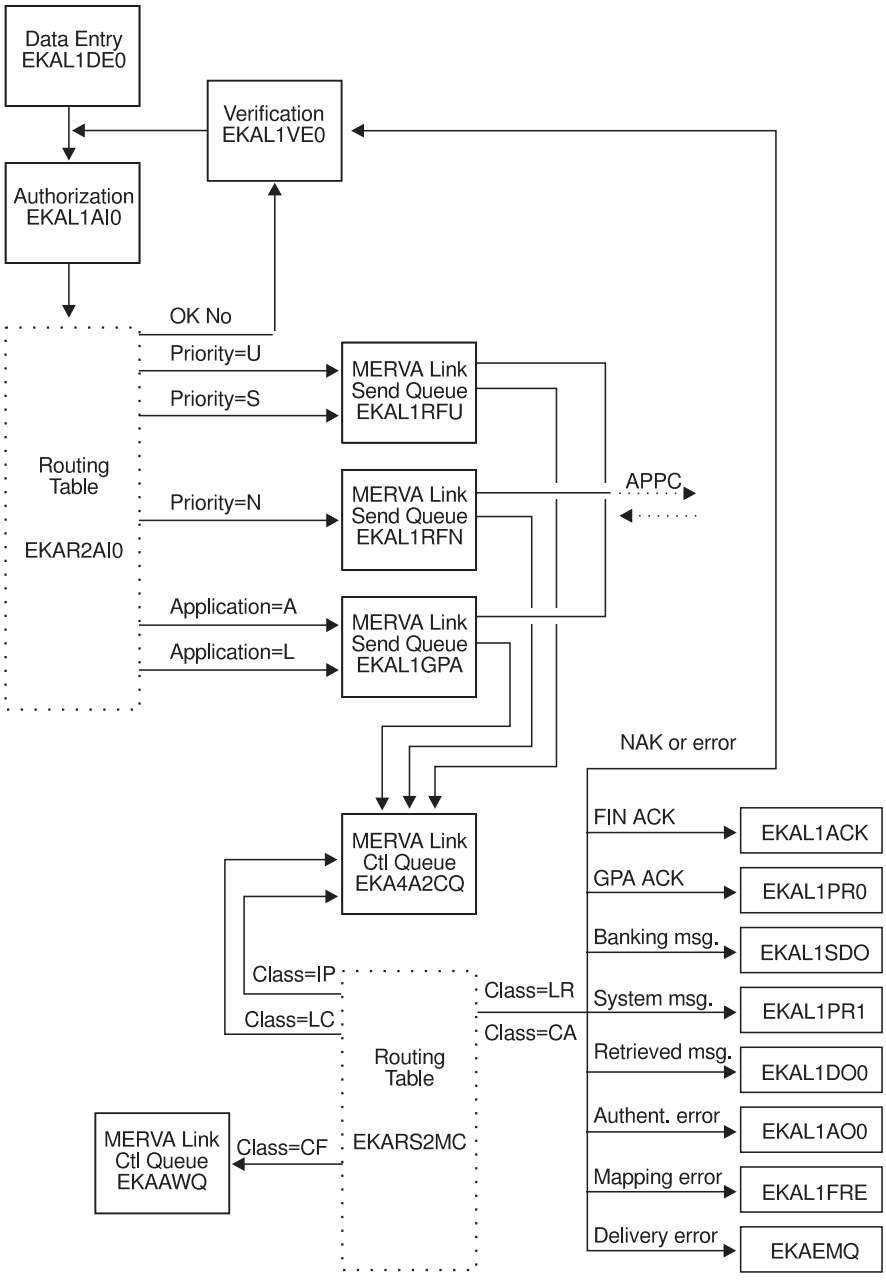

Figure 101. Routing of SWIFT Input and Output Messages in MERVA A

The following comments apply to Figure 101 and Figure 102 on page 216. The numbers shown in brackets in Figure 102 correspond to the numbered notes shown here.

Data Flow and Notes:

[1] SWIFT input messages are created in the data entry queue EKAL1DE0, and authorized in EKAL1AI0. Further processing is now determined by the routing table EKAR2AI0.

[2] If a message cannot be authorized, it is routed by the command **ok no** to the verification queue EKAL1VE0 for correction. Otherwise, it is routed to one of the MERVA Link send queues.

Financial application (FIN) messages are distributed according to their priority. Messages with the priority:

- U or S are moved to the send queue EKAL1RFU
- N are moved to the send queue EKAL1RFN

Application control messages and logical terminal control messages, indicated by A and L respectively, are called general purpose application (GPA) messages and are routed to the send queue EKAL1GPA.

[3] Once in a MERVA Link send queue, the message is transferred by MERVA Link via an APPC connection to MERVA B. In addition, it is copied with the message class IP to the MERVA Link control queue EKA4A2CQ.

[4] MERVA Link in MERVA B informs MERVA Link in MERVA A that the message has been delivered successfully. MERVA Link in MERVA A changes the message class from IP to CF. As an acknowledgment is expected for the delivered message, the confirmed message is routed via the routing table EKARS2MC from the control queue to the acknowledgment wait queue EKAAWQ.

Note: In this sample, each message in queue EKAAWQ has class CF and no MERVA Link receipt return code field EKARECRC. This is because SWIFT input messages only (and no acknowledgments) are sent to and confirmed by MERVA Link in MERVA B.

[5] In MERVA B, the delivered message is sent over the SWIFT network and an acknowledgment received. MERVA Link in MERVA B creates a MERVA Link status report containing acknowledgment information, and transfers it to MERVA Link in MERVA A.

In MERVA A, MERVA Link correlates the confirmed message in queue EKAAWQ with the received status report. MFS user exit EKAMU133 copies the acknowledgment information from the status report to appropriate fields in the correlated message.

MERVA Link assigns class LR to the correlated message and requests routing. Routing table EKARS2MC distributes the message to its final destination queues. Control queue EKA4A2CQ must always be one of the destination queues.

Note: MERVA Link may have assigned class CA to the confirmed and acknowledged message. In this case, control queue EKA4A2CQ **must not** be a destination queue.

[6] If the message either has class CA or class LR combined with the receipt return code field EKARECRC, and the receipt return code starts with 0, the

message contains acknowledgment information. Messages containing an acknowledgment for a FIN message (FIN ACK) are moved to the acknowledgment queue EKAL1ACK. Messages containing acknowledgment for a GPA message (GPA ACK), are routed to the printer queue EKAL1PR0.

If a FIN or GPA message was negatively acknowledged or indicates another error, for example, it could not be sent over the SWIFT network, it is routed to the verification queue EKAL1VE0.

After routing, the confirmed message is deleted from wait queue EKAAWQ.

[7] The queues EKAL1SDO to EKAEMQ contain SWIFT output messages also assigned class LR (but not CA) by the MERVA Link.

[8] MERVA Link in MERVA A informs MERVA Link in MERVA B that the status report or the SWIFT output message has been delivered successfully.

The relevant parts in routing table EKARS2MC for SWIFT input message processing are shown in Figure 102.

```
EKARS2MC DSLROUTE TYPE=DEFINE,FIELD=(CLASS,EKACLASS,,,,,VFIRST)
         DSLROUTE TYPE=DEFINE,FIELD=(RECRC,EKARECRC,,,,,VFIRST)
         DSLROUTE TYPE=DEFINE,FIELD=(ACQNM,EKAACQNM,,,,,VFIRST)
         SPACE
*-------------------------------------------------------------------*
* ROUTE ALL IP (IN PROCESS) MESSAGES AND THE LC CONTROL MESSAGE TO  *
* THE APPLICATION CONTROL QUEUE.                                    *
* THIS PART MAY CORRECT A HANDLING ERROR OF THE SYSTEM ADMINISTRATOR. *
*-------------------------------------------------------------------*
         DSLROUTE TYPE=TEST,COND=(CLASS,'IP',EQ),FALSE=TSTLC
         DSLROUTE TYPE=SET,TARGET=ACQNM,GOTO=END
TSTLC    DSLROUTE TYPE=TEST,COND=(CLASS,'LC',EQ),FALSE=TSTCF
         DSLROUTE TYPE=SET,TARGET=ACQNM,GOTO=END
         SPACE
*-------------------------------------------------------------------*
* TEST FOR AND ROUTE CONFIRMED MSG'S (CLASS=CF) TO THE ACK WAIT QUEUE.*
*-------------------------------------------------------------------*
TSTCF    DSLROUTE TYPE=TEST,COND=(CLASS,'CF',EQ),FALSE=TSTCA            [4]
         DSLROUTE TYPE=SET,TARGET='EKAAWQ',GOTO=END
         SPACE
*-------------------------------------------------------------------*
* TEST FOR AND ROUTE CONFIRMED AND ACKNOWLEDGED MSG'S (CLASS=CA).   *
* ROUTE A FINALLY ACKED MSG (RECRC=00 OR RECRC=08) ACCORDING TO THE *
* ROUTING LOGIC OF MERVA STANDALONE SAMPLE L1 ORGANIZATION,         *
* LABEL TMSGACK.                                                    *
*-------------------------------------------------------------------*
TSTCA    DSLROUTE TYPE=TEST,COND=(CLASS,'CA',EQ),TRUE=TMSGACK,       * [6]
               FALSE=TSTLR
         SPACE
*-------------------------------------------------------------------*
* TEST FOR AND ROUTE LR (INBOUND) MESSAGES. ROUTE A FINALLY ACKED   *
* MSG (RECRC=00 OR RECRC=08) ACCORDING TO THE ROUTING LOGIC OF MERVA *
* STANDALONE SAMPLE L1 ORGANIZATION, LABEL TMSGACK.                 *
* ...                                                               *
* ...                                                               *
*-------------------------------------------------------------------*
* NOTE THAT ALL INBOUND MESSAGES MUST BE ROUTED TO THE CONTROL QUEUE *
* FOR MESSAGE INTEGRITY CONTROL PURPOSES.                           *
*-------------------------------------------------------------------*
TSTLR    DSLROUTE TYPE=TEST,COND=(CLASS,'LR',EQ),FALSE=TSTRC            [5]
*
*        RECRC=04 REFERS TO NOT FINALLY ACKED MSGS
*        THIS RECRC IS NOT SET IN THE SAMPLE USER EXIT
*
         DSLROUTE TYPE=TEST,COND=(RECRC,'04',EQ),TRUE=SETLRA
*
         DSLROUTE TYPE=TEST,COND=(RECRC,'0',EQ,SHORT),TRUE=TMSGACK      [6]

⋮

SETLRA   ...

⋮
```

Figure 102. Example of Routing Table EKARS2MC for SWIFT Input Message Processing (Part 1 of 2)

```
*
*--------------------------------------------------------------------*
*        ROUTE SWIFT INPUT MESSAGES WITH ACK INFORMATION EITHER
*        FROM SWIFT OR FROM SWIFT LINK ERROR PROCESSING
*--------------------------------------------------------------------*
*        DEFINE THE DIAGNOSTIC FROM THE MSGACK FIELD
*        ROUTING LOGIC FROM DWSL1IN
TMSGACK  DSLROUTE TYPE=DEFINE,FIELD=(MSGACK,MSGACK,,,,,VFIRST),        *
              NOTFND=VE0
*
*        CHECK FOR AN ERROR MESSAGE STARTING WITH 'DWS'
         DSLROUTE TYPE=TEST,COND=(MSGACK,'DWS',EQ,SHORT),TRUE=VE0
*
*        CHECK FOR THE BASIC HEADER OF APDU ID 21 (ACK OR NAK)
*        THE LAYOUT OF APDU ID 21 IS THE SAME FOR GPA AND FIN:
*        {1:A21TIBMDEPABXXX0001000001} ◄== BASIC HEADER
*        {4:{177:8906271013}{451:0}}   ◄== TEXT BLOCK
*
         DSLROUTE TYPE=TEST,COND=(MSGACK,'{1:',EQ,SHORT),FALSE=VE0
*
*        DEFINE FIELD 451 FROM MSGACK CONTAINING '0' FOR AN ACK
*        OR '1' FOR A NAK

         DSLROUTE TYPE=DEFINE,FIELD=(F451,MSGACK,,,,,VFIRST),          *
               DISP=53,LENGTH=1,EMPTY=VE0,NOTFND=VE0
*
         DSLROUTE TYPE=TEST,COND=(F451,'0',EQ),FALSE=VE0
*
*        SEPARATE GPA FROM FIN ACKNOWLEDGMENTS
         DSLROUTE TYPE=DEFINE,FIELD=(APPL,SWBHAPI,,,,,VFIRST)
         DSLROUTE TYPE=TEST,COND=(APPL,'F',EQ),TRUE=ACK,FALSE=PR00
*
*        ROUTE POSITIVE GPA ACKNOWLEDGMENTS TO THE PRINTER               [6]
PR00     DSLROUTE TYPE=SET,TARGET=('EKAL1PR0'),GOTO=CHKCA
*
*        ROUTE POSITIVE FIN ACKNOWLEDGMENT TO EKAL1ACK                   [6]
ACK      DSLROUTE TYPE=SET,TARGET=('EKAL1ACK'),GOTO=CHKCA
*
*        ROUTE ERRORS AND NEGATIVE ACKNOWLEDGMENTS TO EKAL1VE0           [6]
VE0      DSLROUTE TYPE=SET,TARGET=('EKAL1VE0'),GOTO=CHKCA
*
*        ROUTE TO CONTROL QUEUE FOR MESSAGE CLASS 'LR' ONLY.
CHKCA    DSLROUTE TYPE=TEST,COND=(CLASS,'CA',EQ),TRUE=END
         DSLROUTE TYPE=SET,TARGET=ACQNM,GOTO=END                         [5]

⋮

TSTRC    ...

⋮

END      DSLROUTE TYPE=FINAL
         END
```

Figure 102. Example of Routing Table EKARS2MC for SWIFT Input Message Processing (Part 2 of 2)

The routing of a SWIFT input message through MERVA B is shown below.

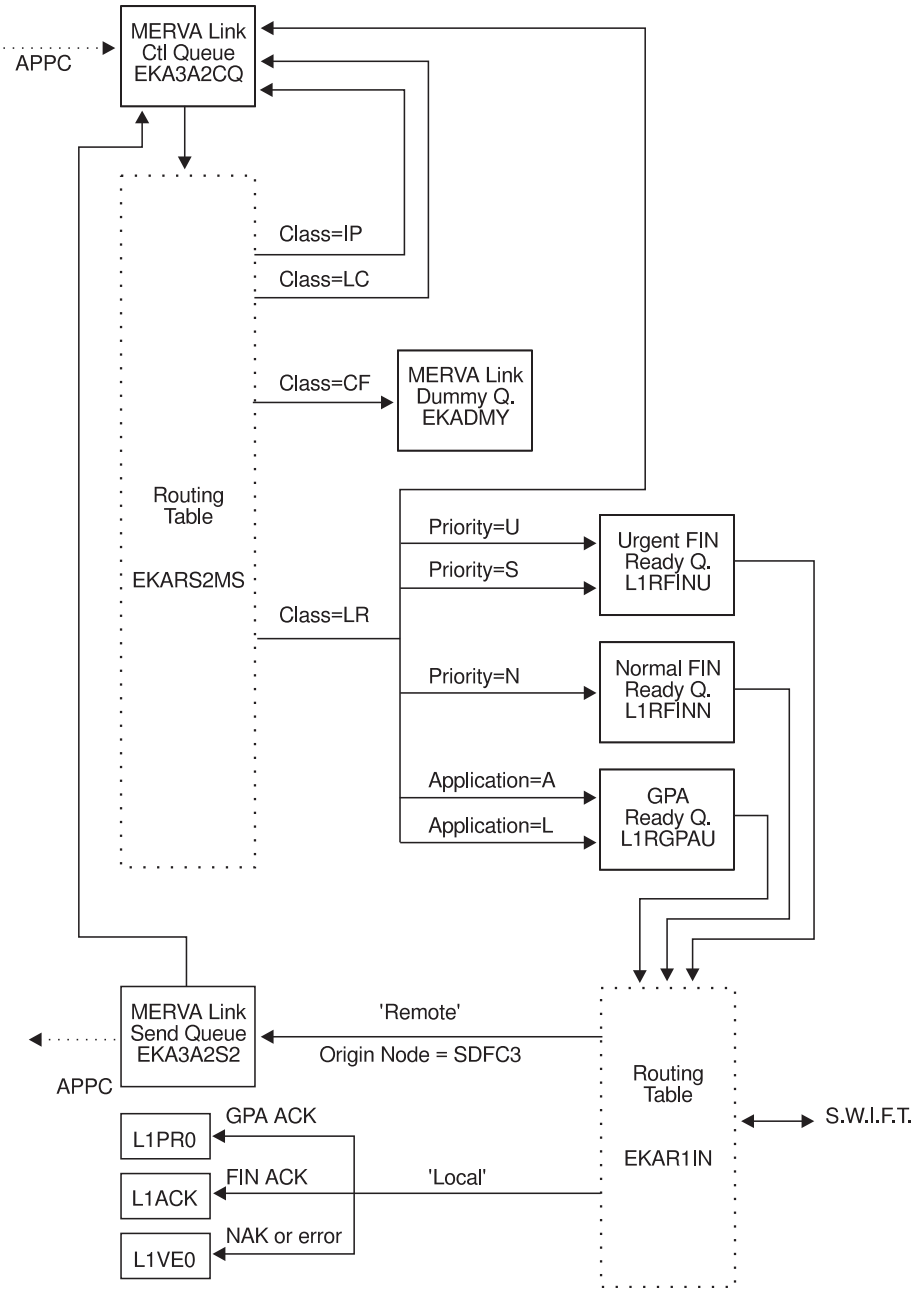

Figure 103. Routing of SWIFT Input Messages in MERVA B

The following comments apply to both Figure 103 and Figure 104.

Data Flow and Notes:

[1] MERVA Link receives the SWIFT input message and assigns the message class LR to it. The same distribution scheme applies in the routing table EKARS2MS as in the routing table EKAR2AI0 in MERVA A.

Financial application (FIN) messages are distributed according to their priority. Messages with the priority:

- U or S are moved to the urgent FIN ready queue L1RFINU
- N are moved to the normal Fin ready queue L1RFINN

Application control messages and logical terminal control messages, indicated by A and L respectively, are called general purpose application (GPA) messages and are routed to the GPA ready queue L1RGPAU. Each message is also routed to the MERVA Link control queue EKA3A2CQ.

[2] The message is sent to the SWIFT network and an acknowledgment is received. Queues L1RFINU, L1RFINN, and L1RGPAU may also contain SWIFT input messages created locally in MERVA B.

[3] The routing table EKAR1IN determines whether a message containing any kind of acknowledgment (ACK, NAK, or error) is routed to queues in MERVA B, or transferred to MERVA Link in MERVA A. If the MERVA Link control field EKAONODE (Origin Node) exists and contains SDFC3, the message is routed to the MERVA Link send queue EKA3A2S2.

[4] However, the complete message is not transferred. Instead, the MFS user exit EKAMU133 provides the information for a MERVA Link status report consisting of:

- Acknowledgment (field MSGACK)
- Basic header (field SWBH)
- Application header (field SWAH)
- Trailer(s) (field SWTRAIL)
- Receipt Return Code (field EKARECRC)

The receipt return code is 00 if the field MSGACK contains a positive acknowledgment, otherwise it is 08. The status report is copied with message class IP to the MERVA Link control queue EKA3A2CQ and transferred to MERVA Link in MERVA A.

[5] MERVA Link in MERVA A informs MERVA Link in MERVA B that the status report has been delivered successfully.

MERVA Link in MERVA B changes the message class from IP to CF and routes the status report via the routing table EKARS2MS to the dummy queue EKADMY. Routing to this queue has the same effect as a **delete** command.

These steps correspond to the labels in Figure 104.

```
EKARS2MS DSLROUTE TYPE=DEFINE,FIELD=(CLASS,EKACLASS,,,,,VFIRST)
         DSLROUTE TYPE=DEFINE,FIELD=(ACQNM,EKAACQNM,,,,,VFIRST)
         SPACE
*-------------------------------------------------------------------*
* ROUTE ALL IP (IN PROCESS) MESSAGES AND THE LC CONTROL MESSAGE TO   *
* THE APPLICATION CONTROL QUEUE.                                    *
* THIS PART MAY CORRECT A HANDLING ERROR OF THE SYSTEM ADMINISTRATOR.*
*-------------------------------------------------------------------*
         DSLROUTE TYPE=TEST,COND=(CLASS,'IP',EQ),FALSE=TSTLC
         DSLROUTE TYPE=SET,TARGET=ACQNM,GOTO=END
TSTLC    DSLROUTE TYPE=TEST,COND=(CLASS,'LC',EQ),FALSE=TSTCF
         DSLROUTE TYPE=SET,TARGET=ACQNM,GOTO=END
         SPACE
*-------------------------------------------------------------------*
* TEST FOR AND ROUTE CONFIRMED MSG'S (CLASS=CF). DELETE THE MERVA LINK*
* STATUS REPORT AND SWIFT OUTPUT MESSAGE                            *
*-------------------------------------------------------------------*
TSTCF    DSLROUTE TYPE=TEST,COND=(CLASS,'CF',EQ),FALSE=TSTLR              [5]
         DSLROUTE TYPE=SET,TARGET='EKADMY',GOTO=END
         SPACE
*-------------------------------------------------------------------*
* TEST FOR AND ROUTE LR (INBOUND) MESSAGES. ROUTE SWIFT INPUT       *
* MESSAGES TO THE APPROPRIATE SWIFT READY QUEUES. THE ROUTING       *
* LOGIC OF THE SAMPLE ROUTING MODULE DWSL1AI0 IS APPLIED.           *
*-------------------------------------------------------------------*
* NOTE THAT ALL INBOUND MESSAGES MUST BE ROUTED TO THE CONTROL QUEUE *
* FOR MESSAGE INTEGRITY CONTROL PURPOSES.                           *
*-------------------------------------------------------------------*
TSTLR    DSLROUTE TYPE=TEST,COND=(CLASS,'LR',EQ),FALSE=TSTRC              [1]
         DSLROUTE TYPE=DEFINE,FIELD=(APPL,SWBHAPI,,,,,VFIRST),        *
               EMPTY=ERR,NOTFND=ERR
*
*        TEST FOR F OR A OR L IN APPLICATION HEADER
         DSLROUTE TYPE=TEST,COND=(APPL,'F',EQ),TRUE=FIN
         DSLROUTE TYPE=TEST,COND=(APPL,'A',EQ),TRUE=GPA
         DSLROUTE TYPE=TEST,COND=(APPL,'L',EQ),TRUE=GPA,FALSE=ERR
*
*        SET THE READY QUEUE FOR GPA
GPA      DSLROUTE TYPE=SET,TARGET=('L1RGPAU')                             [2]
         DSLROUTE TYPE=SET,TARGET=ACQNM,GOTO=END                          [1]
*
*        DEFINE PRTY FROM THE FIELD SWAHIPY (PRIORITY OF APPL HEADER)
FIN      DSLROUTE TYPE=DEFINE,FIELD=(PRTY,SWAHIPY,,,,,VFIRST),        *
               EMPTY=ERR,NOTFND=ERR
*
*        TEST FOR N OR U OR S PRIORITY
         DSLROUTE TYPE=TEST,COND=(PRTY,'N',EQ),TRUE=FINN
         DSLROUTE TYPE=TEST,COND=(PRTY,'U',EQ),TRUE=FINU
         DSLROUTE TYPE=TEST,COND=(PRTY,'S',EQ),TRUE=FINU,FALSE=ERR
*
*        NORMAL READY QUEUE FOR FINANCIAL APPLICATION
FINN     DSLROUTE TYPE=SET,TARGET=('L1RFINN')                             [2]
         DSLROUTE TYPE=SET,TARGET=ACQNM,GOTO=END                          [1]
```

Figure 104. Example of Routing Table EKARS2MS for SWIFT Input Message Processing (Part 1 of 2)

```
*
*        URGENT READY QUEUE FOR FINANCIAL APPLICATION
FINU     DSLROUTE TYPE=SET,TARGET=('L1RFINU')                              [2]
         DSLROUTE TYPE=SET,TARGET=ACQNM,GOTO=END                           [1]
*        MERVA LINK ERROR QUEUE
ERR      DSLROUTE TYPE=SET,TARGET=('EKAEMQ')
         DSLROUTE TYPE=SET,TARGET=ACQNM,GOTO=END                           [1]

⋮

TSTRC    ...

⋮

END      DSLROUTE TYPE=FINAL
         END
```

Figure 104. Example of Routing Table EKARS2MS for SWIFT Input Message Processing (Part 2 of 2)

The routing table EKAR1IN, shown in Figure 105, handles the distribution of messages following the receipt of an acknowledgment from the SWIFT network.

```
*************************************************************************
*        ROUTE INPUT MESSAGES OF GPA AND FIN
*          OF A LOGICAL TERMINAL
*************************************************************************
*        DEFINE THE BASIC HEADER FOR THE ROUTING TRACE
EKAR1IN  DSLROUTE TYPE=DEFINE,FIELD=(BASHEAD,SWBH,,,,,VFIRST)
*
*        DEFINE THE APDU IDENTIFIER
         DSLROUTE TYPE=DEFINE,FIELD=(APDUID,SWBHAPDU,,,,,VFIRST),      *
               EMPTY=PR0
*        APDU ID 01 CONTAINS AN APPLICATION HEADER WITH A MESSAGE TYPE
*        ALL OTHER APDU IDS ARE GENERATED BY DWSDGPA AND ARE ROUTED
*        TO L1PR0 WHERE THEY ARE PRINTED IN SEQUENCE WITH THEIR ACKS
         DSLROUTE TYPE=TEST,COND=(APDUID,'01',EQ),TRUE=MSGACK,         *  [1]
               FALSE=PR0
*
*************************************************************************
*        PROCESSING OF APDU 01
*************************************************************************
*
*        DEFINE THE DIAGNOSTIC FROM THE MSGACK FIELD
*        USE 32 BYTES TO SEE THE KIND OF ERROR IN THE ROUTING TRACE
MSGACK   DSLROUTE TYPE=DEFINE,FIELD=(MSGACK,MSGACK,,,,,VFIRST),        *
               NOTFND=PR0
*        TEST FOR MERVA LINK ORIGIN NODES
*        USE THE 'SHORT' MODIFIER
*        IF FOUND SEND MESSAGE TO MERVA -CREATE- VIA ML SEND QUEUE
         DSLROUTE TYPE=DEFINE,FIELD=(ON,EKAONODE,,,,,VFIRST),          *  [2]
               NOTFND=DWS
         DSLROUTE TYPE=TEST,COND=(ON,'SDFC3',EQ,SHORT),TRUE=SETC3,     *
               FALSE=DWS
SETC3    DSLROUTE TYPE=SET,TARGET='EKA3A2S2',GOTO=END
```

Figure 105. Example of GPA and FIN Routing of SWIFT Input Messages Using the Routing Table EKAR1IN (Part 1 of 2)

```
*
*        CHECK FOR AN ERROR MESSAGE STARTING WITH 'DWS'
*        ROUTE MSG ORIGINATING FROM MERVA -SEND-
DWS      DSLROUTE TYPE=TEST,COND=(MSGACK,'DWS',EQ,SHORT),TRUE=VE0         [3]
*
*        CHECK FOR THE BASIC HEADER OF APDU ID 21 (ACK OR NAK)
*        THE LAYOUT OF APDU ID 21 IS THE SAME FOR GPA AND FIN:
*        {1:A21TIBMDEPABXXX0001000001} ◄== BASIC HEADER
*        {4:{177:8906271013}{451:0}}   ◄== TEXT BLOCK
*
         DSLROUTE TYPE=TEST,COND=(MSGACK,'{1:',EQ,SHORT),FALSE=VE0
*
*        DEFINE FIELD 451 FROM MSGACK CONTAINING '0' FOR AN ACK
*        OR '1' FOR A NAK
         DSLROUTE TYPE=DEFINE,FIELD=(F451,MSGACK,,,,,VFIRST),          *
               DISP=53,LENGTH=1,EMPTY=VE0,NOTFND=VE0
*
         DSLROUTE TYPE=TEST,COND=(F451,'0',EQ),FALSE=VE0
*
*        SEPARATE GPA FROM FIN ACKNOWLEDGMENTS
         DSLROUTE TYPE=DEFINE,FIELD=(APPL,SWBHAPI,,,,,VFIRST)
         DSLROUTE TYPE=TEST,COND=(APPL,'F',EQ),TRUE=ACK,FALSE=PR0
*
*        ROUTE POSITIVE GPA ACKNOWLEDGMENTS TO THE PRINTER
PR0      DSLROUTE TYPE=SET,TARGET=('L1PR0'),GOTO=END
*
*        ROUTE POSITIVE FIN ACKNOWLEDGMENT TO L1ACK
ACK      DSLROUTE TYPE=SET,TARGET=('L1ACK'),GOTO=END
*
*        ROUTE ERRORS AND NEGATIVE ACKNOWLEDGMENTS TO L1VE0
VE0      DSLROUTE TYPE=SET,TARGET=('L1VE0'),GOTO=END
*
*        IN CASE OF ROUTING ERRORS THE MESSAGE IS ROUTED TO L1VE0
END      DSLROUTE TYPE=FINAL,TARGET='L1VE0'
         END
```

Figure 105. Example of GPA and FIN Routing of SWIFT Input Messages Using the Routing Table EKAR1IN (Part 2 of 2)

Notes:

[1] Network controlling messages (LOGIN, LOGOUT, SELECT, QUIT, and ABORT) are kept in MERVA B. Their APDUID is different from 01.

[2] Messages containing the field MSGACK may have been created remotely in MERVA A, or locally in MERVA B. MERVA Link indicates the origin of the message by the field EKANONODE. If this field exists and contains SDFC3, the message is routed to the MERVA Link send queue.

The message origin is defined in the MERVA Link partner table in MERVA A.

[3] Specifies the routing of locally created messages.

Routing of SWIFT Output Messages

Figure 106 shows the routing of SWIFT output messages.
Positive and negative acknowledgments (ACKs and NAKs) for the network

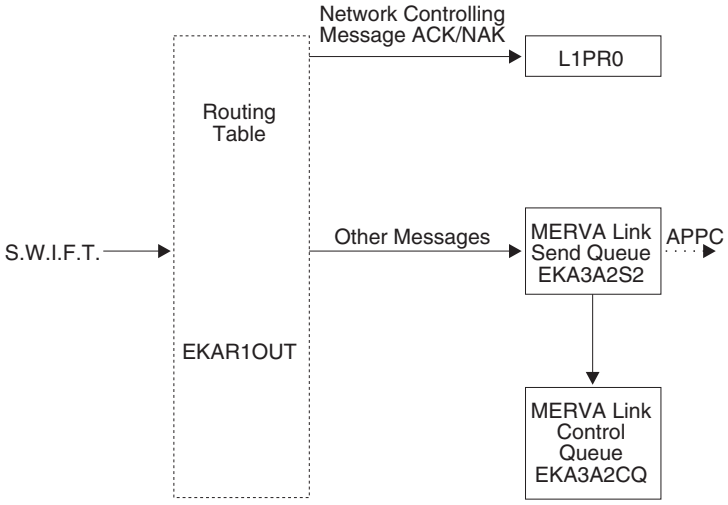

Figure 106. Routing of SWIFT Output Messages in MERVA B

controlling messages LOGIN, LOGOUT, SELECT, QUIT, and ABORT are routed via routing table EKAR1OUT (see Figure 107 on page 224) to the printer queue L1PR0. Other output messages received from the SWIFT network are routed to the MERVA Link send queue EKA3A2S2 for transfer to MERVA Link in MERVA A.

```
*        DEFINE THE BASIC HEADER AND MESSAGE ID FOR THE ROUTING TRACE
EKAR1OUT DSLROUTE TYPE=DEFINE,FIELD=(BASHEAD,SWBH,,,,,VFIRST)
         DSLROUTE TYPE=DEFINE,FIELD=(MSGID,DSLEXIT,,,,,VFIRST)
*********************************************************************
*   MSGS WITH AN UNIDENTIFIED MESSAGE TYPE ARE SENT TO REMOTE CICS
*********************************************************************
*        CHECK FOR FREE FORMAT MESSAGE
         DSLROUTE TYPE=TEST,COND=(MSGID,'0DSL    ',EQ),TRUE=REM
*        CHECK FOR FORMATTING ERRORS IN MESSAGE
         DSLROUTE TYPE=DEFINE,FIELD=(ERROR,DSLLFBUF,,,,,VFIRST),       *
               FOUND=PR0,DISP=0,LENGTH=1
*********************************************************************
*        LAK, LNK, SAK, and SNK (LOGIN/SELECT ACKs and NAKs), LOGOUT,
*        QUIT, and ABORT ACKs are  routed to
*        the local L1PR0, where they can be printed in sequence with
*        the related LOGIN, LOGOUT, SELECT, and QUIT messages.
*        SG22, SG23, SG42, SG43, SG26, SF25, SG13, SG15
*********************************************************************
         DSLROUTE TYPE=TEST,COND=(MSGID,'SG22    ',EQ),TRUE=PR0
         DSLROUTE TYPE=TEST,COND=(MSGID,'SG23    ',EQ),TRUE=PR0
         DSLROUTE TYPE=TEST,COND=(MSGID,'SG42    ',EQ),TRUE=PR0
         DSLROUTE TYPE=TEST,COND=(MSGID,'SG43    ',EQ),TRUE=PR0
         DSLROUTE TYPE=TEST,COND=(MSGID,'SG26    ',EQ),TRUE=PR0
         DSLROUTE TYPE=TEST,COND=(MSGID,'SF25    ',EQ),TRUE=PR0
         DSLROUTE TYPE=TEST,COND=(MSGID,'SG13    ',EQ),TRUE=PR0
         DSLROUTE TYPE=TEST,COND=(MSGID,'SG15    ',EQ),TRUE=PR0
*********************************************************************
*        ALL  OTHER OUTPUT MESSAGES ARE SENT TO MERVA -CREATE-
*********************************************************************
*
REM      DSLROUTE TYPE=SET,TARGET=('EKA3A2S2'),GOTO=END
*
*        SYSTEM LAKs, ACKs ARE PRINTED ON LOCAL L1PR0
PR0      DSLROUTE TYPE=SET,TARGET=('L1PR0'),GOTO=END
*
*        IN CASE OF ROUTING ERRORS THE MESSAGE IS ROUTED TO L1PR1
END      DSLROUTE TYPE=FINAL,TARGET='L1PR1'
         END
```

Figure 107. Example of the Routing Table EKAR1OUT

Messages in the send queue are copied with class IP to control queue EKA3A2CQ. MERVA Link in MERVA A informs MERVA Link in MERVA B that the message has been delivered successfully. Figure 103 on page 218 shows the further processing. MERVA Link changes the message class from IP to CF, and routes the message via the routing table EKARS2MS to the dummy queue EKADMY. Routing to this queue has the same effect as a **delete** command.

Figure 101 on page 213 shows how SWIFT output messages are handled in MERVA A. MERVA Link in MERVA A receives a message and assigns the message class LR to it. The message is then distributed to its destination queue according to the routing table EKARS2MC. Figure 108 shows the relevant parts in the routing table EKARS2MC for SWIFT output message processing.

```
EKARS2MC DSLROUTE TYPE=DEFINE,FIELD=(CLASS,EKACLASS,,,,,VFIRST)
         DSLROUTE TYPE=DEFINE,FIELD=(RECRC,EKARECRC,,,,,VFIRST)
         DSLROUTE TYPE=DEFINE,FIELD=(DELRC,EKADELRC,,,,,VFIRST)
         DSLROUTE TYPE=DEFINE,FIELD=(ACQNM,EKAACQNM,,,,,VFIRST)
         .
         .
         .
*-------------------------------------------------------------------*
* TEST FOR AND ROUTE LR (INBOUND) MESSAGES. ...                     *
* ...                                                               *
* ...                                                               *
* ROUTE AN INBOUND MESSAGE (SWIFT OUTPUT MSG) WHICH INDICATES A     *
* DELIVER ERROR (DELRC!=00) TO THE RECEIVED ERRONEOUS MESSAGE       *
* QUEUE (EKAEMQ) AND TO THE CONTROL QUEUE. ROUTE AN INBOUND MESSAGE *
* (SWIFT OUTPUT MSG) WHICH INDICATES NO DELIVER ERROR ACCORDING TO  *
* THE ROUTING LOGIC OF MERVA STANDALONE SAMPLE L1 ORGANIZATION,     *
* LABEL SETLRI.                                                     *
*-------------------------------------------------------------------*
* NOTE THAT ALL INBOUND MESSAGES MUST BE ROUTED TO THE CONTROL QUEUE *
* FOR MESSAGE INTEGRITY CONTROL PURPOSES.                           *
*-------------------------------------------------------------------*
TSTLR    DSLROUTE TYPE=TEST,COND=(CLASS,'LR',EQ),FALSE=TSTRC

⋮

         DSLROUTE TYPE=TEST,COND=(RECRC,'0',EQ,SHORT),TRUE=TMSGACK        [1]
         DSLROUTE TYPE=TEST,COND=(DELRC,'00',EQ),TRUE=SETLRI              [2]
         DSLROUTE TYPE=SET,TARGET='EKAEMQ'
         DSLROUTE TYPE=SET,TARGET=ACQNM,GOTO=END
*-------------------------------------------------------------------*
*        ROUTE SWIFT OUTPUT MESSAGES (GPA AND FIN)
*-------------------------------------------------------------------*
*        DEFINE THE BASIC HEADER AND MESSAGE ID FOR THE ROUTING TRACE
SETLRI   DSLROUTE TYPE=DEFINE,FIELD=(BASHEAD,SWBH,,,,,VFIRST)
         DSLROUTE TYPE=DEFINE,FIELD=(MSGID,DSLEXIT,,,,,VFIRST)
*********************************************************************
*        MSGS WITH AN UNIDENTIFIED MESSAGE TYPE ARE ROUTED TO EKAL1FRE
*********************************************************************
*        CHECK FOR FREE FORMAT MESSAGE
         DSLROUTE TYPE=TEST,COND=(MSGID,'0DSL    ',EQ),TRUE=FREE          [3]
*        CHECK FOR FORMATTING ERRORS IN MESSAGE
         DSLROUTE TYPE=DEFINE,FIELD=(ERROR,DSLLFBUF,,,,,VFIRST),      *
               FOUND=FREE,DISP=0,LENGTH=1
*********************************************************************
*        MESSAGES AND REPORTS CONTAINING DELIVERY AND NON-DELIVERY
*        INFORMATION ARE SEPARATED FROM THE REST AND ROUTED TO
*        EKAL1DO0 (DISTRIBUTION OUTPUT) FOR FURTHER PROCESSING.
*        SF010 (non-delivery warning)
*        SF011 (delivery notification)
*        SF015 (delayed NAK)
*        SF066 (undelivered message solicited report)
*        SF082 (undelivered message report at fixed hour)
*        SF083 (undelivered message report at cut-off time)
*********************************************************************
```

Figure 108. Example of Routing Table EKARS2MC for SWIFT Output Message Processing (Part 1 of 3)

```
         DSLROUTE TYPE=TEST,COND=(MSGID,'SF010   ',EQ),TRUE=DO0           [4]
         DSLROUTE TYPE=TEST,COND=(MSGID,'SF011   ',EQ),TRUE=DO0
         DSLROUTE TYPE=TEST,COND=(MSGID,'SF015   ',EQ),TRUE=DO0
         DSLROUTE TYPE=TEST,COND=(MSGID,'SF066   ',EQ),TRUE=DO0
         DSLROUTE TYPE=TEST,COND=(MSGID,'SF082   ',EQ),TRUE=DO0
         DSLROUTE TYPE=TEST,COND=(MSGID,'SF083   ',EQ),TRUE=DO0
*
*        DEFINE MSGTYPE FROM THE MESSAGE TYPE FIELD (FIRST NESTING ID)
*        IF NOT FOUND, IT IS AN APDU DIFFERENT FROM 01 WHICH IS PRINTED
         DSLROUTE TYPE=DEFINE,FIELD=(MSGTYPE,SWAHMT,,,,,VFIRST),       *
               EMPTY=PR1,NOTFND=PR1                                       [5]
*
*        IF THE MESSAGE TYPE IS 021, IT IS A RETRIEVED MESSAGE
*          EKAL1DO0 MUST LOOK IF IT INCLUDES A BANKING MESSAGE
*        IF THE MESSAGE CATEGORY IS ZERO, IT IS A SYSTEM MESSAGE
*          WHICH IS PRINTED VIA L1PR1
*        USE THE 'SHORT' MODIFIER TO TEST THE FIRST BYTE OF MSGTYPE
         DSLROUTE TYPE=TEST,COND=(MSGTYPE,'021',EQ),TRUE=DO0              [4]
         DSLROUTE TYPE=TEST,COND=(MSGTYPE,'0',EQ,SHORT),TRUE=PR1          [5]
*
*        DEFINE AUT FROM THE MSGACK FIELD WHICH CONTAINS THE RESULT
*        OF THE AUTHENTICATION AS A MESSAGE DWS7XXI ....
         DSLROUTE TYPE=DEFINE,FIELD=(AUT,MSGACK,,,,,VFIRST),           *  [6]
               DISP=0,LENGTH=6,EMPTY=AO0,NOTFND=AO0
*
*        'DWS765' MEANS: AUTHENTICATION SUCCESSFUL WITH PRIMARY KEY
*        'DWS766' MEANS: MESSAGE NOT TO BE AUTHENTICATED
*        ANY OTHER AUTHENTICATION RESULT WILL BE ROUTED TO EKAL1AO0
         DSLROUTE TYPE=TEST,COND=(AUT,'DWS765',EQ),TRUE=MSGERR
         DSLROUTE TYPE=TEST,COND=(AUT,'DWS766',EQ),TRUE=MSGERR,        *
               FALSE=AO0
*
*        DEFINE MSGERR FROM MSGTRERR (ERROR CODE OF THE MESSAGE TRACE
*        FIELD)
*        MSGTRERR MUST ALWAYS BE FOUND AND NOT EMPTY
MSGERR   DSLROUTE TYPE=DEFINE,FIELD=(MSGERR,MSGTRERR,,,,,VFIRST,       *  [4]
               LASTDA),NOTFND=DO0,EMPTY=DO0
*
*        MSGTRERR IS NOT '0000' IF THERE ARE ERRORS IN THE MESSAGE
         DSLROUTE TYPE=TEST,COND=(MSGERR,'0000',EQ),FALSE=DO0
*
*        BANKING MESSAGES WITHOUT ERRORS                                  [7]
         DSLROUTE TYPE=SET,TARGET=('EKAL1SDO'),GOTO=CTL
*
*        SYSTEM MESSAGES ARE PRINTED ON EKAL1PR1
PR1      DSLROUTE TYPE=SET,TARGET=('EKAL1PR1'),GOTO=CTL                   [5]
*
*        MESSAGES WITH AUTHENTICATION ERRORS
AO0      DSLROUTE TYPE=SET,TARGET=('EKAL1AO0'),GOTO=CTL                   [6]
```

Figure 108. Example of Routing Table EKARS2MC for SWIFT Output Message Processing (Part 2 of 3)

```
*
*        MESSAGES WITH UNIDENTIFIED MESSAGE TYPE AND MAPPING ERRORS
FREE     DSLROUTE TYPE=SET,TARGET=('EKAL1FRE'),GOTO=CTL                  [3]
*
*        DISTRIBUTION OUTPUT FOR DELIVERY AND NON-DELIVERY INFORMATION,
*        FOR RETRIEVED MESSAGES AND ERRONEOUS BANKING MESSAGES
DO0      DSLROUTE TYPE=SET,TARGET=('EKAL1DO0'),GOTO=CTL                  [4]
*
CTL      DSLROUTE TYPE=SET,TARGET=ACQNM,GOTO=END                         [8]

⋮

TMSGACK  ...

⋮

TSTRC    ...

⋮

END      DSLROUTE TYPE=FINAL
         END
```

Figure 108. Example of Routing Table EKARS2MC for SWIFT Output Message Processing (Part 3 of 3)

Notes:

[1] A SWIFT output message does not contain the receipt return code field EKARECRC as this field is set by MFS user exit EKAMU133 for a status report only. You can differentiate between an acknowledged SWIFT input message with a receipt return code field and a SWIFT output message without a receipt return code field.

[2] If MERVA Link detected an error concerning the delivery of the message indicated by a nonzero MERVA Link control field EKADELRC, the queue EKAEMQ contains the erroneous message.

[3] Messages with a mapping error are routed to the queue EKAL1FRE.

[4] Messages containing delivery and nondelivery information, retrieved messages and erroneous banking messages are routed to the queue EKAL1DO0.

[5] System messages are routed to printer queue EKAL1PR1.

[6] Messages with an authentication error are routed to the queue EKAL1AO0.

[7] Correct banking messages are routed to the queue EKAL1SDO.

[8] Each message is also routed to the MERVA Link control queue EKA4A2CQ.

Connecting MERVA A to MERVA B with Telex Link via a Fault-Tolerant System

The connecting of MERVA A to MERVA B with Telex Link via a fault-tolerant system is explained here. The message flow and the routing table involved are described in detail.

Message Flow

- Outgoing telex messages:
  1. Telex messages are created in MERVA A, transferred using MERVA Link to MERVA B, and then sent to the telex network.
  2. Acknowledgments received from the network are transferred by MERVA Link back to MERVA A.
  3. The acknowledged messages are available in MERVA A for further processing.

  Telex messages can also be created in MERVA B, sent over the telex network, and acknowledged independently of MERVA Link and the MERVA A system.
- Incoming telex messages:

  Messages received from the telex network are routed to the MERVA B TXRCV queue. You can route telex messages for MERVA A to the MERVA Link send queue for transfer to the MERVA A system.
- Telex network control:

  The telex network is controlled exclusively from MERVA B. Telex messages transferred from MERVA A are held in the telex ready queues in MERVA B if the connection to the telex network has not been initiated.

Routing of Telex Messages in MERVA A

Figure 109 shows in detail the movement of messages between queues in MERVA A.

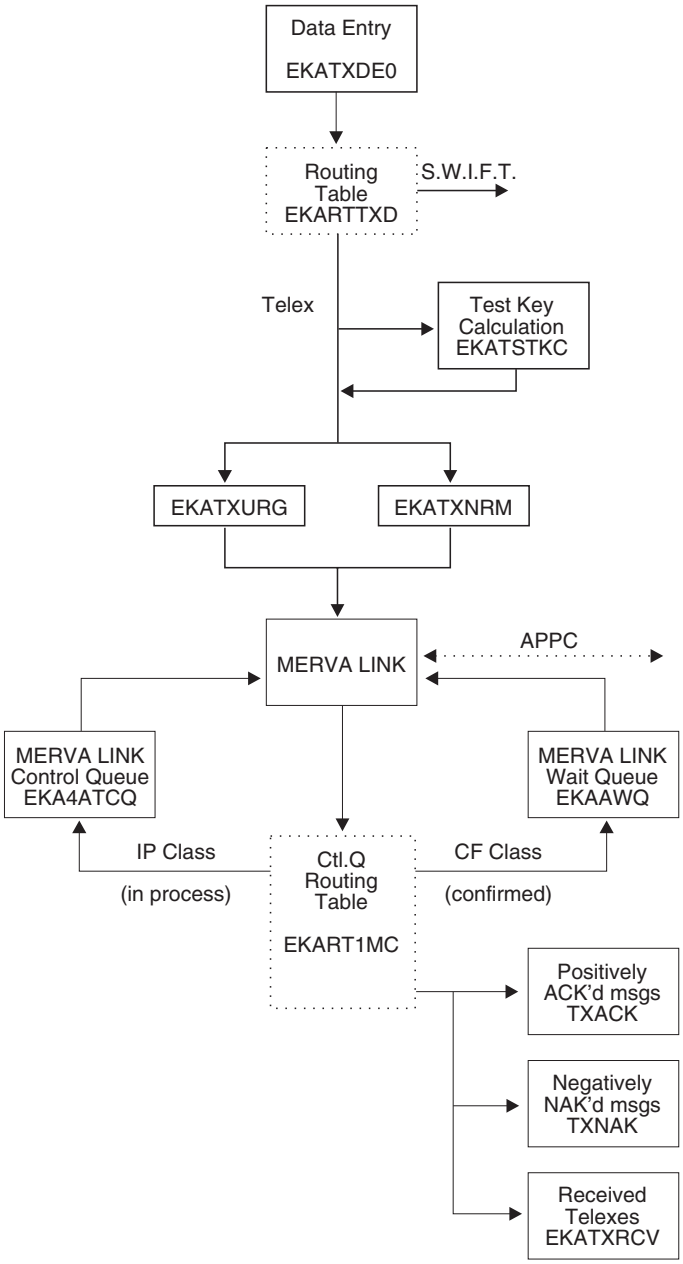

Figure 109. Routing of Telex Input and Output Messages in MERVA A

The following comments refer to Figure 109 and Figure 110 on page 231.

Data Flow and Notes:

[1] Messages are created in data entry queue EKATXDE0. These messages can be either unformatted telex messages or SWIFT messages and are handled differently.

[2] SWIFT messages are routed to queues TXAI0 or TXVE0 or both. This is not part of this example and is not described here.

[3] If YES is specified in the Test field of the telex header, the Telex messages are moved to the Test-Key calculation queue EKATSTKC, and routed to one of the MERVA Link send queues. If no Test-Key is required, a message is routed directly to a send queue.

[4] If the Type field in the header is 'U' (urgent) the message is routed to the send queue EKATXURG. It is otherwise routed to the queue EKATXNRM.

[5] Once in a MERVA Link send queue, the telex is transferred by MERVA Link using an APPC connection to MERVA B. MERVA Link also holds the message in its control queue EKA4ATCQ as an IP (in process) class message until a delivery report, indicating successful transfer to the remote MERVA Link system, is received from MERVA Link in MERVA B.

[6] The message class is changed to CF (confirmed) and the message is routed to the MERVA Link wait queue, EKAAWQ, to await the Telex Link acknowledgment from MERVA B.

[7] A message with class CA is treated like a received message (class LR) with acknowledgment information available (receipt return code field starts with '0'). The message is routed to the TXACK queue or, if the telex received a negative acknowledgment in MERVA B, to the TXNAK queue.

[8] In MERVA B Telex Link controls the sending of the message over the telex network. When the transmission acknowledgment is received, it is passed to MERVA Link in MERVA B and a MERVA Link status report is built. The status report is the MERVA Link mechanism for transferring acknowledgment data.

The Telex Link acknowledgment information is put into the status report by the user exit EKAMU133, and the status report is sent to MERVA Link in MERVA A.

[9] MERVA Link in MERVA A correlates the status report with the appropriate message in queue EKAAWQ.

User exit EKAMU133 is invoked to extract the acknowledgment data from the status report and add it to the message selected from EKAAWQ.

[10] After correlation, MERVA Link changes the class of the acknowledged message from CF to LR (last received), and the message is routed to the TXACK queue or, if the telex received a negative acknowledgment in MERVA B, to the TXNAK queue.

[11] Each message with class LR is also routed to the MERVA Link control queue EKA4ATCQ.

[12] MERVA Link in MERVA A informs MERVA Link in MERVA B that the status report has been delivered successfully.

This routing is carried out by the control queue routing table EKART1MC and depends on the MERVA Link receipt return code, field EKARECRC, set by the user exit EKAMU133. The code in EKART1MC that processes IP, CF, CA, and LR message classes is shown below.

```
EKART1MC DSLROUTE TYPE=DEFINE,FIELD=(CLASS,EKACLASS,,,,VFIRST)
         DSLROUTE TYPE=DEFINE,FIELD=(RECRC,EKARECRC,,,,VFIRST)
         DSLROUTE TYPE=DEFINE,FIELD=(DELRC,EKADELRC,,,,VFIRST)
         DSLROUTE TYPE=DEFINE,FIELD=(ACQNM,EKAACQNM,,,,VFIRST)
*-------------------------------------------------------------------*
* ROUTE ALL IP (IN PROCESS) MESSAGES AND THE LC CONTROL MESSAGE TO   *
* THE APPLICATION CONTROL QUEUE.                                     *
* THIS PART MAY CORRECT A HANDLING ERROR OF THE SYSTEM ADMINISTRATOR. *
*-------------------------------------------------------------------*
         DSLROUTE TYPE=TEST,COND=(CLASS,'IP',EQ),FALSE=TSTLC
         DSLROUTE TYPE=SET,TARGET=ACQNM,GOTO=END
TSTLC    DSLROUTE TYPE=TEST,COND=(CLASS,'LC',EQ),FALSE=TSTCF
         DSLROUTE TYPE=SET,TARGET=ACQNM,GOTO=END

⋮

TSTCF    DSLROUTE TYPE=TEST,COND=(CLASS,'CF',EQ),FALSE=TSTCA              [6]
         DSLROUTE TYPE=SET,TARGET='EKAAWQ',GOTO=END

⋮

*-------------------------------------------------------------------*
* TEST FOR AND ROUTE CONFIRMED AND ACKNOWLEDGED MESSAGES             *
*-------------------------------------------------------------------*
TSTCA    DSLROUTE TYPE=TEST,COND=(CLASS,'CA',EQ),TRUE=SETLRF,       *    [7]
               FALSE=TSTLR

⋮

TSTLR    DSLROUTE TYPE=TEST,COND=(CLASS,'LR',EQ),FALSE=TSTRC             [10]

⋮

         DSLROUTE TYPE=TEST,COND=(RECRC,'0',EQ,SHORT),TRUE=SETLRF        [10]
         DSLROUTE TYPE=TEST,COND=(DELRC,'00',EQ),TRUE=SETLRI

⋮

SETLRI   DSLROUTE TYPE=SET,TARGET='EKATXRCV'
         DSLROUTE TYPE=SET,TARGET=ACQNM,GOTO=END
SETLRA   DSLROUTE TYPE=SET,TARGET='EKAAWQ'
         DSLROUTE TYPE=SET,TARGET=ACQNM,GOTO=END

⋮

SETLRF   DSLROUTE TYPE=DEFINE,FIELD=(TXSTAMP,ENLSTAMP)
*        STAMPS FOR OUTGOING TELEX MESSAGES
*        'NAKENLXM' INDICATES NEG LOGICAL ACK RECEIVED
*        FROM THE TXIP OR FORMATTING ERROR
*        'ACKXMIT' INDICATES POSITIVE TRANSMISSIONS ACKS RECEIVED
*        'NAKXMIT' INDICATES NEGATIVE TRANSMISSIONS ACKS RECEIVED
         DSLROUTE TYPE=TEST,COND=(TXSTAMP,'ACK',EQ,SHORT),TRUE=ACK
NAK      DSLROUTE TYPE=SET,TARGET='TXNAK',GOTO=CHKCA                     [10]
ACK      DSLROUTE TYPE=SET,TARGET='TXACK',GOTO=CHKCA                     [10]
*
CHKCA    DSLROUTE TYPE=TEST,COND=(CLASS,'CA',EQ),TRUE=END
         DSLROUTE TYPE=SET,TARGET=ACQNM,GOTO=END                         [11]

⋮
```

Figure 110. Example of the Routing Table EKART1MC

Figure 110 also shows the routing of telex messages received by Telex Link in MERVA B and passed by MERVA Link to MERVA A. These telex messages have a

class of LR but no EKARECRC field, as no status reports are created. It does however have a delivery return code field EKADELRC, as it has been successfully delivered from MERVA B. The message is then routed to the MERVA A receive queue, EKATXRCV and to the MERVA Link control queue EKA4ATCQ.

Routing of Telex Messages in MERVA B

Figure 111 provides an overview of the flow of telex messages in MERVA B.

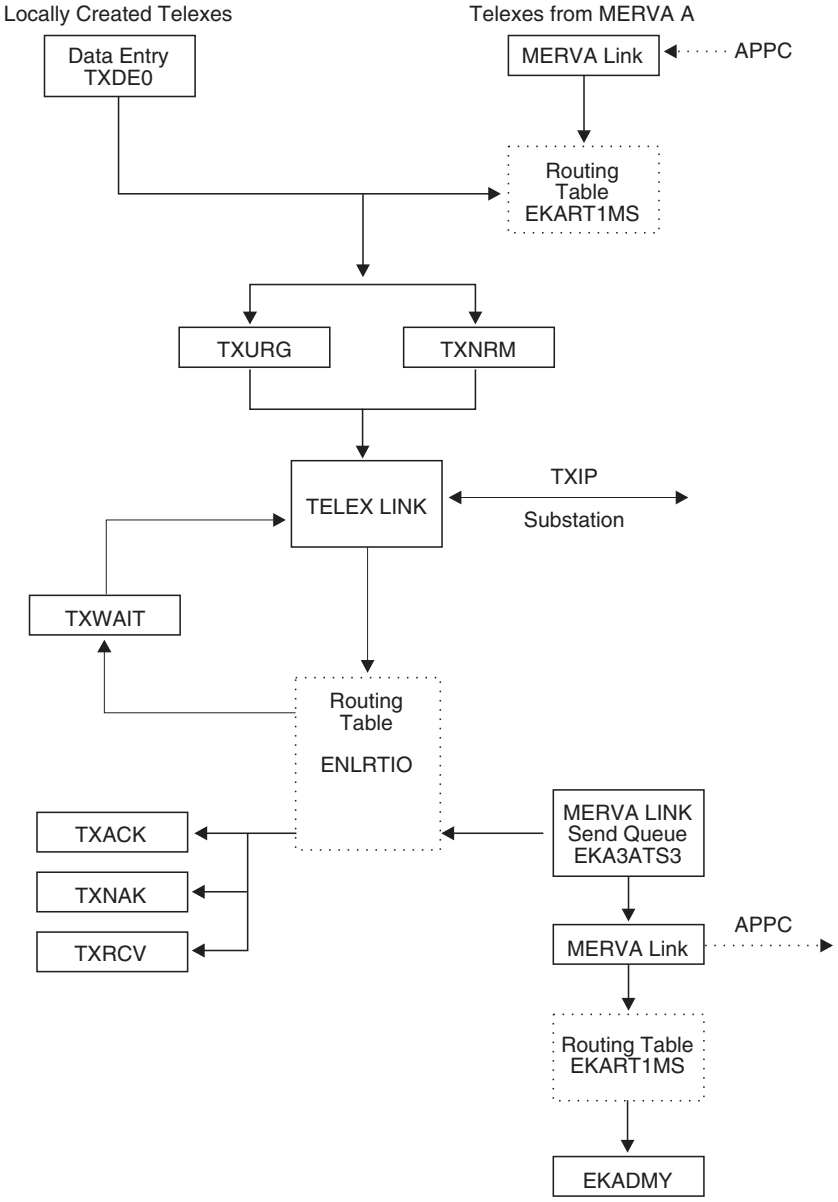

Figure 111. Routing of Telex Input and Output Messages in MERVA B

Messages received by MERVA Link in MERVA B are assigned the message class LR and routed by the EKART1MS routing table to one of the local Telex Link ready queues, TXNRM or TXURG. The code used for the routing table EKART1MS is shown below.

```
EKART1MS DSLROUTE TYPE=DEFINE,FIELD=(CLASS,EKACLASS,,,,,VFIRST)
         DSLROUTE TYPE=DEFINE,FIELD=(ACQNM,EKAACQNM,,,,,VFIRST)

⋮

*-------------------------------------------------------------------*
* TEST FOR AND ROUTE LR (INBOUND) MESSAGES.                         *
* ROUTE TELEX INPUT MESSAGES TO CORRESPONDING TELEX READY QUEUES.   *
*-------------------------------------------------------------------*
TSTLR    DSLROUTE TYPE=TEST,COND=(CLASS,'LR',EQ),FALSE=TSTRC
         DSLROUTE TYPE=DEFINE,FIELD=(TXPRI,ENLTXPRI,,,,,VFIRST)
         DSLROUTE TYPE=TEST,COND=(TXPRI,'U',EQ),TRUE=SETURG
         DSLROUTE TYPE=SET,TARGET='TXNRM'
         DSLROUTE TYPE=SET,TARGET=ACQNM,GOTO=END
SETURG   DSLROUTE TYPE=SET,TARGET='TXURG'
         DSLROUTE TYPE=SET,TARGET=ACQNM,GOTO=END

⋮
```

Figure 112. Example of Routing Table EKART1MS

Message processing is taken over by Telex Link and the message is submitted to the Telex substation for transmission over the telex network and routed from the ready queue to the TXWAIT queue to await acknowledgment. The routing table ENLRTIO controls the further routing specified in the Telex Link parameter module ENLPRMS. After a positive logical acknowledgment, the message is routed again to the TXWAIT queue to await transmission acknowledgment.

Any other acknowledgment is treated as a final acknowledgment and the message must be routed to an ACK or NAK queue in MERVA B or MERVA A. The message origin is indicated by MERVA Link control field EKAONODE (Origin Node). If this field is not found, the message was created locally in MERVA B and the message is routed to the Telex Link queue TXACK or TXNAK. If the field is present the message has been handled by MERVA Link, and must come from MERVA A, so the acknowledgment data must be returned to MERVA A. Figure 113 on page 234 shows the ENLRTIO routing table.

```
*  DEFINE TELEX LINK STAMP FIELD FROM THE TOF FOR THE ROUTING DECISION
ENLRTIO  DSLROUTE TYPE=DEFINE,FIELD=(TXSTAMP,ENLSTAMP),               +
               LENGTH=8,NOTFND=ERROR,EMPTY=ERROR
*
*        STAMPS FOR OUTGOING TELEX MESSAGES
*
*        'ACKENLXM' INDICATES POSITIVE LOGICAL ACKNOWLEDGMENT RECEIVED
*        FROM THE TXIP
         DSLROUTE TYPE=TEST,COND=(TXSTAMP,'ACKENLXM',EQ),TRUE=WAIT
*
*        'NAKENLXM' INDICATES NEGATIVE LOGICAL ACKNOWLEDGMENT RECEIVED
*        FROM THE TXIP OR FORMMATTING ERROR
         DSLROUTE TYPE=TEST,COND=(TXSTAMP,'NAKENLXM',EQ),TRUE=NAK
*
*        'ACKXMIT ' INDICATES POSITIVE TRANSMISSION ACKNOWLEDGMENT
*        RECEIVED
         DSLROUTE TYPE=TEST,COND=(TXSTAMP,'ACKXMIT ',EQ),TRUE=ACK
*
*        'NAKXMIT ' INDICATES NEGATIVE TRANSMISSION ACKNOWLEDGMENT
*        RECEIVED
         DSLROUTE TYPE=TEST,COND=(TXSTAMP,'NAKXMIT ',EQ),TRUE=NAK

⋮

WAIT     DSLROUTE TYPE=SET,TARGET=('TXWAIT'),GOTO=END
*
*        MESSAGES WITH POSITIVE TRANSMISSION ACKNOWLEDGMENT ARE ROUTED
*        TO TXACK AND PRINTED
ACK      DSLROUTE TYPE=DEFINE,FIELD=(EKAOR,EKAONODE),                 *
               NOTFND=ACK1,EMPTY=ACK1,LENGTH=8
*        ROUTE MSG ORIGINATING FROM MERVA -CREATE-
         DSLROUTE TYPE=SET,TARGET=('EKA3ATS3'),GOTO=END
*        ROUTE MSG ORIGINATING FROM MERVA -SEND-
ACK1     DSLROUTE TYPE=SET,TARGET=('TXACK'),GOTO=END
*
*        MESSAGES WITH NEGATIVE LOGICAL OR TRANSMISSION ACKNOWLEDGMENT
*        ARE ROUTED TO TXNAK
NAK      DSLROUTE TYPE=DEFINE,FIELD=(EKAOR,EKAONODE),                 *
               NOTFND=NAK1,EMPTY=NAK1
         DSLROUTE TYPE=SET,TARGET=('EKA3ATS3'),GOTO=END
NAK1     DSLROUTE TYPE=SET,TARGET=('TXNAK'),GOTO=END

⋮
```

Figure 113. Example of the Routing Table ENLRTIO

MERVA Link provides a status report message for the transfer of application status information to the partner system. The acknowledgment data is passed back to MERVA A as follows:

1. The user exit EKAMU133 adds the following data to the status report:
   - Type of acknowledgment (field MSGACK)
   - Telex Link transmission acknowledgment (field ENLXMACK)
   - Telex Link transmission header (field ENLXMHD)
   - Telex Link transmission status (field ENLSTAMP)
   - Telex Link transmission data (field ENLXMNDX)

   The MERVA Link receipt return code (field EKARECRC) and receipt diagnostic code are set.
2. The status report is copied with message class IP to MERVA Link control queue EKA3ATCQ and transferred to MERVA Link in MERVA A.
3. Following receipt of the delivery report from MERVA Link in MERVA A, MERVA Link in MERVA B changes the message class from IP to CF and routes

the status report using the control queue's routing table EKART1MS to the dummy queue EKADMY. Routing to this queue has the same effect as a **delete** command.

4. Processing of this message in MERVA B is complete.

Figure 114 shows the routing instructions.

```
EKART1MS DSLROUTE TYPE=DEFINE,FIELD=(CLASS,EKACLASS,,,,,VFIRST)
         DSLROUTE TYPE=DEFINE,FIELD=(ACQNM,EKAACQNM,,,,,VFIRST)

⋮

*--------------------------------------------------------------------*
* TEST FOR AND ROUTE CONFIRMED MESSAGES. DELETE MERVA LINK STATUS    *
* REPORT AND TELEX OUTPUT MESSAGES.                                  *
*--------------------------------------------------------------------*
TSTCF    DSLROUTE TYPE=TEST,COND=(CLASS,'CF',EQ),FALSE=TSTLR
         DSLROUTE TYPE=SET,TARGET='EKADMY',GOTO=END

⋮
```

Figure 114. Handling of Confirmed Messages in the Routing Table EKART1MS

The routing table ENLRTIO handles the routing of all messages from the Telex substation. All incoming telex messages are routed to the local Telex Link receive queue TXRCV for visual inspection. Any messages for MERVA A are transferred by routing the message to the MERVA Link send queue, EKA3ATS3. MERVA Link transfers the telex message to MERVA A. On receipt of the delivery indication from MERVA Link in MERVA A, which indicates that the message has been successfully transferred to MERVA A, the message class changes from IP to CF and is routed to the EKADMY dummy queue. Routing to this queue has the same effect as a **delete** command (see Figure 114). Figure 115 shows the ENLRTIO routing table.

```
*  DEFINE TELEX LINK STAMP FIELD FROM THE TOF FOR THE ROUTING DECISION
ENLRTIO  DSLROUTE TYPE=DEFINE,FIELD=(TXSTAMP,ENLSTAMP),               +
               LENGTH=8,NOTFND=ERROR,EMPTY=ERROR

⋮

*        'TELEXRCV' INDICATES A RECEIVED TELEX MESSAGE
         DSLROUTE TYPE=TEST,COND=(TXSTAMP,'TELEXRCV',EQ),TRUE=RCV

⋮

*        RECEIVED TELEX MESSAGES ARE ROUTED TO TXRCV AND TXSTPLR
RCV      DSLROUTE TYPE=SET,TARGET=('TXRCV')
         DSLROUTE TYPE=SET,TARGET=('TXSTPLR'),GOTO=END

⋮
```

Figure 115. Example of Routing Table ENLRTIO

Customizing MERVA A and MERVA B

In the sample library, in the source library, and in the macro library you can find the tables and programs required to set up the connection of two MERVA ESA systems using MERVA Link.

In MVS, the MERVA ESA sample library is a partitioned data set with the low level qualifier SDSLSAM0.

The MERVA ESA source library is a partitioned data set with the low level qualifier SDSLSRC0.

The MERVA ESA macro library is a partitioned data set with the low level qualifier SDSLMAC0.

In VSE, the MERVA ESA sample programs are part of the source library.

The following two tables show the source files and copy books that must be installed in both MERVA A and MERVA B.

If not stated otherwise for the MVS environment, the source files and copy books can be found in the sample library.

MERVA A (Message Creating)			
File	**Copy Book**	**Description**	**Network**
DSLFNTT	EKAFNTMC	Function table entries for SWIFT and Telex. For MVS: copy book EKAFNTMC is contained in the macro library, file DSLFNTT is contained in the source library.	n.a.
EKAPTC		MERVA Link partner table	n.a.
EKAR2AI0		Routing Table: after authorization, routing messages to MERVA Link send queues	SWIFT
DWSR1AO0		Routing Table: routing after authentication output	SWIFT
DWSR1DO0		Routing Table: routing banking messages without errors after distribution output	SWIFT
EKARS2MC		Routing Table: routing of acknowledged messages and SWIFT output messages	SWIFT
EKAMU133		MFS User Exit required by MERVA Link. For MVS: this file is contained in the source library.	SWIFT Telex
EKART1MC		Routing Table: routing of acknowledged messages and received messages	Telex
EKARTTXD		Routing Table: routing after data entry	Telex
EKARTTXK		Routing Table: routing after test key calculation	Telex
	EKATCTC	CICS TCT entry for VTAM node required by MERVA Link	n.a.

MERVA B (Message Sending to External Network)			
File	**Copy Book**	**Description**	**Network**
DSLFNTT	EKAFNTMS	Function table entries for MERVA Link send queues and control queues. For MVS: copy book EKAFNTMS is contained in the macro library, file DSLFNTT is contained in the source library.	n.a.
EKAPTS		MERVA Link partner table	n.a.
EKAR1IN		Routing Table: SWIFT Link routing of input messages	SWIFT
EKAR1OUT		Routing Table: SWIFT Link routing of output messages	SWIFT
EKARS2MS		Routing Table: routing of SWIFT input messages to SWIFT Link Ready queues	SWIFT
DWSLTTS		SWIFT Link logical terminal table	SWIFT
EKAMU133		MFS User Exit required by MERVA Link. For MVS: this file is contained in the source library.	SWIFT Telex
EKART1MS		Routing Table: routing to send queues	Telex
ENLRTIO		Routing Table: routing of sent and received messages	Telex
ENLPRMS		Telex Link generation parameter module	Telex
	EKATCTS	CICS TCT entries for VTAM node required by MERVA Link and for the Telex Link	n.a.

Installing MERVA A and MERVA B

The tables and programs in MERVA A and MERVA B are installed as follows:

1. Network independent tables:
   - Function Table

     MERVA A: Assemble DSLFNTT containing copy book EKAFNTMC, and link-edit as DSLFNTT.

     MERVA B: Assemble DSLFNTT containing copy book EKAFNTMS, and link-edit as DSLFNTT.
   - MERVA ESA Customization Parameter Module

     MERVA A and MERVA B:
     Modify the parameter module DSLPRM as required. The values for the following parameters must be different in each MERVA ESA system:
     - CVTEXTO
     - DSLID
     - NAME

     Assemble and link-edit as DSLPRM.
   - MERVA Link partner table

     MERVA A: Assemble EKAPTC, and link-edit as EKAPT.

     MERVA B: Assemble EKAPTS, and link-edit as EKAPT.
2. Tables for SWIFT network:
   - Routing Tables

MERVA A: Assemble EKAR2AI0, and link-edit as EKAR2AI0.

Assemble DWSR1AO0, and link-edit as DWSR1AO0.

Assemble DWSR1DO0, and link-edit as DWSR1DO0.

Assemble EKARS2MC, and link-edit as EKARS2MC.

MERVA B: Assemble EKAR1IN, and link-edit as EKAR1IN.

Assemble EKAR1OUT, and link-edit as EKAR1OUT.

Assemble EKARS2MS, and link-edit as EKARS2MS.

- Logical Terminal Table

  MERVA B: Modify DWSLTTS as required. Assemble and link-edit as DWSLTT.

3. MFS user exit for SWIFT and telex network:
   - User Exit

     MERVA A and MERVA B:
     EKAMU133 must already be available as an executable program after MERVA ESA has been installed.
4. Tables for the telex network:
   - Routing Tables

     MERVA A: Assemble EKART1MC, and link-edit as EKART1MC.

     Assemble EKARTTXD, and link-edit as EKARTTXD.

     Assemble EKARTTXK, and link-edit as EKARTTXK.

     MERVA B: Assemble EKART1MS, and link-edit as EKART1MS.

     Assemble ENLRTIO, and link-edit as ENLRTIO.
   - Telex Link Generation Parameter Module

     MERVA B: Assemble ENLPRMS, and link-edit as ENLPRM.
5. CICS definitions:
   - Terminal Definition

     Note: The following description relates to parameters of the TCT (Terminal Control Table). If you define terminals online in your CICS installation, use the appropriate parameters of the terminal definition statements.

     MERVA A: In copy book EKATCTC, modify the following parameters:
     - NETNAME in macro DFHTCT TYPE=SYSTEM, specify the VTAM network name of the **remote CICS**.
     - MODENAM in macro DFHTCT TYPE=MODESET, specify the name used in the LOGMODE parameter of the appropriate VTAM LOGON Mode table entry.

     Include copy book EKATCTC in your TCT definitions and create a new TCT.

     MERVA B: In copy book EKATCTS, modify the following parameters:
     - NETNAME in macro DFHTCT TYPE=SYSTEM, specify the VTAM network name of the **remote CICS**.

– MODENAM in macro DFHTCT TYPE=MODESET, specify the name used in the LOGMODE parameter of the appropriate VTAM LOGON Mode table entry.
– TRMIDNT and NETNAME in macro DFHTCT TYPE=TERMINAL

Include copy book EKATCTS in your TCT definitions and create a new TCT.

- SIT - System Initialization Table

  MERVA A and MERVA B:
  Specify parameter ISC=YES.

  Create a new SIT or specify ISC=YES in your CICS startup job.

MERVA Link Partner Table Relationships

Many of the parameters in a MERVA Link partner table relate to definitions that must be made in MERVA ESA tables, in CICS definitions, or both. These relationships and possible references to specific partner table parameters are described based on a partner table entry in the file EKAPTC as shown in Figure 116.

```
       TITLE 'MERVA LINK SAMPLE PARTNER TABLE FOR NODE C3'
*-------------------------------------------------------------------
*  THIS PARTNER TABLE IS USED IN THE REMOTE SWIFT/TELEX LINK SCENARIO
*  FOR A SAMPLE MERVA LINK CICS APPC CONNECTION.
*-------------------------------------------------------------------
         SPACE
         EKAPT TYPE=INITIAL,                  MESSAGE TRANSFER NODE C3-
               NODE=SDFC3                                              [1]
         EJECT
*-------------------------------------------------------------------
* A4AS2 APPC CONNECTION FROM MERVA -CREATE- FOR SWIFT
*-------------------------------------------------------------------
         EKAPT TYPE=ASP,                          APPL SUPPORT PROCESS-
               SENDQC=(EKAL1GPA,EKAL1RFU,EKAL1RFN),  SEND QUEUE CLUSTER-  [2]
               DEST=(SDFC4,A3AS2),                DESTINATION ADDRESS-
               CONTROL=EKA4A2CQ,                   CONTROL QUEUE NAME-  [3]
               IRROUTE=(ACK,EKAAWQ,CTLQ),        REC REPORT CORRELATION-  [4]
               NAME=(A4AS2,'APPC TO SWIFT FRONT END'),               -
               MFSEXIT=7133,                                         -  [5]
               MTP=T4AS2,                                            -
               FORMAT=(QUEUE)
         SPACE
*-------------------------------------------------------------------
         EKAPT TYPE=MTP,                          MSG TRANSFER PROCESS-
               LINK=(APPC,CA04),                   PARTNER SYSTEM INFO-  [6]
               PARTNER=(X43AS2,EKAR),           PARTNER MTP INFORMATION-  [7]
               NAME=(T4AS2,X34AS2),                                  -
               ASP=A4AS2

⋮
```

Figure 116. MERVA Link Partner Table Entry in the File EKAPTC

Notes:

[1] NODE=SDFC3

This is the local node name for MERVA Link in MERVA A. In MERVA B this information can be used in routing tables to determine the origin of a message by testing the MERVA Link control field EKAONODE. An example is contained in the routing table EKAR1IN.

To change the value for this parameter in the partner table, you must also change it in the routing table concerned (and in the partner table of MERVA Link in MERVA B, file EKAPTS).

[2] SENDQC=(EKAL1GPA,EKAL1RFU,EKAL1RFN)

MERVA Link send queues must be defined in the MERVA ESA function table. The following sample entries are provided in the copy book EKAFNTMC:

```
*        SOURCE QUEUE FOR GENERAL PURPOSE APPLICATION
         DSLFNT NAME=EKAL1GPA,QUEUE=YES,THRESH=30,NEXT=EKAL1ERR,       *
               TRAN=EKAS,RELATED=(EKAL1RFU,EKAL1RFN),                  *
               DESCR='Ready Queue of GPA for Sending to SWIFT'
*
*        SOURCE QUEUE FOR URGENT FINANCIAL APPLICATION
         DSLFNT NAME=EKAL1RFU,QUEUE=YES,THRESH=30,NEXT=EKAL1ERR,       *
               TRAN=EKAS,RELATED=(EKAL1GPA,EKAL1RFN),                  *
               DESCR='Urgent Ready Queue of FIN for Sending to SWIFT'
*
*        SOURCE QUEUE FOR NORMAL FINANCIAL APPLICATION
         DSLFNT NAME=EKAL1RFN,QUEUE=YES,THRESH=30,NEXT=EKAL1ERR,       *
               TRAN=EKAS,RELATED=(EKAL1GPA,EKAL1RFU),                  *
               DESCR='Normal Ready Queue of FIN for Sending to SWIFT'
```

Each of these function table entries contain the parameter `TRAN=EKAS`. This is the default transaction identifier of the sending application support process. It is generated for the partner table entry, if parameter TRAN was not specified there, as in the example.

A CICS transaction definition for this transaction identifier is also required.

Note: This transaction definition must be already active after the installation of MERVA ESA.

To change the names of MERVA Link send queues, modify them in the:

- Partner table
- Function table

To change the transaction identifier associated with a send queue, modify the:

- Partner table
- Function table
- CICS transaction definition

[3] CONTROL=EKA4A2CQ

A MERVA Link control queue must also be defined in the MERVA ESA function table. The copy book EKAFNTMC, shown in Figure 117 contains the following entry:

```
*        CONTROL QUEUE
         DSLFNT NAME=EKA4A2CQ,KEY1=(EKACLASS,2,1),ROUTE=EKARS2MC,      *
               QUEUE=YES,THRESH=50,SPCMND=(ROU,DEL),                  *
               DESCR='Application Control Queue of ASP A4AS2'
```

Figure 117. MERVA Link Control Queue in the MERVA ESA Function Table

Routing table EKARS2MC is associated with the control queue and requires a CICS program definition.

Note: This program definition must be already active after the installation of MERVA ESA.

To change the name of a MERVA Link control queue, modify it in the:

- Partner table
- Function table

To change the name of a routing table associated with a control queue, modify the:

- Function table
- CICS program definition

[4] IRROUTE=(...,EKAAWQ,...)

A MERVA Link acknowledgment wait queue must also be defined in the MERVA ESA function table. The copy book EKAFNTTC, which is not part of this sample but should be available after MERVA ESA installation, contains the entry shown in Figure 118.

```
*        DO NOT SPECIFY NEXT= FOR AN ACK WAIT QUEUE.
         DSLFNT NAME=EKAAWQ,KEY1=(EKAAMSID,16,1),ROUTE=EKARTS,         *
               QUEUE=YES,THRESH=50,SPCMND=(ROU,DEL),                  *
               DESCR='MERVA Link Sample Ack Wait Queue'
```

Figure 118. MERVA Link Acknowledgment Wait Queue in the MERVA ESA Function Table

To change the name of a MERVA Link acknowledgment wait queue, modify it in the:

- Partner table
- Function table

[5] MFSEXIT=7133

7133 is the number of MFS user exit EKAMU133. The user exit must be defined in the MERVA ESA MFS program table.

A CICS program definition for the user exit is also required.

Note: The entry in the MFS program table and the program definition must be already active after the installation of MERVA ESA.

To change the number of the user exit, modify it in the:

- Partner table
- MFS program table

To change the name of the user exit, modify it in the:

- MFS program table

- CICS program definition

In both cases, link-edit MERVA ESA module DSLMMFS with the modified MFS program table.

[6] LINK=(APPC,CA04)

This parameter defines an APPC connection to the partner system MERVA Link in MERVA B.

A modification in the CICS terminal definition is also necessary. In this sample, CICS TCT entries are used. The copy book EKATCTC contains the following entries:

```
BISCS    DFHTCT TYPE=SYSTEM,                                          *
               ACCMETH=VTAM,                                          *
               NETNAME=I4WAC381,                                      *
               TRMTYPE=LUTYPE62,                                      *
               SYSIDNT=CA04,                                          *
               CONNECT=AUTO,                                          *
               RUSIZE=2048,                                           *
               BUFFER=2048
         DFHTCT TYPE=MODESET,                                         *
               SYSIDNT=CA04,                                          *
               MAXSESS=(4,2),                                         *
               CONNECT=AUTO,                                          *
               MODENAM=APPCLU62       MUST MATCH VTAM LOGMODE TABLE
```

An entry is also required in the VTAM LOGON mode table, for example (this example is not contained in a MERVA ESA source library):

```
APPCLU62  MODEENT LOGMODE=APPCLU62,FMPROF=X'12',TSPROF=X'04',          *
                PRIPROT=X'B1',SECPROT=X'B1',COMPROT=X'70A0',           *
                PSERVIC=X'060038000000380000000000',RUSIZES=X'8686',   *
                PSNDPAC=X'03',SRCVPAC=X'04',SSNDPAC=X'05',TYPE=0
```

To change the name of the intercommunication link CA04, modify it in the:

- Partner table
- CICS terminal definition (TCT)

Note: The name specified in the SYSIDNT parameter must be the name used in both the DFHTCT TYPE=SYSTEM and the DFHTCT TYPE=MODESET macros.

To change the VTAM LOGMODE name APPCLU62, modify it in the:

- CICS terminal definition (TCT)
- VTAM LOGON mode table

[7] PARTNER=(...,EKAR)

EKAR is the transaction identifier for the remote receiving message transfer process. In the transaction definition in the remote CICS, an entry is required for this transaction identifier.

Note: This transaction definition must be already active after the installation of MERVA ESA.

To change the transaction identifier, modify the:

- Transaction definition in the remote CICS
- Partner table

Chapter 7. The MERVA Link for Unix System Services (USS)

MERVA Link USS is a set of MERVA Link functions that execute in the OS/390 UNIX System Services (USS) environment, and that route MERVA Link conversations from SNA APPC to TCP/IP, and vice versa. For example, using the MERVA Link USS Gateway, which handles the protocol conversion, a MERVA USE & Branch workstation can be connected to MERVA ESA V4.1 via a TCP/IP-based MERVA Link protocol.

This chapter describes how to adapt the MERVA Link functions executing in the OS/390 USS environment to meet the requirements of a MERVA Link USS Gateway. The customization of the MERVA Link functions executing in the MERVA ESA CICS and MERVA ESA IMS environments are described in "Chapter 6. The MERVA Link for CICS and IMS" on page 161.

Defining Application Control Table Entries (Samples)

Use the MERVA Link application control table (ACT) to customize MERVA Link in the OS/390 UNIX System Services (USS) environment. The ACT contains the definitions of all MERVA Link processes that support the exchange of messages between partner systems. It consists of the ACT header and two sets of ASP and ISC entries:

- The ACT header contains definitions that apply to the MERVA Link node.
- An ASP entry contains definitions that apply to a particular MERVA Link message transfer application. The ASPs for MERVA Link USS are for installation verification and test purposes only, which is why their customization is not described in this chapter.
- An ISC entry contains definitions that apply to the intersystem connection to a partner MERVA Link system. ISC information is used to establish a conversation between a local Message Transfer Process (MTP) or a local Routing Process (RP) and its partner process in a (remote) partner system. Message routing is the main function of MERVA Link USS in MERVA ESA V4.1.

The definitions that must go into a MERVA Link USS ACT are provided by a MERVA Link administrator in a MERVA Link USS Configuration File (CFG). CFG statements are the means to describe the MERVA Link USS customization.

Sample 1 describes a MERVA Link USS Gateway that provides message routing services to all four MERVA Link CICS and IMS nodes described in "Chapter 6. The MERVA Link for CICS and IMS" on page 161. Sample 2 extends sample 1 by adding definitions for TCP/IP connections to MERVA Link AIX and MERVA Link NT partner systems.

The sample MERVA Link network consists of a number of MERVA Link nodes. These nodes are numbered and characterized as follows:

Node 1 (C1) The first MERVA CICS system (SNA APPC LU is CTS1LUNM)

Node 2 (C2) The second MERVA CICS system (SNA APPC LU is CTS2LUNM)

Node 3 (I1) The first MERVA IMS system (SNA APPC LU is IA01LUNM)

Node 4 (I2) The second MERVA IMS system (SNA APPC LU is IA02LUNM)

Node 5 (A1) MERVA AIX system (TCP/IP host name is AIX1HOST)

Node 7 (W1) MERVA NT system (TCP/IP host name is WNT1HOST)

Node 9 (U1) MERVA Link USS Gateway with the APPC/MVS System Base LU name MVS9LUNM and the TCP/IP host name USS1HOST

Sample 1: Gateway between MERVA Link CICS and IMS Systems

This CFG sample shows the definition of the MERVA Link USS node U1 and its connections to nodes C1, C2, I1, and I2. The sample configuration file consists of a main configuration file and a set of configuration include files. Alternatively, you can provide all resource definitions in the main configuration file instead of using configuration include files.

All sample configuration files are contained in the **cfg** subdirectory of the sample MERVA USS Instance Directory **/u/merva1**.

ACT Configuration Main File

The sample main configuration file is named **ekaacd.cfg**. It is shown below.

```
#*********************************************************************
# MERVA Link USS EKAACT Configuration
#*********************************************************************
ACTH:                          # ACT Header Parameter Group              [1]
      local_node             = U1                                        [2]
      trace_file_directory   = /u/merva1/trc                             [3]

#====================================================================
# MERVA Link USS EKAACT ISC Entries
#====================================================================
INCC: C1                       # include parameters for partner node C1  [4]
INCC: C2                       # include parameters for partner node C2  [5]
INCC: I1                       # include parameters for partner node I1  [6]
INCC: I2                       # include parameters for partner node I2  [7]
```

Figure 119. ACT Sample 1: Main CFG File for Local Node U1 (ekaacd.cfg)

Notes:

[1] The keyword **ACTH** starts an ACT header parameter group. The CFG statements up to the next parameter group keyword or an INClude statement are interpreted as ACT header parameter definition statements.

[2] The local node name is U1. It must be different from all partner MERVA Link node names.

[3] Processing trace files are written to the sample trace directory **/u/merva1/trc** if a trace is requested. A processing trace is not requested in this sample (tpi and tci trace levels default to 0). Trace levels can be modified interactively.

[4] The keyword **INCC** specifies the inclusion of a CFG include file. The name of the include file (**cfgc.C1**) consists of the include file group identifier **cfgc** (that corresponds to the keyword INCC) and the specified partner node name **C1**.

[5] Includes the CFG file for partner node C2 (**cfgc.C2**).

[6] Includes the CFG file for partner node I1 (**cfgc.I1**).

[7] Includes the CFG file for partner node I2 (**cfgc.I2**).

ACT Configuration Include File for Partner Node C1

The configuration include file for partner node C1 is named **cfgc.C1**. It is shown below.

```
#*********************************************************************
# MERVA Link USS EKAACT Configuration Include File for Partner Node C1
#*********************************************************************
ACTC:                          # ACT ISC Entry Parameter Group          [1]
      partner_node           = C1                                       [2]
      symbolic_destination   = EKARC1                                   [3]
```

Figure 120. ACT Sample 1: CFG Include File for Partner Node C1 (cfgc.C1)

Notes:

[1] The keyword **ACTC** starts an ACT ISC parameter group. The following CFG statements (up to the next parameter group keyword or an INClude statement) are interpreted as ACT ISC entry parameter definition statements.

[2] The name of the partner MERVA Link node defined in this ACT ISC entry is C1.

[3] The identifiers for the SNA APPC connection to partner node C1 are contained in APPC/MVS Side Information. The symbolic destination name defined in APPC/MVS for this set of connection parameters is **EKARC1**.

ACT Configuration Include File for Partner Node C2

The configuration include file for partner node C2 is named **cfgc.C2**. It is shown below.

```
#*********************************************************************
# MERVA Link USS EKAACT Configuration Include File for Partner Node C2
#*********************************************************************
ACTC:                          # ACT ISC Entry Parameter Group          [1]
      partner_node           = C2                                       [2]
      symbolic_destination   = EKARC2                                   [3]
```

Figure 121. ACT Sample 1: CFG Include File for Partner Node C2 (cfgc.C2)

Partner MERVA Link node C2 is defined in this ACT ISC entry. The symbolic destination defined in APPC/MVS for C2 is **EKARC2**.

ACT Configuration Include File for Partner Node I1

The configuration include file for partner node I1 is named **cfgc.I1**, and is shown below.

```
#*********************************************************************
# MERVA Link USS EKAACT Configuration Include File for Partner Node I1
#*********************************************************************
ACTC:                          # ACT ISC Entry Parameter Group          [1]
      partner_node           = I1                                       [2]
      symbolic_destination   = EKARI1                                   [3]
```

Figure 122. ACT Sample 1: CFG Include File for Partner Node I1 (cfgc.I1)

Partner MERVA Link node I1 is defined in this ACT ISC entry. The symbolic destination defined in APPC/MVS for I1 is **EKARI1**.

ACT Configuration Include File for Partner Node I2

The configuration include file for partner node I2 is named **cfgc.I2**, and is shown below.

```
#*********************************************************************
# MERVA Link USS EKAACT Configuration Include File for Partner Node I2
#*********************************************************************
ACTC:                          # ACT ISC Entry Parameter Group            [1]
      partner_node           = I2                                         [2]
      symbolic_destination   = EKARI2                                     [3]
```

Figure 123. ACT Sample 1: CFG Include File for Partner Node I2 (cfgc.I2)

Partner MERVA Link node I2 is defined in this ACT ISC entry. The symbolic destination defined in APPC/MVS for I2 is **EKARI2**.

Sample 2: Gateway between MERVA Link ESA and MERVA Workstations

CFG sample 2 extends CFG sample 1 by adding two MERVA workstations to the set of partner MERVA systems. The additional partner MERVA systems are:

- A MERVA AIX system (node name A1)
- A MERVA NT system (node name W1)

The two workstations are connected to the MERVA Link USS Gateway via TCP/IP only. An SNA APPC connection to the workstations is not supported in this sample.

The additional configuration files are also contained in the **cfg** subdirectory of the sample MERVA USS Instance Directory **/u/merva1**.

ACT Configuration Main File

The extended sample main configuration file **ekaacd.cfg** is shown below.

```
#*********************************************************************
# MERVA Link USS EKAACT Configuration
#*********************************************************************
ACTH:                          # ACT Header Parameter Group
      local_node             = U1
      trace_file_directory   = /u/merva1/trc

#====================================================================
# MERVA Link USS EKAACT ISC Entries
#====================================================================
INCC: C1                       # include parameters for partner node C1
INCC: C2                       # include parameters for partner node C2
INCC: I1                       # include parameters for partner node I1
INCC: I2                       # include parameters for partner node I2
INCC: A1                       # include parameters for partner node A1     [1]
INCC: W1                       # include parameters for partner node W1     [2]
```

Figure 124. ACT Sample 2: Main CFG File for Local Node U1 (ekaacd.cfg)

Notes:

[1] Includes the CFG file for partner node A1 (**cfgc.A1**).

[2] Includes the CFG file for partner node W1 (**cfgc.W1**).

ACT Configuration Include File for Partner Node A1

The configuration include file for partner node A1 is named **cfgc.A1**, and is shown below.

```
#*********************************************************************
# MERVA Link USS EKAACT Configuration Include File for Partner Node A1
#*********************************************************************
ACTC:                          # ACT ISC Entry Parameter Group            [1]
      partner_node           = A1                                         [2]
      partner_host_name      = aix1host.financial.institution.com         [3]
      msg_port_number        = 7110                                       [4]
      password_encryption    = basic                                      [5]
```

Figure 125. ACT Sample 2: CFG Include File for Partner Node A1 (cfgc.A1)

Notes:

[1] The keyword **ACTC** starts an ACT ISC parameter group. The following CFG statements (up to the next parameter group keyword or an INClude statement) are interpreted as ACT ISC entry parameter definition statements.

[2] The name of the partner MERVA Link node defined in this ACT ISC entry is A1.

[3] The TCP/IP partner host name of partner node A1 can be specified as a fully-qualified domain name, or as a short partner host name that is defined in the local system or in a domain name server.

[4] The sample TCP/IP port number for the MERVA Link messaging application is 7110 in all MERVA Link systems supporting TCP/IP. The TCP/IP port number that is actually assigned to the MERVA Link messaging service in partner host AIX1HOST must be specified here.

[5] The basic password encryption algorithm **must** be used for a conversation between OS/390 USS and AIX systems. Basic password encryption is used by default in MERVA Link USS. You can omit this statement.

ACT Configuration Include File for Partner Node W1

The configuration include file for partner node W1 is named **cfgc.W1**, and is shown below.

```
#*********************************************************************
# MERVA Link USS EKAACT Configuration Include File for Partner Node W1
#*********************************************************************
ACTC:                          # ACT ISC Entry Parameter Group            [1]
      partner_node           = W1                                         [2]
      partner_host_name      = wnt1host.securities.institution.com        [3]
      msg_port_number        = 7110                                       [4]
```

Figure 126. ACT Sample 2: CFG Include File for Partner Node W1 (cfgc.W1)

The partner MERVA Link node W1 is defined in this ACT ISC entry. The TCP/IP host name of the partner node W1 is specified by its fully-qualified domain name. The TCP/IP port number for the MERVA Link messaging application is 7110.

Customizing the MERVA Link USS ACT

The MERVA Link USS Configuration Facility is used to customize a MERVA Link USS instance. The non-confidential data of this facility is stored in a unique configuration file, or in a main configuration file and a set of configuration include files.

A MERVA Link configuration file is a flat-text file in an HFS directory that can be generated by any text editor, for example, the ISPF-like editor provided in the TSO/E USS environment. Configuration include files are retrieved by default from the directory that contains the main configuration file. Configuration include files can be included from other directories if the fully-qualified path name is specified in the INClude statement.

The structure of a configuration file corresponds to the structure of the MERVA Link ACT. This means, there is one group of ACT header parameters, one or more groups of ASP parameters, and one or more groups of ISC parameters. The ASP and ISC parameter groups can be dynamically included in a configuration file from configuration include files by coding include statements in the main configuration file.

Configuration File Syntax

A MERVA Link USS Configuration File consists of a sequence of the following lines:

- Empty lines and comment lines
- ACT parameter group identification lines
- ACT parameter lines
- Include lines

Empty Lines and Comment Lines

An empty line contains only blanks and the new-line character. An explicit comment line starts with a hash character (**#**) in the first column of a line. An implicit comment line is any line that is not recognized as one of the other line types.

ACT Parameter Group Identification Lines

An ACT parameter group identification line defines the beginning and the type of an ACT parameter group. It starts with an ACT Parameter Group Identifier:

ACTH: An ACT Header parameter group

ACTA: An ACT ASP parameter group

ACTC: An ACT ISC parameter group

All information following an ACT Parameter Group Identifier is ignored.

ACT parameter lines

An ACT parameter line consists of three tokens, which can be followed by comment information. The three tokens are a parameter identifier, an equals sign (**=**), and a parameter value.

A parameter identifier is a character string of up to 22 characters that defines the meaning of the parameter value. The set of valid parameter identifiers is defined by the applicable ACT Parameter Group. Parameter lines with a parameter identifier that is invalid for the applicable parameter group are considered to be comment lines.

A parameter identifier must be followed by at least one blank and an equals sign (=). One or more blanks precede the parameter value. A parameter value can be one of the following:

- Resource name of up to 8 characters, for example, an ASP name
- Trace file directory name of up to 32 characters
- TCP/IP partner host name of up to 64 characters
- A number of up to 5 digits, for example, TCP/IP port numbers and trace levels
- A yes/no indicator (represented by y, Y, or 1 for yes, and n, N, or 0 for no)
- Keyword, for example, password encryption method 'crypt' or 'basic'

Data after a parameter value is considered to be a comment except in the ASP Free Form Name parameter. In this case, the data in the parameter line after the equals sign is interpreted as the parameter value. The length of the ASP Free Form Name is limited to 60 characters.

Include Lines

An include line contains an include statement that reads configuration data from a configuration include file. An include statement consists of an include instruction followed by an include file identifier. Data following the include file identifier is ignored.

Include instructions are **INCF:**, **INCA:**, and **INCC:**. The different include instructions define the meaning of the include file identifier as follows:

INCF: file_name
Either the fully-qualified path of the configuration include file, starting with a slash (/), or the relative path based on the directory of the main configuration file, without a slash.

INCA: asp_name
The file **cfga.asp_name** must be included from the directory of the main include file. A configuration include file starting with **cfga.** is supposed to contain one ACT ASP parameter group.

INCC: partner_node_name
The file **cfgc.partner_node_name** must be included from the directory of the main include file. A configuration include file starting with **cfgc.** is supposed to contain one ACT ISC parameter group.

Include files with the prefixes **cfga.** and **cfgc.** can contain any number and types of parameter groups. However, you can follow the convention associated with these prefixes to avoid confusion.

ACT Header Parameters

The ACT Header parameters and their valid parameter values are:

local_node
The name of the local MERVA Link node. This parameter is mandatory. The local node name identifies this node in a network of interconnected MERVA Link systems.

trace_file_directory
The fully-qualified path of the HFS directory for all MERVA Link USS processing trace files. This parameter is mandatory if MERVA Link USS processes must trace their activity. The trace file directory name can be up to 32 characters long.

tpi_trace_level
The trace level for inbound SNA APPC processes. Valid trace levels are 0, 1, 2, 3, and 9. A trace is not written for trace level 0.

tpi_trace_wrap_limit
The trace file naming scheme for inbound SNA APPC processes. If the wrap limit is 0, the trace file names contain a date/time stamp and are not reused. If the wrap limit is 1 to 255, the trace file names contain a wrap index with the maximum index as specified by the wrap limit. These file names are reused, and old trace files are overwritten.

tci_trace_level
The trace level for inbound TCP/IP processes. Valid trace levels are 0, 1, 2, 3, and 9. A trace is not written for trace level 0.

tci_trace_wrap_limit
The trace file naming scheme for inbound TCP/IP processes. If the wrap limit is 0, the trace file names contain a date/time stamp and are not reused. If the wrap limit is 1 to 255, the trace file names contain a wrap index with the maximum index as specified by the wrap limit. These file names are reused, and old trace files are overwritten.

ACT ASP Parameters

MERVA Link application support processes (ASPs) are not required to provide the MERVA Link USS Gateway function. ASPs are supported by MERVA Link USS of MERVA ESA V4.1 with very limited functionality for installation verification and test purposes only. For example, a MERVA Link USS receiving ASP cannot deliver an inbound message to a MERVA messaging application. For this reason, the customization of a MERVA Link USS ASP is not described.

ACT ISC Parameters

The ACT ISC parameters and their valid parameter values are:

partner_node
The name of the MERVA Link node in the applicable partner system. This parameter is mandatory. The partner node name is the key into this ACT ISC entry.

symbolic_destination
The name of a Side Information profile defined in APPC/MVS. This parameter is optional. The symbolic destination name provides the SNA APPC intersystem connection parameters to establish a conversation with the applicable partner system. If this parameter is not specified, an SNA APPC connection cannot be established to the partner system.

msg_tp_name
The name of the MERVA Link receiving TP in the partner system. This parameter is optional. The TP name specified in this parameter overwrites the partner TP name specified in the APPC/MVS Side Information profile. If this parameter is not specified, the TP name specified in the SI profile applies.

partner_host_name
The name of the partner host system for a TCP/IP connection. The partner host name can consist of up to 64 characters. Any format of an IP host name can be specified that can be translated by the local host or by a domain name server into a host IP address. Partner host name formats are, for example, the dotted decimal representation of the host IP address, and the fully-qualified host domain name.

Numbers supplied as address parts in standard dotted-decimal notation can be decimal, hexadecimal, or octal. Numbers are interpreted in C language syntax as specified by the OS/390 C function **inet_addr()**.

msg_port_number
The TCP port number in the partner system that represents the receiving MERVA Link process in the partner system. This parameter is mandatory if the partner host name is specified.

The TCP port number has a value between 1024 (1KB) and 65535 (64KB-1). It must be agreed upon between the cooperating hosts. The sample TCP port number for MERVA Link instance 1 is 7110.

password_encryption
The algorithm used to encrypt a conversation security password before it is sent to a partner system. This parameter is optional. The parameter values are **crypt** or **basic**. This parameter applies only to a TCP/IP connection.

The basic password encryption algorithm can be understood by all MERVA Link TCP/IP implementations. It is used if this parameter is not specified. The password encryption algorithm that uses the **crypt()** function, an unrestricted DES function, can be used only for a connection to another MERVA Link USS system. Other MERVA Link implementations cannot decrypt a password encrypted with this algorithm.

Generating a Configuration File from an Active ACT

Use the ACC command **cxf** (export ACT configuration to a unique file) to generate a configuration file based on the parameters of the active ACT. The ACT parameter groups are all contained in one file.

Use the ACC command **cxs** (export ACT configuration to separate files) to generate a main configuration file that contains the ACTH parameter group and **INCA:** and **INCC:** statements for the configuration include files that are also generated. The command **cxs** generates one include file for each ACT ASP and ISC entry in the directory of the main include file. These include files have the prefixes **cfga** or **cfgc**.

An exported configuration file contains the names of all supported configuration parameters regardless of whether parameter values are available. It can be used as a data entry map for a new configuration file.

Customizing MERVA Link USS Conversation Security

A conversation security function is provided by MERVA Link USS when messages are exchanged using TCP/IP services. SNA APPC conversation security is provided by APPC/MVS.

Conversation security information is the information that authorizes a client process to access the resources of a server process. A server process specifies whether a client process must provide conversation security information, or whether it provides its service without client authentication at the conversation level.

Conversation security information is a client **user ID** (called **user name** in the USS environment), and a client user **password**. Passwords are considered to be confidential, and must not be stored in plain text format on any storage media. This is why conversation security information must be handled separately from MERVA Link USS configuration files in MERVA Link USS security files.

The MERVA Link USS Security File Facility consists of a set of MERVA Link USS security files in the security subdirectory of the MERVA USS instance directory, and the MERVA Link USS Conversation Security Control Application (ACS) that provides for specifying, storing, and modifying conversation security information.

Conversation Security Files

MERVA Link USS Security Files provide for storing confidential security information (client user passwords). Confidential information must be stored in encrypted form.

The MERVA Link USS Application Control Daemon (ACD) combines information from a configuration file and security files to create an ACT.

The data in a security file is encrypted. The local host identifier (TCP/IP address) is used as part of the encryption algorithm. This has two consequences for the portability of security files:

- Security files cannot be moved or copied to another host and decrypted in this host. The recovered plain text is unreadable.
- Security files become unreadable when the TCP/IP address of the local host is changed. You must regenerate all security files using the MERVA Link USS ACS application in this case.

Conversation Security Control Application

The MERVA Link USS Conversation Security Control Application (ACS) maintains conversation security information in MERVA Link USS security files. The MERVA Link USS Daemon (ACD) refers to the information in the security files when it creates the MERVA Link USS ACT. MERVA Link USS message processing programs refer only to conversation security information in the ACT.

The ACS Program

The ACS program name is **ekaacs**. The command **ekaacs** is used to call the ACS in an OS/390 USS shell environment. The ACS can be executed in an OS/390 TSO USS shell or in a remote login shell at a remote host.

The program **EKAACS** is used to execute the ACS in an OS/390 batch environment.

The ACS Execution Environment

The ACS must have access to an active MERVA Link USS ACT. This is why the MERVA Link USS Daemon must be active when the ACS is called.

The fully-qualified path name of the MERVA USS Instance Directory must be provided as the first ACS command parameter when the USS shell environment variable **MERVA_DIR** is not defined.

The ACS Execution Modes

The ACS supports the following execution modes that differ in the way the security information is provided:

- In interactive mode, ACS prompts the operator for conversation security information. This is the default mode. A dialog can be terminated at any point by entering **end** or **exit**.
- In command parameter mode, the conversation security information is provided as a set of keyword parameters. Interactive and command parameter modes can

be mixed. This means that ACS prompts the operator in interactive mode for those conversation security information items that are not provided as command parameters.

- In batch mode, the conversation security information is retrieved by the ACS from standard input (**stdin**). ACS batch mode is supported in an OS/390 USS shell environment and in an OS/390 batch job environment. The standard input file can contain conversation security information for more than one partner node.

The ACS Command Parameters

The ACS command parameters can be grouped into the following parameter classes:

- MERVA USS instance directory parameter
- ACS execution control parameters
- Conversation security information parameters

MERVA USS Instance Directory Parameter

The fully-qualified path name of the MERVA USS instance directory must be specified as the first ACS command parameter if this information is not available from the applicable USS shell environment variable. The character **/** at the beginning of the first ACS command parameter indicates that the MERVA USS instance directory is specified.

ACS Execution Control Parameters

These parameters control the ACS execution mode. ACS execution control parameters have the format of single-character keywords:

h Causes ACS to print help information and then terminate. The character **?** has the same meaning.

c Sets confirm mode. Confirm mode means that a password must be entered twice, and a confirmation of a complete set of conversation security information is requested before it is processed. If confirm mode is not requested, a password must not be re-entered, and the parameter set is processed as soon as all of its elements are available.

v Sets verbose mode. Additional explanations and operator messages are displayed by ACS in verbose mode.

s Causes ACS to retrieve conversation security parameters from standard input (stdin). If this parameter is specified, ACS runs in nonconfirm mode automatically.

Conversation Security Information Parameters

These parameters provide for specifying conversation security information as ACS command parameters. ACS security information parameters have the format of keyword parameters (parameter keyword followed by parameter data). A parameter keyword and its data must be specified as two tokens separated by one or more blanks. The security information parameter keywords are:

node The partner MERVA Link node. The partner MERVA Link node identifies the partner system that can be accessed using the client user name and password, and identifies the ACT ISC entry that must be modified.

user The client user ID (user name).

pswd The client user password.

Any subset of the security information parameters can be specified when the ACS is called. The ACS prompts for the missing security information items.

Note: If the client user password is specified as an ACS command parameter, the password can be displayed as part of the output of an OS/390 USS lp command (list active processes) while the ACS process is active. The execution time of an ACS process can be minimized by specifying all required security information items as ACS command parameters, and execute ACS in nonconfirm mode.

If the client user password is not specified as ACS command parameter, the operator is prompted for this information.

Sample ACS Commands

The ACS command **ekaacs /u/merva1 h** sets the USS shell environment variable for the MERVA USS instance directory and displays ACS help information.

The ACS command **ekaacs v c** starts the ACS in verbose interactive mode, prompts the operator for all security information, and requests confirmation. The USS shell environment variable for the MERVA USS instance directory must have been set.

The ACS command **ekaacs node pnode1 user pn1user c** starts the ACS in normal interactive mode, prompts the operator for the client user password, prompts the operator to enter the password again, and requests confirmation.

The ACS command **ekaacs node pnode1 user pn1user pswd pn1pswd** handles the complete security information without asking for a confirmation.

The ACS Standard Input File

The ACS retrieves the conversation security information for a set of partner systems from standard input (**stdin**) if it is started with the control parameter **s**. The conversation security information in a standard input file has the following format:

- The file is structured as a set of lines of up to 80 characters. The security information of one partner node is specified in one line.
- The security information consists of three tokens, optionally followed by a free-form comment.
- The three tokens are:
  - MERVA Link partner node name
  - Client user ID (user name)
  - Client user password

  The tokens have a maximum of 8 significant characters and are separated by one or more blanks. Tokens with more than 8 characters are truncated.
- Security information lines with less than three tokens, empty lines, and lines starting with a hash character (**#**) are ignored.

A sample ACS standard input file is shown below.

```
#-------------------------------------------------------------
# ACS Standard Input Conversation Security Information
#-------------------------------------------------------------

#-------------------------------------------------------------
# p_node      user_name    password   comment
#-------------------------------------------------------------
  A1          a1user       a1pswd     partner node A1
  W1          W1USER       W1PSWD     partner node W1
#-------------------------------------------------------------
```

Figure 127. Sample ACS Standard Input File

The ACS Batch Mode

ACS in batch mode can run in both an OS/390 batch environment and a USS shell environment. The USS shell environment supports input from the terminal and input from an HFS file in an OS/390 USS shell and in an rlogin shell.

ACS Batch Sample for USS Shell

Standard USS shell rules apply for passing standard input data to ACS. Some commands to start the ACS in USS shell batch mode are:

- **ekaacs s**

  Starts the ACS in batch mode and expects security information from the terminal. Enter the three tokens of conversation security information (partner node name, client user name, and password) separated by blanks in one line, and press ENTER. The set of security information is processed, and you can enter another set.

  To end the terminal input (indicate End Of File), press Ctrl+D in an rlogin shell, or enter the keyword **end** or **exit**. The keywords end or exit must be used in the OS/390 USS shell environment because Ctrl+D is not supported.

- **ekaacs s < secfl**

  Starts the ACS in batch mode and reads security information from the file **secfl** in the current directory. The output of the ACS execution goes to the terminal. You can use the redirect operator (>) to write the ACS output to an HFS file.

- **cat secfl | ekaacs s**

  Has the same effect as **ekaacs s < secfl**. The cat command writes the content of file secfl to stdout. The pipe command (|) passes stdout as stdin to ekaacs.

ACS Batch Sample for OS/390 Batch

A sample OS/390 batch job to generate MERVA Link USS security files is shown below.

```
//........ JOB ..................
//*-------------------------------------------------------------------
//*        EKAACS EXECUTION IN OS/390 MVS ENVIRONMENT
//*-------------------------------------------------------------------
//EKAACS   EXEC PGM=EKAACS,REGION=4M,
//         PARM='//u/merva1 s v'
//STEPLIB  DD DSN=hlq.SDSLLIB,DISP=SHR
//SYSPRINT DD SYSOUT=*
//SYSIN    DD *
  A1          a1user       a1pswd
  W1          W1USER       W1PSWD
/*
//
```

Figure 128. Sample OS/390 ACS Batch Job

The ACS program EKAACS is contained in the MERVA Link USS library. It is called in verbose batch mode in this example. Verbose mode causes information about successfully generated security files to be written to stdout (SYSPRINT). The name of the MERVA USS instance directory (for example **/u/merva1**) must be specified as the first program parameter in OS/390 MVS batch mode.

The SYSPRINT and SYSIN DD statements identify the stdout and stdin files.

Customizing APPC/MVS for MERVA Link USS

MERVA Link USS uses APPC/MVS to support the interconnection of MERVA systems within an SNA network. The following MERVA Link USS programs use APPC/MVS services for communication with MERVA Link programs in partner systems:

EKATPI The MERVA Link USS Inbound SNA APPC transaction program (TP). It is scheduled by the APPC/MVS transaction scheduler ASCH in an APPC/MVS initiator to handle an inbound SNA APPC conversation.

The characteristics of EKATPI are specified in an APPC/MVS TP profile. The sample TP profile name is **EKAR1**. The APPC/MVS transaction scheduler ASCH refers to this profile in order to start program EKATPI.

EKATPO The MERVA Link USS Outbound SNA APPC transaction program (TP). It runs as part of a MERVA Link USS process in an APPC/MVS initiator (routing SNA APPC to SNA APPC) or in an OS/390 USS process region (routing TCP/IP to SNA APPC).

APPC/MVS provides the following for customizing APPC/MVS resources:

- An inbound APPC/MVS transaction program (TP) is defined in an APPC/MVS TP profile. The TP profile contains all parameters that are necessary to start the transaction program in an OS/390 region.
- An APPC/MVS side information profile defines an SNA APPC partner TP. It is the means by which an outbound TP identifies its partner TP.

APPC/MVS TP Profile for MERVA Link USS

The MERVA Link USS receiving transaction program (TP) that handles inbound messages in the OS/390 system must be defined in an APPC/MVS TP profile. The most important information in an APPC/MVS TP profile is the OS/390 JCL to schedule the MERVA Link USS inbound TP **ekatpi** in an APPC/MVS initiator.

APPC/MVS and MERVA Link USS provide the following methods for running the MERVA Link USS inbound TP:

- Execute EKATPI from a PDSE
- Invoke BPXBATCH to execute ekatpi from the HFS
- Invoke BPXBATCH to execute an ekatpi shell script

The methods are explained and samples are shown in the following sections. The sample profile names are EKARP1, EKARH1, and EKARS1. EKAR1 is used elsewhere in this book as the sample MERVA Link USS TP profile name.

Running EKATPI as a PDSE Member (EKARP1)

The MERVA Link USS inbound TP **ekatpi** can reside in an OS/390 PDSE (Partitioned Data Set Extended), also called a program library. The TP name in a PDSE is **EKATPI**.

When APPC/MVS schedules a TP from a PDSE, the started transaction does not run in a USS environment initially. So, the TP profile and the TP itself must define the mandatory USS environment variables.

Sample JCL in a TP profile for MERVA Link USS (EKARP1) that executes EKATPI as a PDSE Member is shown below.

```
 TP name: EKARP1
 Level  : SYSTEM          ID . . . :
****************************** Top of Data ***************************
//xxxxxxx  JOB  xxxxxxxxxxxxxxxxxxxxxxxxxx
//EKAR1    EXEC PGM=EKATPI,
//              PARM='//u/merva1 TZ=CET-1 '
//STEPLIB  DD   DSN=hlq.SDSLLIB,DISP=SHR
//SYSPRINT DD   DSN=hlq.STDOUT,DISP=SHR
****************************** Bottom of Data ************************
```

Figure 129. Sample APPC/MVS TP Profile EKARP1

The sample TP profile in Figure 129 defines the MERVA Link USS receiving TP that is associated with MERVA USS instance 1. It can receive messages from all kinds of MERVA Link partner systems.

The format and content of the JOB statements depends on requirements of the OS/390 installation:

- The statement **//EKAR1 EXEC** starts the job step and identifies the program to be run (EKATPI). Parameters must be passed to EKATPI as follows:
  - The first slash (**/**) identifies the end of the runtime options (they are empty in this example).
  - The string **/u/merva1** is the first program parameter (arg1), which specifies the name of the directory containing the MERVA USS instance.
  - The string **TZ=CET-1** is the second program parameter (arg2), which specifies the local time-zone in the format of the corresponding OS/390 USS environment variable. If this parameter is missing or incorrect, the timestamps shown in MERVA Link USS processing traces might be incorrect.
- The statement **STEPLIB DD** identifies the location of the executable MERVA Link inbound TP. Program EKATPI resides in an OS/390 program library (PDSE).
- The statement **SYSPRINT DD** defines the standard output file (stdout) to be used by EKATPI. It is an optional statement. Normally, data is not written to stdout.

Running EKATPI as an HFS Program (EKARH1)

The MERVA Link USS inbound TP **ekatpi** can reside in an HFS directory, for example, /usr/lpp/merva/bin/, and be executed in that format. However, an HFS executable cannot be called directly in OS/390 JCL. Instead, the OS/390 utility **BPXBATCH** must be invoked to call the HFS program.

BPXBATCH establishes a USS environment before it calls an HFS program. The USS environment variables required by the MERVA Link USS TP must be provided in the BPXBATCH parameter (PARM=).

Sample JCL in a TP profile for MERVA Link USS (EKARH1) that executes EKATPI as an HFS program is shown below.

```
 TP name: EKARH1
 Level  : SYSTEM          ID . . . :
****************************** Top of Data ***************************
//xxxxxxx  JOB  xxxxxxxxxxxxxxxxxxxxxxxxx
//EKAR1    EXEC PGM=BPXBATCH,
//              PARM='PGM /usr/lpp/merva/bin/ekatpi'
//STDENV   DD   PATH='/u/merva1/cmd/ekatpi.env',PATHOPTS=ORDONLY
//STDOUT   DD   PATH='/u/merva1/cmd/ekatpi.out',
//              PATHOPTS=(OWRONLY,OCREAT),PATHMODE=SIRWXU
****************************** Bottom of Data ************************
```

Figure 130. Sample APPC/MVS TP Profile EKARH1

The sample TP profile in Figure 130 defines the MERVA Link USS receiving TP that is associated with MERVA USS instance 1.

The format and content of the JOB statements depends on the requirements of the OS/390 installation:

- The statement **//EKAR1 EXEC** starts the job step and identifies the program to be executed (BPXBATCH). Parameters must be passed to BPXBATCH as follows:
  - PGM must be specified to identify the executable as an HFS program.
  - The string **/usr/lpp/merva/bin/ekatpi** is the fully-qualified name of the HFS program to be called by BPXBATCH.
  - In this example, the MERVA Link USS DLLs are assumed to be contained in the directory **/usr/lpp/merva/lib**. The program **ekatpi** sets this LIBPATH automatically based on the path specified in the fully-qualified name of the HFS program.
  - Parameters for **ekatpi** can follow the program path name. If the MERVA USS instance directory is not specified, as in this example, it must be specified in an environment file (STDENV).
- The statement **STDENV DD** identifies the HFS file containing the OS/390 USS environment variables that must be set by BPXBATCH before the HFS program is called. Specifying the mandatory MERVA USS instance directory and other environment variables in STDENV is a more flexible way to customize the ekatpi environment.

  A sample environment file for the MERVA Link USS inbound TP ekatpi is:

  ```
  MERVA_DIR=/u/merva1
  TZ=CET-1
  ```

The environment variables are set by BPXBATCH as specified in the environment file. A blank line results in an empty environment variable. Variable symbols, such as $LIBPATH, are used as specified rather than expanded as in a shell script.

- The statement **STDOUT DD** defines the standard output file (stdout) to be used by EKATPI. It is an optional statement. Error information can be written to stdout if the MERVA Link inbound TP cannot be successfully started. Normally, data is not written to stdout.

Running EKATPI as a Shell Script (EKARS1)

The MERVA Link USS inbound TP **ekatpi** can reside in an HFS directory, for example, /usr/lpp/merva/bin/, and be executed as part of an OS/390 USS shell script. The shell script sets up the required USS environment before the HFS program is called.

An OS/390 USS shell script residing in the HFS cannot be called directly in OS/390 JCL. The OS/390 utility **BPXBATCH** must be used to set up an environment to run an HFS executable, and invoke the shell script.

BPXBATCH can establish a USS environment that corresponds to a USS user's interactive shell environment before it runs a shell script. The USS environment variables required by the MERVA Link USS TP can already be part of this environment.

Sample JCL in a TP profile for MERVA Link USS (EKARS1) that runs EKATPI as part of a shell script is shown below.

```
 TP name: EKARS1
 Level  : SYSTEM          ID . . . :
****************************** Top of Data ***************************
//xxxxxxx  JOB  xxxxxxxxxxxxxxxxxxxxxxxxx
//EKAR1    EXEC PGM=BPXBATCH
//STDIN    DD   PATH='/u/merva1/cmd/ekatpi.cmd',PATHOPTS=ORDONLY
****************************** Bottom of Data ************************
```

Figure 131. Sample APPC/MVS TP Profile EKARS1

The sample TP profile in Figure 131 defines the MERVA Link USS receiving TP that is associated with MERVA USS instance 1. It can receive messages from all kinds of MERVA Link partner systems.

The format and content of the JOB statements depends on requirements of the OS/390 installation:

- The statement **//EKAR1 EXEC** starts the job step and identifies the program to be run by BPXBATCH. No parameters are passed to BPXBATCH, so the BPXBATCH default parameters apply.
- The statement **STDIN DD** identifies the shell script to be executed by BPXBATCH. A sample shell script for the MERVA Link USS inbound TP **ekatpi** is:

  ```
  export MERVA_DIR=/u/merva1
  export TZ=CET-1
  /usr/lpp/merva/bin/ekatpi
  ```

  Use any text editor to create and update the shell script, and ensure that it can be executed after it is created.

- Because the MERVA Link USS inbound TP must run in the foreground of the USS shell that executes this script, the line calling **ekatpi** must not end with an ampersand (&). The inbound APPC/MVS conversation cannot be connected to the inbound TP if it runs in the background.
- In this example, the MERVA Link USS DLLs are assumed to be contained in the directory **/usr/lpp/merva/lib**. The program **ekatpi** sets this LIBPATH automatically based on the path specified for **ekatpi** in the shell script.

APPC/MVS Side Information for MERVA Link USS

The MERVA Link receiving processes in the partner LUs must be defined in APPC/MVS Side Information. The name of a Side Information profile is also called a **Symbolic Destination Name**. An APPC/MVS Side Information profile contains the parameters TP Name, Mode Name, and Partner LU.

The **TP Name** identifies the MERVA Link receiving process in the partner system. The sample MERVA Link TP names are:

EKAR MERVA Link CICS and IMS (APPC/MVS)

EKARI510 MERVA Link APPC/IMS

EKAR1 MERVA Link AIX and USS

EKAOSVR MERVA Link OS/2

This parameter can be dynamically overwritten by the partner TP name parameter in the MERVA Link USS partner node definition.

The **Mode Name** specifies the VTAM logon mode to be used for LU-LU sessions to the partner system. This mode must be defined in the local VTAM system and in the partner SNA node.

The **Partner LU** must specify the LU 6.2 that represents the partner system.

SNA APPC Conversation Security

Cooperative processing allows application programs to establish communications with partner programs on other systems, and to share work, data, and services between systems and across networks. This ability to access other programs and all the resources at their disposal poses special security considerations for installations that use cooperative processing.

In a client-server environment it is the server that sets the conversation security requirements for a conversation with a specific client (or for all clients). A client must be aware of the server's security requirements, and must provide the appropriate access security information.

MERVA Link USS can be considered as a client when it allocates a conversation to a partner system to send messages to a partner application, the server. MERVA Link USS can be considered as a server when it accepts an inbound conversation from a partner system to receive messages.

For more information about SNA APPC conversation security, refer to the appropriate documentation in the OS/390 library.

Customizing TCP/IP for MERVA Link USS

TCP/IP is the communication method most often used in a network of interconnected UNIX® hosts. A set of hosts that is interconnected by TCP/IP is called an **internet**. MERVA Link USS can execute in an OS/390 USS host that is part of an internet, and use TCP/IP services to send and receive messages. These services must be customized to support MERVA Link USS.

TCP/IP customization for MERVA Link USS must be provided in:

- TCP/IP Hosts Table (/etc/hosts or HOSTS.LOCAL)
- TCP/IP Client Network Services (/etc/services)
- TCP/IP InetD Configuration (/etc/inetd.conf)

OS/390 TCP/IP can be customized only by authorized system administrators, for example, OS/390 USS root users with user ID 0 (zero).

Hosts Table (/etc/hosts or HOSTS.LOCAL)

Every partner host used by MERVA Link as a partner system must be defined in the TCP/IP network by its host name. Usually, a name server is available that can map a partner host name to the applicable TCP/IP address.

If a name server is not available, or if you do not want to use the services of a name server for MERVA Link partner hosts, you must define the partner hosts in the applicable Hosts Table of your local OS/390 USS system, for example, **/etc/hosts**. For more information, refer to OS/390 USS TCP/IP documentation or contact your OS/390 USS system administrator.

Client Network Services (/etc/services)

The link between a TCP/IP port number and a service is established in an entry in the TCP/IP services file. This entry is represented by a text line in the file **/etc/services**. An authorized user can edit this file to create and update a TCP/IP service entry. The sample TCP/IP service entry for the MERVA Link USS message transfer service reads:

```
ekamsg1      7110/tcp          # MERVA Link Messaging Service
```

Internet Daemon Configuration (/etc/inetd.conf)

A TCP/IP service supported (scheduled) by the OS/390 Internet Superserver (InetD) is defined in the InetD configration in the HFS file **/etc/inetd.conf**. A TCP/IP service supported by InetD is also called an *InetD subserver*. The parameters of an InetD subserver must be specified in its InetD configuration entry. This entry is represented by a text line in the file **/etc/inetd.conf**. An authorized user can edit this file to create and update an InetD subserver entry.

The MERVA Link USS InetD subserver program **ekatci** requires a set of USS environment variables for correct operation. The environment variable for the MERVA instance directory (**MERVA_DIR**) identifies the MERVA USS instance. The **LIBPATH** environment variable identifies the location of the MERVA USS Dynamic Link Libraries (DLLs).

A LIBPATH is not set by default when **ekatci** is called in the OS/390 USS environment. If **ekatci** is called from a directory with the name **bin** (for example **/usr/lpp/merva/bin**), **ekatci** sets the LIBPATH to the corresponding **lib** directory

(for example **/usr/lpp/merva/lib**). If not, the LIBPATH must be set to the directory containing the MERVA USS DLLs before ekatci is called.

There are two techniques for setting up the environment for the MERVA Link USS InetD subserver, and the environment for a direct and an indirect call of program EKATCI. The content of the InetD subserver entry for the MERVA Link USS message transfer service depends on the technique used.

Direct Call of EKATCI

When program **ekatci** must be directly called by InetD:

- Program ekatci must be called out of a /bin/ directory that has a corresponding /lib/ directory. The /lib/ directory must contain the MERVA USS DLLs.
- The name of the MERVA USS instance directory must be specified as the first program parameter.

The sample InetD Subserver entry for a directly called MERVA Link USS message transfer service reads:

```
ekamsg1 stream tcp nowait merva1 /usr/lpp/merva/bin/ekatci ekatci /u/merva1
```

The InetD subserver is called out of the **bin** subdirectory of the MERVA USS installation directory **/usr/lpp/merva**. The PATH environment variable is therefore set by InetD to **PATH=/usr/lpp/merva/bin**. The InetD subserver program **ekatci** expects to find its DLLs in the corresponding **lib** directory. The LIBPATH environment variable is therefore set by ekatci to **LIBPATH=/usr/lpp/merva/lib**.

InetD passes the subserver name (`ekatci` in this example) as program name (arg[0]) to the called program. The called program does not get the fully-qualified program path name as an argument.

The first program argument (`/u/merva1` in this example) is used by program ekatci to set the MERVA USS instance directory environment variable to **MERVA_DIR=/u/merva1**.

A timezone parameter, for example **TZ=CET-1**, can be specified as second program argument. It tells ekatci to set the time zone environment variable as specified in that parameter. This parameter can be used to adjust the date/time displayed in a receiving process trace.

Indirect Call of EKATCI

Program **ekatci** can be called indirectly by InetD via a USS shell script if the environment setup provided by ekatci does not meet the requirements of the MERVA USS installation. In this case, InetD starts a USS shell (/bin/sh). The shell process executes a shell script, and the shell script sets up an environment and calls program ekatci.

The shell script can provide the complete set of environment variables required by program ekatci. As an alternative, it may specify a subset that enables ekatci to do the rest.

The contents of the sample InetD subserver entry for an indirectly called MERVA Link USS message transfer service are:

```
ekamsg1 stream tcp nowait merva1 /bin/sh ekatci /u/merva1/cmd/ekatci.cmd
```

The InetD subserver, a USS shell, is called from the **/bin** directory. The PATH environment variable is therefore set by InetD to **PATH=/bin**. This directory has no meaning for ekatci, and cannot be used to find the location of the MERVA USS DLLs.

InetD passes the full path name of the shell script (in this example, **/u/merva1/cmd/ekatci.cmd**) to the USS shell and the shell executes the script. The contents of the sample shell script ekatci.cmd are:

```
export MERVA_DIR=/u/merva1
/usr/lpp/merva/bin/ekatci
```

The shell process calls ekatci from the **bin** subdirectory of the MERVA USS installation directory **/usr/lpp/merva**. The shell sets the program path environment variable _ to **_=/usr/lpp/merva/bin/ekatci**. The program ekatci expects to find its DLLs in the corresponding **lib** subdirectory, so it sets the LIBPATH environment variable to **LIBPATH=/usr/lpp/merva/lib**.

Refreshing the InetD Process

A modified InetD configuration becomes automatically active when the InetD process is started again. To activate a modification immediately, InetD must be asked to refresh its configuration by sending a signal (SIGHUP) to the InetD process:

1. To find the InetD process_id, enter:

   ```
   ps -e | grep inetd
   ```

2. To send a SIGHUP to the InetD process, enter:

   ```
   kill -1 process_id
   ```

   where ***process_id*** is the ID returned from the previous **ps** command. The parameter **-1** of the kill commands indicates that the signal is a SIGHUP.

Chapter 8. MERVA-to-MERVA Financial Message Transfer/ESA (FMT/ESA)

MERVA-to-MERVA Financial Message Transfer/ESA (FMT/ESA) uses the capabilities of MERVA Link or MERVA-MQI Attachment to transfer SWIFT messages between two MERVA ESA systems in a way similar to the way MERVA ESA transfers messages via the SWIFT network.

FMT/ESA:

- Prepares SWIFT input messages for a follow-on transfer
- Requests that MERVA Link or MERVA-MQI Attachment transfer the messages from the sending to the receiving MERVA ESA system (messages can be transmitted either in SWIFT format or MERVA ESA queue format)
- Transforms the received SWIFT input messages to SWIFT output messages
- Routes (or lets MERVA Link or MERVA-MQI Attachment route) the output messages to target queues
- Provides a SWIFT acknowledgment for the messages transmitted from the sending MERVA ESA system (the acknowledgment may be generated either in the sending or in the receiving MERVA ESA system)
- Provides a SWIFT delivery notification (message type 011) (if requested by a received SWIFT input message and the customer)
- Journals SWIFT input and output messages and their acknowledgments
- Authenticates SWIFT input and output messages, if this is requested by the customer
- Checks SWIFT input messages (if using MERVA Link or MERVA-MQI Attachment) and output messages (only if using MERVA-MQI Attachment), if this is requested by the customer

When using MERVA Link, and when there is a message recovery for an inoperable MERVA Link ASP using the MERVA Link RECOVER command, FMT/ESA:

- Appends a PDE trailer generated by FMT/ESA to a SWIFT input message that can be routed to a SWIFT ready queue
- Appends a PDE trailer generated by the SWIFT Link to a SWIFT input message before MERVA Link transfers the message to the receiving MERVA ESA

When using MERVA-MQI Attachment, a PDE trailer does not need to be appended, because, in contrast with the MERVA Link RECOVER command, messages are not copied from the MERVA control queue during recovery.

You can also set up FMT/ESA to run on a single MERVA ESA system. This might be of interest for an installation that has one instance of MERVA ESA serving more than one bank, and where these banks currently exchange messages among each other via the SWIFT network. By using FMT/ESA, messages and acknowledgments are routed from the sender to the recipient and back to the sender without ever leaving the MERVA ESA system, even though they look and behave exactly like messages sent and received via the SWIFT network. This is a less expensive way to exchange messages.

To do this using MERVA Link, customize the back-to-back (BTB) environment as described in "Customizing a Synchronous Back-to-Back Test Environment" on page 193, and in the *MERVA for ESA Installation Guide*. To do this using MERVA-MQI Attachment, customize the process table as described in the *MERVA for ESA Installation Guide*.

FMT/ESA can be run in all environments supported by MERVA ESA.

Using FMT/ESA with MERVA Link

This section explains the functions provided by FMT/ESA and describes a sample scenario to give you an idea of how the functions can be used in two connected MERVA ESA systems using MERVA Link.

FMT/ESA Message Flow with MERVA Link

You have two options for acknowledging a SWIFT input message in the sending MERVA ESA system. Each option influences the way messages are passed within and between the two MERVA ESA systems.

- Option 1: generate the acknowledgment in the sending MERVA ESA.

  This option is highly recommended, because it avoids the need to send back acknowledgments from the receiving MERVA ESA, thereby increasing throughput.

  An acknowledgment is provided only for a SWIFT input message that was transformed and properly stored in a queue in the receiving MERVA ESA as:

  – A SWIFT delivery notification (if requested by the SWIFT input message)
  – Another SWIFT output message (if a delivery notification was not requested)

  MERVA Link indicates this by a proprietary confirmation and FMT/ESA reacts upon it.
- Option 2: generate the acknowledgment in the receiving MERVA ESA.

  With this option you have full control over when to send back the acknowledgment to the sending MERVA ESA:

  – MERVA Link routes the acknowledged SWIFT input message to a queue for received messages and to the MERVA Link control queue.
  – You process the data in the receiving MERVA ESA according to your needs.
  – You send back the acknowledgment.

  Some examples of times you want to send a delayed or immediate acknowledgment are:

  – When you want to store the generated SWIFT output message in a database that is not controlled by MERVA ESA.

    In such a case, only after a message has been stored in the database is it considered to have been successfully received. Your application must route the acknowledged SWIFT input message from the queue for received messages to a MERVA Link send queue. When FMT/ESA retrieves the acknowledged SWIFT input message so that it can send an acknowledgment to the sending MERVA ESA, it discards the generated SWIFT output message from the receiving MERVA ESA by routing it to a MERVA ESA dummy queue, provided that the appropriate routing table is set up accordingly.
  – When you want to modify or replace the prepared acknowledgment in the transmitted SWIFT input message.

    The acknowledgment is contained in the MERVA ESA TOF field MSGACK. The modified acknowledgment is sent back to the sending MERVA ESA.

– When you want to add one or more trailers to the transmitted SWIFT input message.

  Trailers can be stored in the MERVA ESA TOF field SWTRAIL in different data areas. The added trailers are sent back to the originally sending MERVA ESA.

– When you want to inform the operator of the MERVA Link in the sending MERVA ESA of errors detected by FMT/ESA (for example, during testing).

  FMT/ESA reports an error in the TOF field MSGACK. However, by means of an appropriate routing table, you can override the chosen option and prevent MERVA Link from sending back to the sending MERVA ESA a transmitted SWIFT input message containing an error message in field MSGACK.

In "Customizing MERVA Link for Use with FMT/ESA" on page 286 you find the description of how you can request the appropriate acknowledgment generation.

Acknowledgment and Delivery Notification

A SWIFT input message is first acknowledged to the sender before a delivery notification informs the sender that the corresponding SWIFT output message was successfully stored in the receiving MERVA ESA. This is independent of where the acknowledgment is generated (in the sending or receiving MERVA ESA).

You have full control over when to send back the delivery notification to the sending MERVA ESA. You can process the SWIFT output message in the receiving MERVA ESA before you send back the delivery notification, but the nature of this processing depends on where the acknowledgment is generated:

- If it is generated in the sending MERVA ESA:
  1. MERVA Link routes the delivery notification message to a queue for received messages and to the MERVA Link control queue. After successful routing, the acknowledgment is generated in the sending MERVA ESA.
  2. You process the SWIFT output message in the receiving MERVA ESA according to your needs.

     FMT/ESA stores the SWIFT output messages in a dedicated queue, the ISN control queue. You retrieve the SWIFT output message from this queue using the TOF field EKAAMSID as KEY2. EKAAMSID is the MERVA Link IAM message identifier. You determine the contents of EKAAMSID by scanning the queue for received messages that contains the delivery notifications. Each delivery notification is associated with a SWIFT output message by field EKAAMSID.
  3. You send back the delivery notification to the sending MERVA ESA.
- If it is generated in the receiving MERVA ESA, FMT/ESA stores both the SWIFT output messages and the delivery notifications in the ISN control queue of the receiving MERVA ESA:
  1. You have to decide how to handle the generated acknowledgment. You can send it back immediately to the sending MERVA ESA, or you can do some other processing before you send it back as described above. However, the acknowledgment has to be sent back prior to the delivery notification.

     When you send it back immediately, also route the acknowledged SWIFT input message to a queue for received messages and to the MERVA Link control queue. This is necessary for the association with the SWIFT output message described in the next step.

     Note: When the acknowledgment is sent back, FMT/ESA routes the SWIFT output message and the delivery notification to appropriate target queues. If the SWIFT output message or the delivery notification need

further processing, their target queues have to be defined in the MERVA ESA Function Table with field EKAAMSID as KEY1 or KEY2.

2. You process the SWIFT output message in the receiving MERVA ESA according to your needs. During processing, you have to save the contents of the field EKAAMSID.

   You retrieve the SWIFT output message from a queue using field EKAAMSID as KEY2 (or KEY1) depending on the definition in the MERVA ESA Function Table. You determine the contents of EKAAMSID by scanning the queue for received messages that contains the acknowledged SWIFT input messages. Each acknowledged SWIFT input message is associated with a SWIFT output message by field EKAAMSID.

3. You send back the delivery notification to the sending MERVA ESA.

   Before you can retrieve the associated delivery notification from a queue you have to modify field EKAAMSID saved in the previous step.

   The saved EKAAMSID has the following structure:

```
  Byte01   Byte02   Byte03   Byte04    ....    Byte15   Byte16
|--------|--------|--------|--------|--------|--------|--------|
```

   The modified EKAAMSID must have the following structure:

```
  Byte16   Byte01   Byte02   Byte03    ....    Byte14   Byte15
|--------|--------|--------|--------|--------|--------|--------|
```

Generating an Acknowledgment and Delivery Notification

Figure 132 shows the message flow when an acknowledgment is generated in the sending MERVA ESA.

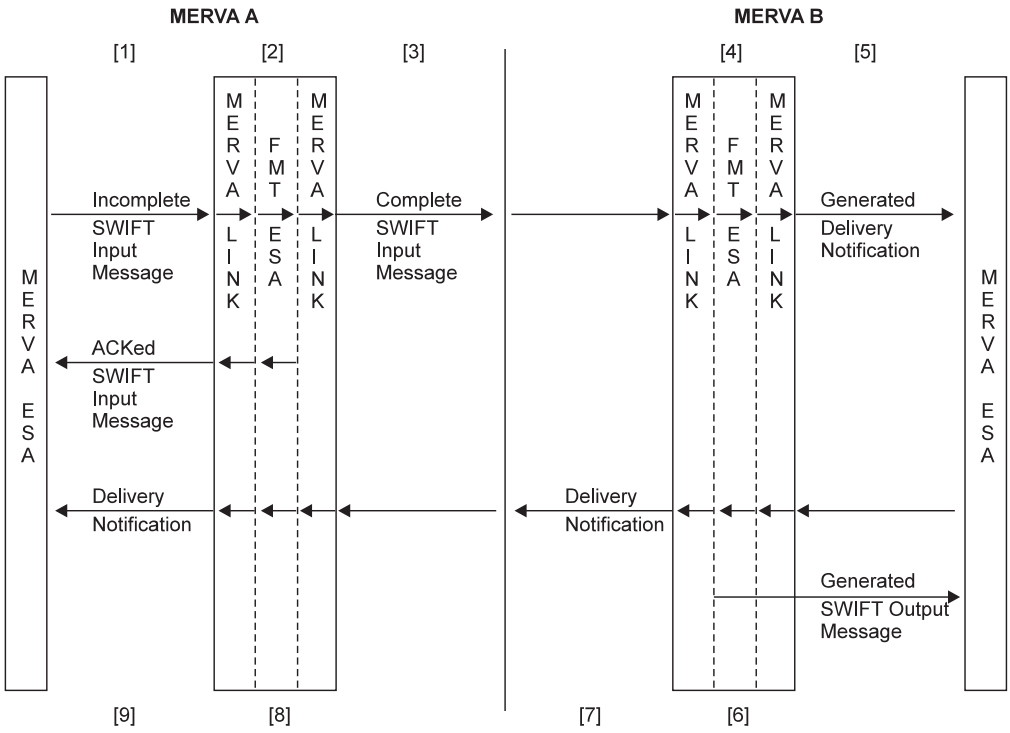

Figure 132. SWIFT Acknowledgment Generated at Message Sending Side (MERVA A)

Note: The following comments contain information that is useful when you want to write your own user exit that calls FMT/ESA. For a detailed description

of the FMT/ESA activities refer to Table 10 on page 294 in "Calling FMT/ESA from an MFS User Exit" on page 291.

Notes:

[1] A SWIFT input message is stored in one of the three MERVA Link send queues. It is the responsibility of either the data creation process within MERVA ESA or of an application outside MERVA ESA to put the message in a MERVA Link send queue.

[2] FMT/ESA completes and processes the input message passed from MERVA Link. FMT/ESA does the following:

1. Inserts an Input Sequence Number (ISN) into the message
2. Checks the SWIFT input message, if requested by the customer
3. Authenticates the financial application (FIN) message, if requested by the customer
4. Journals the message

Deviation from SWIFT message protocol

The Session Number is always 0 when a SWIFT input message is prepared by FMT/ESA and transmitted via MERVA Link rather than over the SWIFT network. This may be used in routing tables to distinguish the messages according to the transmission media.

[3] MERVA Link takes the completed SWIFT input message and sends it to the partner MERVA Link in MERVA B. In a dedicated queue, the ACK Wait Queue, the input message is stored by MERVA Link, until the acknowledgment can be provided by FMT/ESA in MERVA A.

[4] MERVA Link in MERVA B passes the received input message to FMT/ESA. FMT/ESA does the following:

1. Transforms the input message to a SWIFT output message
2. Inserts an output sequence number (OSN) into the output message
3. Authenticates the financial application (FIN) output message if requested by the customer
4. Journals the output message
5. Creates an acknowledgment for the output message and journals it
6. Writes the output message temporarily to a dedicated queue, the ISN control queue, if a delivery notification is to be created
7. Creates a delivery notification if requested by the input message and by the customer
8. Passes either the delivery notification or the output message to MERVA Link

The next items (5 to 9) are specific to the case in which a delivery notification is to be created:

[5] MERVA Link routes the delivery notification to one of the three MERVA Link send queues and to the MERVA Link control queue. The logic is defined in a routing table associated with the MERVA Link control queue.

This initiates the sending back of the delivery notification.

Note: If you want to process the SWIFT output message before you send back the delivery notification, refer to "Acknowledgment and Delivery Notification" on page 267.

After successful routing, MERVA Link confirms the proper receipt of the delivery notification to MERVA Link in MERVA A.

Processing of the acknowledged SWIFT input message:

[8] After receipt of the confirmation from MERVA Link in MERVA B, MERVA Link in MERVA A passes the 'waiting' input message (refer to item 3) to FMT/ESA. FMT/ESA does the following:

1. Generates the acknowledgment and writes it to the MERVA ESA TOF field MSGACK thereby creating an acknowledged SWIFT input message
2. Journals the acknowledgment
3. Passes the acknowledged input message to MERVA Link

[9] MERVA Link routes the acknowledged SWIFT input message to the target queue or queues according to the logic of a routing table associated with the MERVA Link control queue.

Processing the delivery notification:

[6] FMT/ESA determines that the message passed from MERVA Link is a delivery notification. Bypassing MERVA Link, it routes the SWIFT output message from the ISN control queue (refer to item 4) to the target queue or queues according to the logic of a routing table associated with the MERVA Link control queue.

[7] MERVA Link sends the delivery notification to the partner MERVA Link in MERVA A.

[8] MERVA Link in MERVA A passes the received delivery notification to FMT/ESA. FMT/ESA does the following:

1. Inserts an output sequence number (OSN) into the delivery notification
2. Journals the delivery notification
3. Creates an acknowledgment for the delivery notification and journals it
4. Passes the delivery notification to MERVA Link

[9] MERVA Link routes the delivery notification to the target queue or queues according to the logic of a routing table associated with the MERVA Link control queue.

Figure 133 shows the message flow when an acknowledgment is generated in the receiving MERVA ESA.

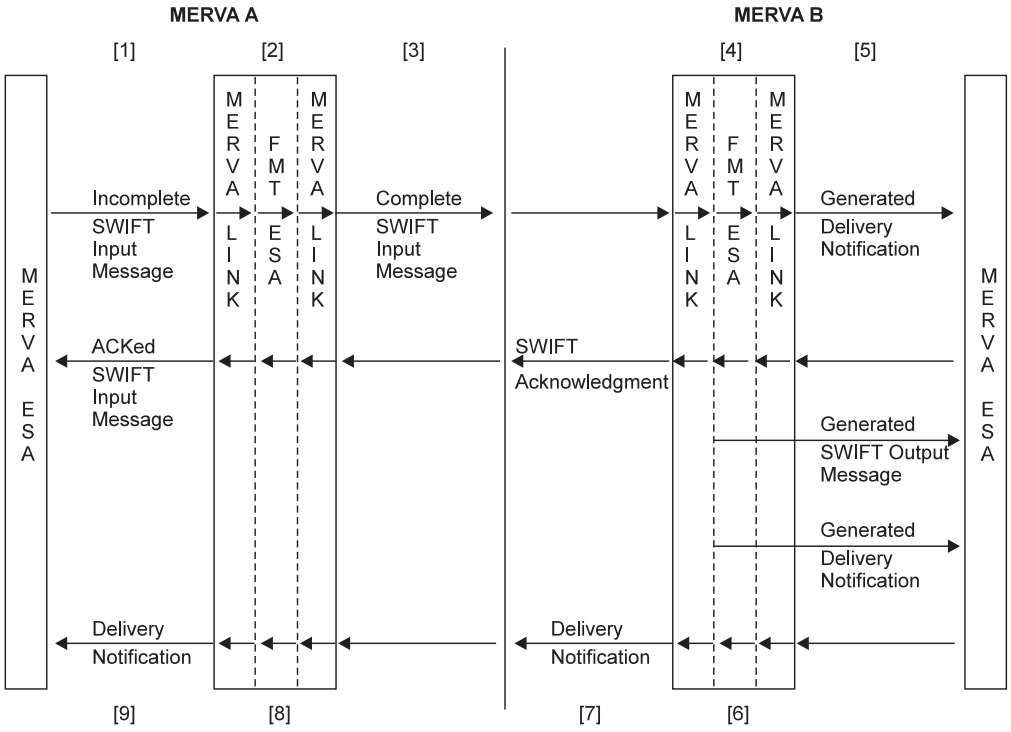

Figure 133. SWIFT Acknowledgment Generated at Message Receiving Side (MERVA B)

Note: The following comments contain information that is useful when you want to write your own user exit that calls FMT/ESA. For a detailed description of the FMT/ESA activities refer to Table 10 on page 294 in "Calling FMT/ESA from an MFS User Exit" on page 291.

Notes:

[1] A SWIFT input message is stored in one of the three MERVA Link send queues. It is the responsibility either of the data creation process within MERVA ESA or of an application outside MERVA ESA to put the message in a MERVA Link send queue.

[2] FMT/ESA completes and processes the input message passed from MERVA Link. FMT/ESA does the following:

1. Inserts an Input Sequence Number (ISN) into the message
2. Checks the SWIFT input message if requested by the customer
3. Authenticates the financial application (FIN) message if requested by the customer
4. Journals the message

Deviation from SWIFT message protocol

The Session Number is always 0 when a SWIFT input message is prepared by FMT/ESA and transmitted via MERVA Link rather than over the SWIFT network. This may be used in routing tables to distinguish the messages according to the transmission media.

[3] MERVA Link takes the completed SWIFT input message and sends it to the partner MERVA Link in MERVA B. In a dedicated queue, the ACK Wait

Queue, the input message is stored by MERVA Link and waits for an acknowledgment to come from MERVA B.

[4] MERVA Link in MERVA B passes the received input message to FMT/ESA. FMT/ESA does the following:

1. Transforms the input message to a SWIFT output message
2. Inserts an output sequence number (OSN) into the output message
3. Authenticates the financial application (FIN) output message if requested by the customer
4. Journals the output message
5. Creates an acknowledgment for the output message and journals it
6. Writes the output message temporarily to a dedicated queue, the ISN control queue
7. Creates the delivery notification if requested by the input message and writes it temporarily to a dedicated queue, the ISN control queue
8. Creates an acknowledgment for the input message
9. Passes the acknowledged input message to MERVA Link

[5] MERVA Link routes the acknowledged and (optionally) authenticated SWIFT input message (received from FMT/ESA, see item 4) to one of the three MERVA Link send queues and to the MERVA Link control queue. The logic is defined in a routing table associated with the MERVA Link control queue.

This initiates the sending back of the acknowledgment.

Note: If you want to modify the acknowledgment or add trailers, or do both, to the message (following the MAC trailer, if available):

1. Replace the MERVA Link send queue by another queue and route the input message both to this queue and to the MERVA Link control queue.
2. Make the modifications by means of your own application.
3. Route the modified message to a MERVA Link send queue only.

The next items (6 to 9) are specific to the case in which a delivery notification is to be created:

[6] FMT/ESA determines that the message passed from MERVA Link contains an acknowledgment. Bypassing MERVA Link, it routes:

- The SWIFT output message
- The delivery notification

from the ISN control queue (refer to item 4) to the target queue or queues according to the logic of a routing table associated with the MERVA Link control queue. For the delivery notification, one of the target queues must be a MERVA Link send queue.

This initiates the sending back of the delivery notification.

Note: If you want to process the SWIFT output message before you send back the delivery notification, refer to "Acknowledgment and Delivery Notification" on page 267.

Then FMT/ESA indicates to MERVA Link that the acknowledgment and the trailers are to be sent rather than the complete message.

Processing of the acknowledgment and the trailers:

[7] MERVA Link takes the acknowledgment and the trailers and sends them to the partner MERVA Link in MERVA A.

[8] MERVA Link in MERVA A passes the received acknowledgment, the received trailers, and the 'waiting' input message (refer to item 3) to FMT/ESA. FMT/ESA processes the passed data:

1. It writes the acknowledgment to the MERVA ESA TOF field MSGACK thereby creating an acknowledged SWIFT input message.
2. If available, it writes the trailers to different data areas of the MERVA ESA TOF field SWTRAIL.
3. It journals the acknowledgment.
4. It passes the acknowledged input message to MERVA Link.

[9] MERVA Link routes the acknowledged SWIFT input message to the target queue or queues according to the logic of a routing table associated with the MERVA Link control queue.

Processing the delivery notification:

[6] FMT/ESA determines that the message passed from MERVA Link is a delivery notification. FMT/ESA passes the delivery notification to MERVA Link.

[7] MERVA Link sends the delivery notification to the partner MERVA Link in MERVA A.

[8] MERVA Link in MERVA A passes the received delivery notification to FMT/ESA. FMT/ESA does the following:

1. Inserts an output sequence number (OSN) into the delivery notification
2. Journals the delivery notification
3. Creates an acknowledgment for the delivery notification and journals it
4. Passes the delivery notification to MERVA Link

[9] MERVA Link routes the delivery notification to the target queue or queues according to the logic of a routing table associated with the MERVA Link control queue.

Scenario Involving FMT/ESA with MERVA Link

In order to illustrate FMT/ESA, a sample scenario is provided. Its main components are:

- A set of queues defined in the MERVA ESA Function Table.

  A top frame MCB EKAETOP with the message type ETOP is provided for each of these queues. It helps you to distinguish whether messages were transmitted by FMT/ESA using MERVA Link or via the SWIFT network. This applies both when you display and when you print messages.

  If you prefer to display and print messages in the usual way, you only have to replace message type ETOP in the Function Table parameter 'FRAME=(ETOP,0BOT)' by 'FRAME=(0TOP,0BOT)'.
- The routing table EKARTSND that determines the appropriate send queue of the MERVA Link send queue cluster for prepared and authorized SWIFT input messages.
- The routing table EKARTSIM that controls the distribution of successfully received messages and handles errors occurred with sent or received messages.

Queues for the FMT/ESA with MERVA Link Scenario

The usage of each queue is as follows:

EKASWDE0 Data entry

A SWIFT input message can be entered at the screen.

EKASWAI0 Message authorization

In the FMT/ESA scenario, each SWIFT input message will be authorized.

Routing table EKARTSND is associated with EKASWAI0.

EKASWSND MERVA Link send queue

In this queue, all messages are stored that are ready to be sent to the partner MERVA Link. This might be:

- SWIFT input messages with or without a SWIFT acknowledgment, or with an error message. If available, the acknowledgment or the error message is contained in the MERVA ESA TOF field MSGACK.
- Generated delivery notifications.

EKASWVE0 Message verification

In this queue, those SWIFT input messages are collected which cannot be sent to the partner MERVA Link due to an error detected by FMT/ESA.

EKASWAWQ SWIFT acknowledgment wait queue

In this queue, the following SWIFT input messages are stored:

- Those for which the partner MERVA Link confirmed the successful receipt
- Those for which the SWIFT acknowledgment is outstanding

EKASWACK SWIFT acknowledged messages queue

These messages could be successfully correlated with the SWIFT acknowledgment.

Note: FMT/ESA always provides a positive acknowledgment. That is, tag 451: is always followed by '0'.

EKASWDMY Dummy Queue

As usual in MERVA ESA, it is used to get rid of obsolete messages. In the FMT/ESA scenario, this applies to:

- Confirmed MERVA Link status reports
- Received ('inbound') MERVA Link status reports, for which FMT/ESA detects an error during processing. These status reports were already routed by FMT/ESA to the local error queue EKASWLEQ

EKASWSDO Accepted SWIFT output messages

This queue contains those SWIFT output messages for which a SWIFT authentication was performed. The authentication result was either successful or that the output message need not be authenticated.

EKASWAO0 SWIFT output messages with or without authentication, delivery notifications

In this queue, those SWIFT output messages are stored that:

- Were authenticated with an unsuccessful result
- Were *not* authenticated, as the customer did not request the authentication
- Are delivery notifications received from the partner MERVA Link

EKASWEMQ SWIFT Erroneous Messages Queue

A SWIFT input message was received by the partner MERVA Link with a delivery return code > 0. Provided that FMT/ESA was able to process the message despite the delivery error, you may find both acknowledged SWIFT input messages (prepared for being sent back to the originally sending MERVA ESA system) and generated SWIFT output messages in this queue.

EKASWLEQ Local Error Queue

This queue holds the messages for which FMT/ESA detected an error while doing one of the following:

- At the message sending side, while processing an inbound status report
- At the message sending side, while processing a confirmed message (the successful receipt in the partner MERVA Link has been signalled)
- At the message receiving side, while processing a received SWIFT input message (as an inbound application message)

EKASWREQ Remote Error Queue

This queue contains those messages for which FMT/ESA in the partner MERVA Link detected an error during processing. The error message contained in TOF field MSGACK, was sent back from the receiving to the originally sending MERVA ESA system.

Remote errors can only be made visible:

- If the SWIFT acknowledgment generation by FMT/ESA in the receiving MERVA ESA system was requested
- If this option was not overridden by the logic in routing table EKARTSIM for FMT/ESA in the receiving MERVA ESA system

In addition to these application related queues, another three queues serving as control queues are provided.

EKASIMCQ MERVA Link application control queue

Routing table EKARTSIM is associated with EKASIMCQ.

EKAISNCQ Input sequence number (ISN) control queue

This queue was introduced for FMT/ESA, and is used to temporarily store a generated SWIFT output message or a generated delivery notification. It contains one ISN for each MERVA Link ASP (connection to a partner) that is part of FMT/ESA. FMT/ESA increments and inserts the appropriate ISN into each SWIFT input message that it processes for a specific ASP.

If you request either of the following, this queue must also be defined in the receiving MERVA ESA:

- The SWIFT acknowledgment generation in the receiving MERVA ESA
- The processing of delivery notifications

This queue must be defined with two keys in the MERVA ESA function table DSLFNTT:

- KEY1 represents the name of the ASP contained in the TOF field EKAASP.
- KEY2 represents the MERVA Link IAM message identifier contained in the TOF field EKAAMSID.

EKAOSNCQ Output sequence number (OSN) control queue

The second control queue introduced for FMT/ESA. It contains one OSN incremented and inserted into each SWIFT output message or delivery notification received by FMT/ESA.

For SWIFT output messages, the queue is required in the receiving MERVA ESA system only. For delivery notifications, the queue is also required in the sending MERVA ESA system.

Routing Table EKARTSIM

EKARTSIM is commonly used in FMT/ESA at the message sending and at the message receiving side. The following items should be kept in mind when using the routing table:

- EKARTSIM must be associated with the MERVA Link Application Control Queue (here: EKASIMCQ) of the appropriate MERVA Link ASP at both sides.
- At the message receiving side, SWIFT output messages and delivery notifications have message class 'LR', because they are generated from received SWIFT input messages known by MERVA Link as inbound application messages.

  SWIFT output messages and delivery notifications can be routed either by MERVA Link or by FMT/ESA depending on MERVA Link customization:

  – If routed by MERVA Link, the MERVA Link Application Control Queue *must* be an additional target queue. This is required by the MERVA Link Message Integrity Protocol for all inbound messages.
  – If routed by FMT/ESA, the MERVA Link Application Control Queue *must not* be a target queue.
- At the message receiving side, a routing error is forced when the following conditions are met:
  – A SWIFT input message is successfully received with MERVA Link delivery return code equal to '00'. The message class is 'LR'.
  – FMT/ESA detects an error when it processes the message. It routes the message to the local error queue EKASWLEQ.
  – When FMT/ESA finishes processing, MERVA Link routes this message with class 'LR'. According to its customization, the MERVA Link at the message sending side requests the SWIFT acknowledgment to be generated at the message sending side.

  The routing error is forced by the following statements in EKARTSIM:

```
IMSG     DSLROUTE TYPE=TEST,COND=(ACKRQ,'0',EQ),TRUE=END
         DSLROUTE TYPE=TEST,COND=(ACKRQ,'1',EQ),TRUE=END
         .
         .
         .
END      DSLROUTE TYPE=FINAL
```

ACKRQ = '0' or ACKRQ = '1' means that the SWIFT acknowledgment is to be generated at the message sending side.

The forced routing error interrupts the message transmission between the two MERVA Link partners. However, you can obtain specific information on the error from the message in queue EKASWLEQ.

- Alternatively, the MERVA Link at the message sending side may have requested that the message receiving side generate the SWIFT acknowledgment. If you do not want the receiving side to send to the sending side the local error message it generates when FMT/ESA processes a received SWIFT input message and an error occurs, you can override this by activating (that is, by removing the *---->>> from) the following statement in EKARTSIM:

```
*---->>> DSLROUTE TYPE=TEST,COND=(MSGACK,'{1:',EQ,SHORT),FALSE=END
         .
         .
         .
END      DSLROUTE TYPE=FINAL
```

After activation, the statement determines whether the TOF field MSGACK contains a SWIFT acknowledgment, whose first three characters are '{1:'. If it is not an acknowledgment, it is an error message. In this case a routing error is forced. As with the previous example, the message transmission is interrupted. All information, however, is available to you, as the erroneous message was already routed to the local error queue EKASWLEQ.

- You must not specify parameter 'TARGET' in the routing table statement 'DSLROUTE TYPE=FINAL'. Errors in processing of a routing table are handled by MERVA Link.

Forced Routing Error Indication

A routing error can be forced at the message receiving side only. It is dependent on the logic in the routing table EKARTSIM associated with the MERVA Link control queue (refer to “Routing Table EKARTSIM” on page 276). So it is your choice whether you want to force a routing error.

A SWIFT input message was received with a delivery return code equal to '00' contained in MERVA Link TOF field EKADELRC. During generation of the SWIFT output message, FMT/ESA detected an error. The message was then routed:

1. By FMT/ESA, to the local error queue with message class 'IE'.
2. By MERVA Link, to a nonexistent target queue with message class 'LR'.

   This forces a routing error, and the ASP becomes inoperable.

MERVA Link informs you of a routing error for a message which cannot be received (class 'LR') with the following hexadecimal diagnostic codes which belong to the category 'Receiving Process Error Diagnostic Information Type AS':

001A 0010 0044 0010

001A IM-ASPDU (application message) processing error.

0010 ROUTE message failed.

0044 DSLQMGT return code 68 (decimal).

The MERVA ESA routing scanner (DSLRTNSC) indicated that no routing target could be determined.

0010 DSLRTNSC reason code 16 (decimal).

Empty target list when final statement found.

Whenever you see these diagnostic codes, inspect the local error queue. If you find messages in this queue, look at them and read the error message they contain. If the error message starts with 'EKA8' it was issued by FMT/ESA. Other types of error messages ('DWS...' or 'DSL...') were issued by other MERVA ESA components under control of FMT/ESA. For more information, refer to *MERVA for ESA Messages and Codes*.

MERVA Link Message Classes for FMT/ESA

There are four message classes that are reserved for FMT/ESA. Each message class indicates an error situation found in four different processing stages of FMT/ESA.

The MERVA Link TOF field EKACLASS contains the 2-character message class indicators:

Class **Meaning**

SE Error detected before a SWIFT input message was sent to the partner MERVA Link. As a result, a complete SWIFT input message cannot be generated.

In the FMT/ESA scenario, such a message is routed to the verification queue EKASWVE0.

IE Error detected after a SWIFT input message was received in the partner MERVA Link. As a result, a SWIFT output message cannot be generated.

In the FMT/ESA scenario, such a message is routed to the local error queue EKASWLEQ at the message receiving side.

LE Error detected after the MERVA Link confirmation was received indicating that a SWIFT input message was successfully received in the partner MERVA Link. As a result, the SWIFT acknowledgment for the same SWIFT input message contained in the acknowledgment wait queue EKASWAWQ cannot be generated.

In the FMT/ESA scenario, such a message is routed to the local error queue EKASWLEQ at the message sending side.

PE Error detected after a MERVA Link inbound status report was received. The status report contains acknowledgment related data in the MERVA Link TOF field EKARDATA:

- Either the SWIFT acknowledgment or an error message generated by FMT/ESA or provided by a customer application in the receiving MERVA ESA system.
- One or more trailers added to the SWIFT input message by a customer application in the receiving MERVA ESA system.

As a result, the acknowledgment related data cannot be inserted into the appropriate SWIFT input message contained in the acknowledgment wait queue EKASWAWQ.

In the FMT/ESA scenario, such a message is routed to the local error queue EKASWLEQ at the message sending side.

FMT/ESA Scenario at the Message Sending Side

Figure 134 shows how messages are routed to different queues:

- Before the messages are transmitted by MERVA Link
- After messages or MERVA Link confirmations have been received from the partner MERVA Link

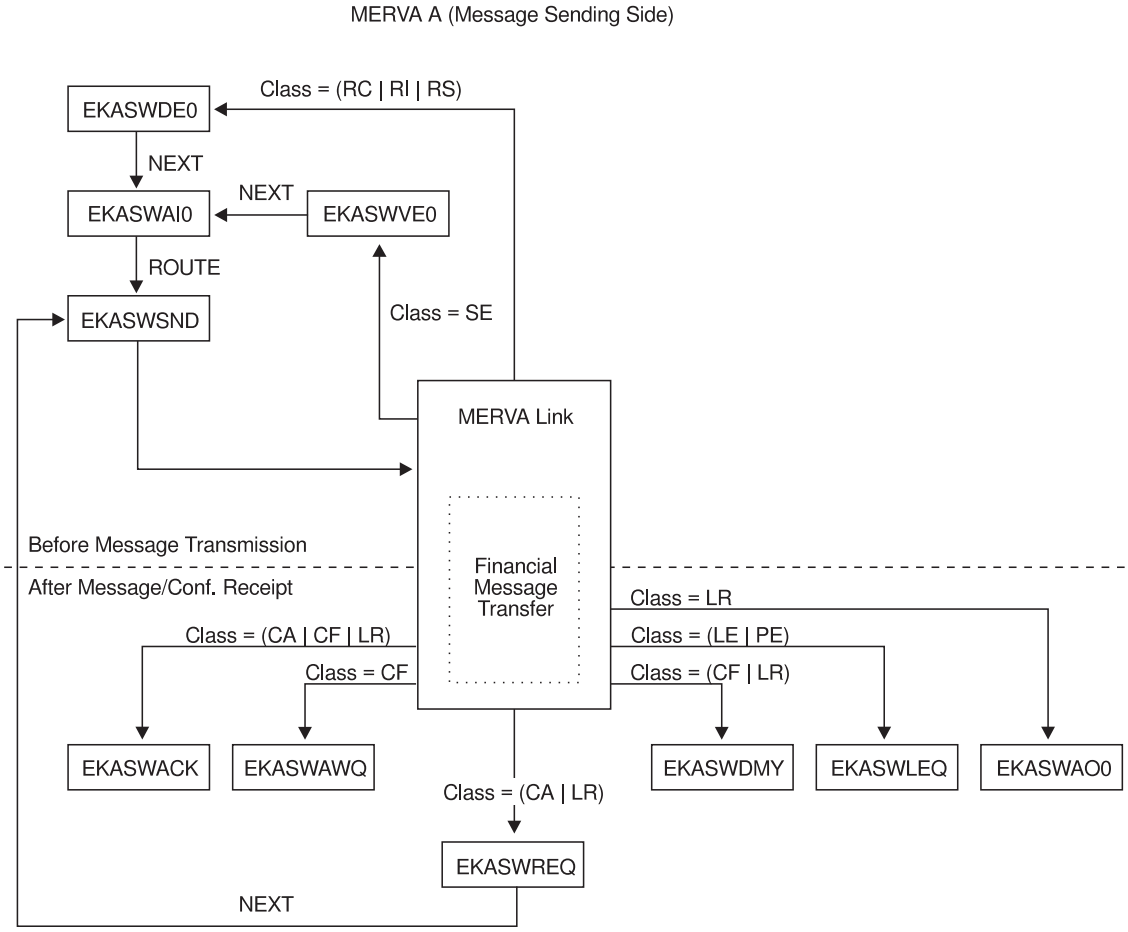

Figure 134. MERVA-to-MERVA Financial Message Transfer/ESA Scenario at Message Sending Side (MERVA A)

When a message is moved to another queue without using a routing table, this is shown by 'NEXT' at the arrows. It indicates that these queues have an entry in the MERVA ESA Function Table, where the parameter 'NEXT' was specified.

Before MERVA Link can send a message, routing table EKARTSND is involved. This is indicated by 'ROUTE'.

EKARTSND is responsible for the selection of a MERVA Link send queue. It is activated after a message left the authorization queue EKASWAI0. In the FMT/ESA scenario, the only send queue is EKASWSND. You may modify table EKARTSND and add one or two send queues according to your needs.

Routing table EKARTSIM contains the logic for message distribution to the other queues. The main routing criterion is the message class represented in the TOF field EKACLASS. In Figure 134 on page 279 this is shown by 'Class =' followed by one to three message classes. Messages with one of these classes will be routed to the same target queue. When a message class occurs for more than one target queue, additional conditions are checked in order to determine the desired queue.

For example, consider a message of the class 'CF' (Confirmed). Such a message may be routed to either of the three queues EKASWACK, EKASWAWQ, or EKASWDMY. Look at routing table EKARTSIM, where messages with CLASS = CF are treated:

```
⋮

TSTCF    DSLROUTE TYPE=TEST,COND=(CLASS,'CF',EQ),FALSE=TSTCA
*
         DSLROUTE TYPE=TEST,COND=(MTYPE,'011',EQ),TRUE=SETDQ             [1]
         DSLROUTE TYPE=TEST,COND=(RECRC,'00',EQ),TRUE=SETDQ              [2]
         DSLROUTE TYPE=TEST,COND=(MSGACK,'{1:',EQ,SHORT),TRUE=SETACKCF   [3]
*
         DSLROUTE TYPE=SET,TARGET='EKASWAWQ',GOTO=END
SETACKCF DSLROUTE TYPE=SET,TARGET='EKASWACK',GOTO=END
SETDQ    DSLROUTE TYPE=SET,TARGET='EKASWDMY',GOTO=END

⋮
```

Notes:

[1] The SWIFT message type represented in TOF field SWAHMT is checked. In EKARTSIM, the field is called MTYPE. If MTYPE is equal to '011', this indicates that a delivery notification application message was confirmed. A confirmation for this application message is obsolete and will therefore be discarded in the dummy queue EKASWDMY. MTYPE was not equal to '011'. The MERVA Link receipt return code represented in TOF field EKARECRC is checked. In EKARTSIM, the field is called RECRC.

[2] If RECRC is equal to '00', this indicates a confirmed MERVA Link status report. A confirmed status report is obsolete and will therefore be discarded in the dummy queue EKASWDMY. RECRC was not equal to '00'. As FMT/ESA always sets a receipt return code of '00', this indicates that the receipt return code is not available.

[3] If the first three characters contained in MSGACK are equal to '{1:', this indicates an acknowledged SWIFT input message, as the SWIFT acknowledgment starts with these unique characters. Therefore the message will be routed to the acknowledged messages queue EKASWACK. The first three characters in MSGACK were unequal to '{1:'. As a message of class 'CF' never has an MSGACK field containing an error message, this indicates that the confirmed SWIFT input message was not yet acknowledged. Therefore the message will be routed to the acknowledgment wait queue EKASWAWQ.

According to this example, you can inspect routing table EKARTSIM for details on messages with other classes.

Appending a PDE Trailer

When a MERVA Link ASP becomes inoperable, you can use the MERVA Link **RECOVER** command. This command provides a means to copy in-process (IP) messages of an inoperable ASP in **CLOSED-HOLD** status to another queue that may serve as a send queue of another transmission medium (for example, a SWIFT Ready Queue).

Thus you can bypass the MERVA Link connection breakdown and send messages over the SWIFT network to the receiving MERVA ESA. Before you can recover a message via the **RECOVER** command, you set an inoperable ASP to status **CLOSED-HOLD**. Enter the following commands on the MERVA Link control panels:

- **hold** *aspname*
- **aclose** *aspname*

  The ASP *aspname* is now in status 'CLOSED - HOLD'.
- **recover** *aspname*

A message that was recovered via the **RECOVER** command, has message class **RC**. Figure 134 on page 279 shows that messages with class **RC** are routed to data entry queue EKASWDE0. You can easily replace this target queue by a SWIFT Ready Queue used in your installation.

Note: At the message receiving side, a special situation can occur: In addition to the class **RC**, you also have to check for the SWIFT message type 011 in your routing table. Messages of this type are delivery notifications. They must not be routed to a SWIFT Ready Queue. You can route them to a MERVA ESA dummy queue to discard them.

Different to other MERVA Link methods of message recovery, the **RECOVER** command may cause messages to be received twice: first via the SWIFT network, and then via MERVA Link after the ASP has become operational again. This is due to the fact that a message remains as an in-process message in the MERVA Link control queue after recovery.

MERVA Link indicates this to the receiver by adding the field EKAPDUPM containing the characters 'PDM' to the message.

FMT/ESA provides this information according to the needs of the SWIFT protocol:

- It generates a PDE trailer and appends it to the SWIFT input message with class 'RC'.
- It uses the PDE trailer generated by the SWIFT Link and appends it to the SWIFT input message that is about to be transferred via MERVA Link after the ASP has become operational again.

Directing SWIFT Input Messages to the SWIFT Link

If you decide to send and receive SWIFT messages using FMT/ESA, you still have the option to use the SWIFT Link for sending and receiving SWIFT messages via the SWIFT network.

MERVA Link offers you a facility to use another transmission medium:

1. Set a MERVA Link sending ASP to status **CLOSED-NOHOLD**.
2. Route SWIFT input messages with message class **RS**.

The AS status **OPEN-NOHOLD** is the initial and the normal AS status of an ASP. To change this status to **CLOSED-NOHOLD**, enter the following commands on the MERVA Link control panels:

- **hold** *aspname*
- **aclose** *aspname*
- **astart** *aspname*

The ASP *aspname* is now in status **CLOSED-NOHOLD**. The ASP routes messages in its send queue cluster with message class **RS** immediately to a queue or to queues specified by the routing table EKARTSIM associated with the application control queue. It does not transmit these messages to the partner ASP.

Figure 134 on page 279 shows that messages with class **RS** are routed to data entry queue EKASWDE0. With some additional logic in the routing table EKARTSIM, you can direct the SWIFT GPA and FIN messages to the SWIFT link ready queues as new target queues instead of to EKASWDE0.

The following excerpt from the MERVA ESA provided routing table DWSL1AI0 serves as an example for such a distribution logic. The appropriate routing table statements may be included in the routing table EKARTSIM:

```
⋮

TSTRS    DSLROUTE TYPE=TEST,COND=(CLASS,'RS',EQ),FALSE=TSTRI
**************** EXCERPT FROM ROUTING TABLE DWSL1AI0 ******************
*        ROUTE CORRECT MESSAGES TO THE RESPECTIVE READY QUEUE
*        DEFINE APPL FROM THE FIELD SWBHAPI
*        (APPLICATION IDENTIFIER IN THE BASIC HEADER)
APPLIC   DSLROUTE TYPE=DEFINE,FIELD=(APPL,SWBHAPI,,,,,VFIRST),         *
               EMPTY=VE0,NOTFND=VE0
*
*        TEST FOR F OR A OR L
         DSLROUTE TYPE=TEST,COND=(APPL,'F',EQ),TRUE=FIN
         DSLROUTE TYPE=TEST,COND=(APPL,'A',EQ),TRUE=GPA
         DSLROUTE TYPE=TEST,COND=(APPL,'L',EQ),TRUE=GPA,FALSE=VE0
*
*        SET THE READY QUEUE FOR GPA
GPA      DSLROUTE TYPE=SET,TARGET=('L1RGPAU'),GOTO=END
*
*        DEFINE PRTY FROM THE FIELD SWAHIPY (PRIORITY OF APPL HEADER)
FIN      DSLROUTE TYPE=DEFINE,FIELD=(PRTY,SWAHIPY,,,,,VFIRST),         *
               EMPTY=VE0,NOTFND=VE0
*
*        TEST FOR N OR U OR S
         DSLROUTE TYPE=TEST,COND=(PRTY,'N',EQ),TRUE=FINN
         DSLROUTE TYPE=TEST,COND=(PRTY,'U',EQ),TRUE=FINU
         DSLROUTE TYPE=TEST,COND=(PRTY,'S',EQ),TRUE=FINU,FALSE=VE0
*
*        NORMAL READY QUEUE FOR FINANCIAL APPLICATION
FINN     DSLROUTE TYPE=SET,TARGET=('L1RFINN'),GOTO=END
*
*        URGENT READY QUEUE FOR FINANCIAL APPLICATION
FINU     DSLROUTE TYPE=SET,TARGET=('L1RFINU'),GOTO=END
************** END OF EXCERPT FROM ROUTING TABLE DWSL1AI0 **************
TSTRI    DSLROUTE TYPE=TEST,COND=(CLASS,'RI',EQ),FALSE=TSTIP

⋮
```

To reset the AS status to **OPEN-NOHOLD**, enter the following commands on the MERVA Link control panels:

- **hold** *aspname*
- **aopen** *aspname*
- **astart** *aspname*

The ASP *aspname* is now started and resumes message transmission to the partner ASP.

FMT/ESA Scenario at the Message Receiving Side

Figure 135 shows how received SWIFT input messages are handled.

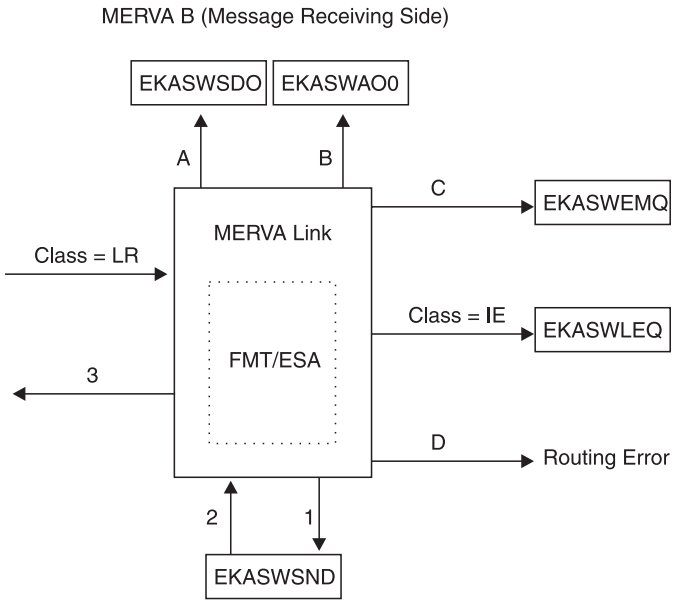

Figure 135. FMT/ESA Scenario at Message Receiving Side (MERVA B)

A received SWIFT input message is an inbound application message as seen by MERVA Link, and has message class 'LR'. FMT/ESA transforms the input message into a SWIFT output message. After successful transformation, a SWIFT acknowledgment or a delivery notification is provided and sent back if requested by the MERVA Link at the message sending side (SWIFT acknowledgment) or by the received SWIFT input message (delivery notification).

In routing table EKARTSIM, the following distribution logic is provided (the names enclosed in parentheses are the field names used in EKARTSIM):

A Either FMT/ESA or MERVA Link routes a SWIFT output message with class 'LR' to queue EKASWSDO if the following conditions are met:

- The MERVA Link delivery return code (DELRC) is equal to '00'.
- The SWIFT input/output message identifier (IOID) is equal to 'O'.
- The SWIFT output message was authenticated, and the result was either successful or indicated that an authentication was not required.

The MERVA Link Application Control Queue EKASIMCQ must be a target queue if MERVA Link routes the output message.

The control queue must not be a target queue if FMT/ESA routes the message.

B Either FMT/ESA or MERVA Link routes a SWIFT output message with class **LR** to queue EKASWAO0 if the following conditions are met:

- The MERVA Link delivery return code (DELRC) is equal to '00'.
- The SWIFT input/output message identifier (IOID) is equal to 'O'.
- The SWIFT output message was authenticated, and the result was unsuccessful.
- The SWIFT output message was not authenticated, as the customer did not request the authentication.

The MERVA Link Application Control Queue EKASIMCQ must be a target queue if MERVA Link routes the output message.

The control queue must not be a target queue if FMT/ESA routes the message.

C Either FMT/ESA or MERVA Link routes a SWIFT input message or a SWIFT output message including a delivery notification with class 'LR' to queue EKASWEMQ if the following condition is met:

- The MERVA Link delivery return code (DELRC) is greater than '00'.

The MERVA Link Application Control Queue EKASIMCQ must be a target queue if MERVA Link routes either the SWIFT input or the SWIFT output message.

The control queue must not be a target queue if FMT/ESA routes either the SWIFT input or the SWIFT output message.

Class = IE
FMT/ESA detected an error during processing of a received SWIFT input message and routes the message to queue EKASWLEQ assigning the new message class 'IE'.

D MERVA Link routes a SWIFT input message with class **LR** and forces a routing error if the following conditions are met:

- FMT/ESA detected an error during processing of a received SWIFT input message and routed the message to queue EKASWLEQ.
- The MERVA Link delivery return code (DELRC) is equal to '00'.
- The SWIFT input/output message identifier (IOID) is equal to 'I'.
- The MERVA Link request for ACK (ACKRQ) is either equal to '0' or equal to '1' indicating that you requested the SWIFT acknowledgment to be generated at the message sending side.

MERVA Link routes a delivery notification with class **LR** and forces a routing error if the following conditions are met:

- FMT/ESA detected an error during processing of a received SWIFT input message and routed the message to queue EKASWLEQ.
- The MERVA Link delivery return code (DELRC) is equal to '00'.
- The SWIFT input/output message identifier (IOID) is equal to 'O'.
- The message acknowledgment field (MSGACK) is available.

The routing error interrupts the message transmission. You find a specific error message in TOF field MSGACK, when you inspect the appropriate message in queue EKASWLEQ.

The next items show how a generated SWIFT acknowledgment or an error message, which is contained in TOF field MSGACK, or a delivery notification can be sent back to FMT/ESA at the message sending side. This applies only when you requested the acknowledgment to be generated at the message receiving side or when the received SWIFT input message requested a delivery notification.

1 MERVA Link routes a SWIFT input message containing field MSGACK to the MERVA Link send queue EKASWSND if the following conditions are met:

- The MERVA Link delivery return code (DELRC) is equal to '00'.
- The SWIFT input/output message identifier (IOID) is equal to 'I'.
- The MERVA Link request for ACK (ACKRQ) is equal to '2' indicating that the acknowledgment be generated at the message receiving side.

Either FMT/ESA or MERVA Link routes a delivery notification to the MERVA Link send queue EKASWSND if the following conditions are met:

- The MERVA Link delivery return code (DELRC) is equal to '00'.
- The SWIFT input/output message identifier (IOID) is equal to 'O'.
- The message acknowledgment field (MSGACK) is not available.
- The MERVA Link IAM message identifier field (IAMID) is available.

2 The message sending process is initiated. MERVA Link processes the message and passes it to FMT/ESA.

3 After processing of the message in FMT/ESA, MERVA Link sends:

- A status report to the MERVA Link at the message sending side. The status report contains the data from the MSGACK field.
- A delivery notification to the MERVA Link at the message sending side.

Customization

This section describes the tailoring facilities that support the integration of FMT/ESA in your MERVA ESA environment. Moreover, they enable you to request different ways of message processing by FMT/ESA. You can make your adjustments at three levels:

- FMT/ESA
- MERVA Link
- MERVA ESA

Customizing FMT/ESA with MERVA Link

Use the customization parameter table called EKASPRM to provide the following information to FMT/ESA:

- The name of the ISN control queue
- The name of the OSN control queue
- The identifier of a MERVA ESA journal record that is written for a sent message
- The identifier of a MERVA ESA journal record that is written for a received message
- The request to check SWIFT input messages before they are sent
- The request to authenticate SWIFT input and output messages

You use the macro EKASPARM to make your specifications. For details refer to the *MERVA for ESA Macro Reference*.

How to Activate the Customization Parameter Table: When you have modified one or more parameters in the customization parameter table, there are two ways to activate the parameter table. They depend on whether the FMT/ESA user exit EKAMU044 was link-edited to the MERVA ESA Message Format Services module DSLMMFS.

- EKAMU044 was *not link-edited to DSLMMFS*.

  This is how EKAMU044 was installed the first time in your system.

  You get the modified customization parameter table activated *without terminating MERVA ESA*. It is especially useful when you test FMT/ESA and you often make changes in the parameter table.

  You can perform the next steps when MERVA ESA is active:

  1. Assemble the customization parameter table and link-edit it to EKAMU044.
  2. For CICS only:

     Get the updated version of EKAMU044 using master terminal transaction CEMT:

     ```
     CEMT SET PROGRAM(EKAMU044) NEWCOPY
     ```

  As a result, DSLMMFS loads EKAMU044 with the modified EKASPRM link-edited to it when FMT/ESA is started the next time.

- EKAMU044 is to be *link-edited to DSLMMFS*.

  You make the following preparation once:

  In the appropriate entry of the MERVA ESA MFS program table DSLMPTT, you change parameter 'LINK' and specify:

  ```
  DSLMPT NAME=EKAMU044,NUMBER=7044,TYPE=U,LINK=YES
  ```

  You assemble DSLMPTT and link-edit it to DSLMMFS.

  Then you assemble the customization parameter table EKASPRM and link-edit it to DSLMMFS.

Customizing MERVA Link for Use with FMT/ESA

FMT/ESA requires at least one entry in the MERVA Link partner table EKAPT. You make all relevant specifications in an EKAPT TYPE=ASP statement. You can set up as many TYPE=ASP statements as you need for FMT/ESA. The following text shows the partner table parameters that are affected. Unless stated otherwise, specify the parameters for the MERVA Link at both the message sending side and the message receiving side.

Basic Customization for FMT/ESA

CONFIRM=NO
You request the SWIFT acknowledgment to be generated in the message sending MERVA ESA.

A SWIFT delivery notification is not generated.

The MERVA Link TOF field EKAACKRQ contains '0' when this parameter was specified. This field is checked in the FMT/ESA routing table EKARTSIM.

Specify this parameter for the MERVA Link of the sending MERVA ESA.

If this parameter is omitted, NO is used by default.

CONFIRM=NON
You request the SWIFT acknowledgment to be generated in the message sending MERVA ESA.

A SWIFT delivery notification is generated in the receiving MERVA ESA if it is requested by the received SWIFT input message.

The MERVA Link TOF field EKAACKRQ contains '1' when this parameter was specified. This field is checked in the FMT/ESA routing table EKARTSIM.

Specify this parameter for the MERVA Link of the sending MERVA ESA.

CONFIRM=ALL
You request the SWIFT acknowledgment to be generated in the message receiving MERVA ESA.

A SWIFT delivery notification is generated in the receiving MERVA ESA if it is requested by the received SWIFT input message.

The MERVA Link TOF field EKAACKRQ contains '2' when this parameter was specified. This field is checked in the FMT/ESA routing table EKARTSIM.

Specify this parameter for the MERVA Link of the sending MERVA ESA.

NAME=*asp name*
Specify the name of the ASP.

Each *asp name* that is involved in FMT/ESA for the sending of SWIFT input messages must be contained in the ISN Control Queue EKAISNCQ. If you want to initialize the ISN used for this ASP, you have to enter the name of the ASP and the ISN as a new control message in EKAISNCQ. Otherwise, FMT/ESA creates the new control message, if it is not yet stored in the queue.

SENDQC=EKASWSND
Specify the name of the MERVA Link send queue.

CONTROL=EKASIMCQ
Specify the name of the MERVA Link application control queue.

The routing table EKARTSIM must be associated with the control queue in the MERVA ESA Function Table.

MFSEXIT=7044
Specify the number of the MERVA ESA MFS User Exit program.

This identifies the FMT/ESA program.

The number and the name of the FMT/ESA program, EKAMU044, must also be specified in the MERVA ESA MFS program table.

IRROUTE=(ACK,**EKASWAWQ**,CTLQ)
Specify the name of the SWIFT acknowledgment wait queue. The parameter values ACK and CTLQ are also required.

Specify this parameter for the MERVA Link of the sending MERVA ESA.

FORMAT=...
Specify the format you prefer for the message transmission. You may use any of the supported format specifications. Messages can be transmitted either in SWIFT format or in MERVA ESA queue format.

Note: When you specify FORMAT=QUEUE, there is no external line format identifier supplied with this format specification. In this case FMT/ESA uses the external line format identifier W when it maps a SWIFT input or output message from the TOF to the network buffer. The mapping occurs when a message is to be authenticated and written to the MERVA ESA journal.

Customizing MERVA ESA for Use with FMT/ESA

The following parameters of the MERVA ESA customizing parameter table macro DSLPARM have an impact on the FMT/ESA processing:

NAME=*installation name*
Specify the name used to identify the MERVA ESA installation.

This name is used to compose the resource identifier of the OSN control queue. The resource identifier must be unique; it denotes the OSN control queue when FMT/ESA enqueues upon it during SWIFT output message processing. You have full control of the resource identifier. FMT/ESA concatenates the resource identifier RI based on your specifications, in this order:

RI = *installation name* || *node name* || *queue name*

Specify the resource identifier components using these macros:

- **DSLPARM NAME=***installation name*
  In MERVA ESA macro DSLPARM.
- **EKAPT TYPE=INITIAL,NODE=***node name*
  In MERVA Link partner table macro EKAPT.
- **EKASPARM OSNCQ=***queue name*
  In FMT/ESA customization parameter table macro EKASPARM.

NICBUF=*buffer size*
Specify the length of the data area used as a service data buffer in DSLNIC TYPE=REQ parameter BUF.

FMT/ESA allocates a buffer with the specified *buffer size* and uses it for data mapping purposes.

Global Customization versus Specific Customization

When you customize FMT/ESA as described in "Customizing FMT/ESA with MERVA Link" on page 285, the customization applies to all MERVA Link ASPs (connections to partners) where you want FMT/ESA to run. This is called *global customization*. However, you may want to use a different FMT/ESA profile depending on which partner is involved in the message exchange. For example:

- Some messages need to be authenticated only for certain partners, and not for others.
- You might want to use unique journal identifiers for sent and received messages on each MERVA Link ASP. This makes it easy for your journal analysis program to record information about the message traffic for each connection.

This is called *specific customization*. To apply a specific customization, do the following *for each ASP*:

1. Prepare a MERVA Link MFS user exit that calls the FMT/ESA user exit.

   The calling user exit contains the new customization parameters defined by the macro call:

   ```
   EKASPARM TYPE=INLINE,...
   ```

2. Provide an entry for the user exit in the MERVA ESA MFS program table DSLMPTT:

   ```
   DSLMPT NAME=nam,NUMBER=num,TYPE=U,LINK=NO
   ```

   Link-edit the MFS program table to DSLMMFS.

3. Provide an entry for the ASP (and the associated MTP) in the MERVA Link partner table EKAPT:

   ```
   EKAPT TYPE=ASP,...,MFSEXIT=num,...
   ```

   num represents the same user exit number as in the previous step in the DSLMPTT.

4. For CICS only:

   Provide an entry for the user exit in the CICS Program Definition.

Figure 136 on page 290 shows the sample user exit EKAMU045 which calls the FMT/ESA user exit EKAMU044, represented by the number '7044'.

Note: To test user exit EKAMU045:

1. Modify the parameter MFSEXIT in the partner table used for the installation verification. For more information about this partner table, refer to *MERVA for ESA Installation Guide*. Specify
   EKAPT TYPE=ASP,...,MFSEXIT=7045,...
2. Install the partner table.
3. Run FMT/ESA.

   The results should be the same as for the FMT/ESA user exit EKAMU044, except that queue EKASWAO0 now contains the generated SWIFT output message that was not authenticated.

The other modifications mentioned here are provided as part of the installation material.

```
         TITLE 'EKAMU045 - MERVA LINK USER EXIT 7045'
*********************************************************************
*        EKAMU045 MERVA LINK USER EXIT 7045
*********************************************************************
EKAMU045 EKAUXS NUM=7045,PRINT=OFF
         SPACE
*-------------------------------------------------------------------
*        FMT/ESA CUSTOMIZATION PARAMETER TABLE
*-------------------------------------------------------------------
         SPACE
         EKASPARM TYPE=INLINE,                                      *
               ISNCQ=EKAISNCQ,       NAME OF ISN CONTROL QUEUE      *
               OSNCQ=EKAOSNCQ,       NAME OF OSN CONTROL QUEUE      *
               JIDSENT=62,           MODIFIED: JOURNAL ID SENT  MSGS.  *
               JIDRCVD=63,           MODIFIED: JOURNAL ID RCVD. MSGS.  *
               AUTHENT=(NO,NO),      MODIFIED: AUTHENTICATION NOT REQ. *
               CHECK=YES             MESSAGE CHECKING REQUESTED
         SPACE
*-------------------------------------------------------------------
*        CALL FMT/ESA USER EXIT
*-------------------------------------------------------------------
         SPACE
         DSLMFS TYPE=USER,                                          *
               MODNUM=7044,          FMT/ESA USER EXIT NUMBER       *
               MSGID=UXMSGID,        MESSAGE IDENTIFIER             *
               MF=(E,MFSL)           MFS PARAMETER LIST
         SPACE
         MVC   MFSLREAS,MF$LREAS     PASS MFS REASON CODE TO CALLER
         B     MFSEXIT               RETURN TO MERVA ESA MFS
         EJECT
*-------------------------------------------------------------------
*        USER EXIT WORK FIELDS
*-------------------------------------------------------------------
         SPACE
MFSTS    DSECT                       MFS TEMP STORAGE (CONTINUED)
         DS    0D
UXMSGID  DS    CL8                   MESSAGE ID
UXFLDPF  DS    CL3                   TOF FIELD NAME PREFIX
MFSTTSLL EQU   *-MFSTS
         END
```

Figure 136. Sample MFS User Exit for ASP Specific Customization

When you create new user exits, you modify:

- The name of the exit ('EKAMU045')
- The number of the exit ('7045')
- The appropriate parameters in macro EKASPARM TYPE=INLINE,...

Macro EKASPARM must be specified before calling the FMT/ESA user exit (DSLMFS TYPE=USER,...).

The other code remains unchanged.

Interface to MERVA ESA MFS:

MFSLFLD This MFS parameter list field contains the address of the customization parameter table that will be passed to the FMT/ESA user exit EKAMU044.

Calling FMT/ESA from an MFS User Exit

It might be necessary to modify a message before or after FMT/ESA processes it. In the CICS environment, it is even possible to request CICS services by writing EXEC CICS commands. You can do all this in your own MERVA Link MFS user-exit routine, and then call FMT/ESA from that routine.

Figure 136 on page 290 shows a special case how a MERVA Link MFS user exit calls FMT/ESA.

Another example of an MFS user exit that calls FMT/ESA is shown below:

```
         TITLE 'EKAMU000 - MERVA LINK USER EXIT 7000'
*********************************************************************
*        EKAMU000 MERVA LINK MFS USER EXIT CALLING FMT/ESA
*********************************************************************
EKAMU000 EKAUXS NUM=7000
*/* SELECT USER EXIT FUNCTION:
*********************************************************************
*/* BEFORE READY-TO-SEND MESSAGE IS PROCESSED BY FMT/ESA:
*********************************************************************
MU00011W DS    0H
         CLI   CPLMUXF,@UXFRTS         READY-TO-SEND MESSAGE ?
         BNE   MU00012W                NO, CHECK FOR OTHER FUNCTION
*/* IF REQUESTED, INSERT YOUR CODE FOR READY-TO-SEND MESSAGE HERE.
         B     MU00010X                CALL FMT/ESA
*********************************************************************
*/* BEFORE OUTGOING MESSAGE IS PROCESSED BY FMT/ESA:
*********************************************************************
MU00012W DS    0H
         CLI   CPLMUXF,@UXFOBM         OUTBOUND MESSAGE ?
         BNE   MU00013W                NO, CHECK FOR OTHER FUNCTION
*/* IF REQUESTED, INSERT YOUR CODE FOR OUTGOING MESSAGE HERE.
         B     MU00010X                CALL FMT/ESA
*********************************************************************
*/* BEFORE CONFIRMED MESSAGE IS PROCESSED BY FMT/ESA:
*********************************************************************
MU00013W DS    0H
         CLI   CPLMUXF,@UXFCFM         CONFIRMED MESSAGE ?
         BNE   MU00014W                NO, CHECK FOR OTHER FUNCTION
*/* IF REQUESTED, INSERT YOUR CODE FOR CONFIRMED MESSAGE HERE.
         B     MU00010X                CALL FMT/ESA
*********************************************************************
*/* BEFORE INCOMING STATUS REPORT IS PROCESSED BY FMT/ESA:
*********************************************************************
MU00014W DS    0H
         CLI   CPLMUXF,@UXFIBR         INBOUND STATUS REPORT ?
         BNE   MU00015W                NO, CHECK FOR OTHER FUNCTION
*/* IF REQUESTED, INSERT YOUR CODE FOR ACKNOWLEDGED MESSAGE HERE.
         B     MU00010X                CALL FMT/ESA
*********************************************************************
*/* BEFORE INCOMING APPLICATION MESSAGE IS PROCESSED BY FMT/ESA:
*********************************************************************
MU00015W DS    0H
         CLI   CPLMUXF,@UXFIBM         INBOUND MESSAGE ?
         BNE   MU00016W                NO, CHECK FOR OTHER FUNCTION
*/* IF REQUESTED, INSERT YOUR CODE FOR INCOMING APPLICATION MESSAGE HERE.
         B     MU00010X                CALL FMT/ESA
*********************************************************************
*/* BEFORE RECOVERED OR RE-ROUTED MESSAGE IS PROCESSED BY FMT/ESA:
*********************************************************************
MU00016W DS    0H
         CLI   CPLMUXF,@UXFRCV         RECOVERED MESSAGE ?
         BNE   MU00017W                NO, CHECK FOR OTHER FUNCTION
*/* IF REQUESTED, INSERT YOUR CODE FOR RECOVERED OR RE-ROUTED MESSAGE HERE.
         B     MU00010X                CALL FMT/ESA
```

```
MU00017W DS    0H
*/* ENDSELECT
*---------------------------------------------------------------------
*        CALL FMT/ESA USER EXIT
*---------------------------------------------------------------------
MU00010X DS    0H
         DSLMFS TYPE=USER,                                             *
               MODNUM=7044,           FMT/ESA USER EXIT NUMBER        *
               MSGID=UXMSGID,         MESSAGE IDENTIFIER              *
               MF=(E,MFSL)            MFS PARAMETER LIST
         SPACE
         MVC   MFSLREAS,MF$LREAS      PASS MFS REASON CODE TO CALLER
         LTR   R15,R15                ANY ERROR OCCURRED ?
         BNZ   MFSEXIT                YES, RETURN TO MERVA ESA MFS
         SPACE
**********************************************************************
*/* AFTER READY-TO-SEND MESSAGE WAS PROCESSED BY FMT/ESA:
**********************************************************************
MU00021W DS    0H
         CLI   CPLMUXF,@UXFRTS        READY-TO-SEND MESSAGE ?
         BNE   MU00022W               NO, CHECK FOR OTHER FUNCTION
         OC    8(2,R4),8(R4)          MESSAGE CLASS SET BY FMT/ESA ?
         BNZ   MFSGOOD                YES, RETURN TO MERVA ESA MFS
*/* IF REQUESTED, INSERT YOUR CODE FOR READY-TO-SEND MESSAGE HERE.
         XC    8(2,R4),8(R4)          NO MESSAGE CLASS SET FOR ROUTING
         B     MFSGOOD                RETURN TO MERVA ESA MFS
**********************************************************************
*/* AFTER OUTGOING MESSAGE WAS PROCESSED BY FMT/ESA:
**********************************************************************
MU00022W DS    0H
         CLI   CPLMUXF,@UXFOBM        OUTBOUND MESSAGE ?
         BNE   MU00023W               NO, CHECK FOR OTHER FUNCTION
*/* IF REQUESTED, INSERT YOUR CODE FOR OUTGOING MESSAGE HERE.
         B     MFSGOOD                RETURN TO MERVA ESA MFS
**********************************************************************
*/* AFTER CONFIRMED MESSAGE WAS PROCESSED BY FMT/ESA:
**********************************************************************
MU00023W DS    0H
         CLI   CPLMUXF,@UXFCFM        CONFIRMED MESSAGE ?
         BNE   MU00024W               NO, CHECK FOR OTHER FUNCTION
*/* IF REQUESTED, INSERT YOUR CODE FOR CONFIRMED MESSAGE HERE.
         B     MFSGOOD                RETURN TO MERVA ESA MFS
**********************************************************************
*/* AFTER INCOMING STATUS REPORT WAS PROCESSED BY FMT/ESA:
**********************************************************************
MU00024W DS    0H
         CLI   CPLMUXF,@UXFIBR        INBOUND STATUS REPORT ?
         BNE   MU00025W               NO, CHECK FOR OTHER FUNCTION
*/* IF REQUESTED, INSERT YOUR CODE FOR ACKNOWLEDGED MESSAGE HERE.
         B     MFSGOOD                RETURN TO MERVA ESA MFS
**********************************************************************
*/* AFTER INCOMING APPLICATION MESSAGE WAS PROCESSED BY FMT/ESA:
**********************************************************************
MU00025W DS    0H
         CLI   CPLMUXF,@UXFIBM        INBOUND MESSAGE ?
         BNE   MU00026W               NO, CHECK FOR OTHER FUNCTION
*/* IF REQUESTED, INSERT YOUR CODE FOR INCOMING APPLICATION MESSAGE HERE.
         B     MFSGOOD                RETURN TO MERVA ESA MFS
**********************************************************************
*/* AFTER RECOVERED OR RE-ROUTED MESSAGE WAS PROCESSED BY FMT/ESA:
**********************************************************************
MU00026W DS    0H
         CLI   CPLMUXF,@UXFRCV        RECOVERED MESSAGE ?
         BNE   MU00027W               NO, CHECK FOR OTHER FUNCTION
*/* IF REQUESTED, INSERT YOUR CODE FOR RECOVERED OR RE-ROUTED MESSAGE HERE.
         B     MFSGOOD                RETURN TO MERVA ESA MFS
         SPACE
```

```
MU00027W DS    0H
         B     MFSGOOD                 RETURN TO MERVA ESA MFS
         SPACE
         LTORG
*------------------------------------------------------------------------
*        USER EXIT WORK FIELDS
*------------------------------------------------------------------------
MFSTS    DSECT                         MFS TEMP STORAGE (CONTINUED)
         DS    0D
UXMSGID  DS    CL8                     MESSAGE ID
UXFLDNM  DS    0CL8                    TOF FIELD NAME
UXFLDPF  DS    CL3                     TOF FIELD NAME PREFIX
UXFLDID  DS    CL5                     TOF FIELD NAME IDENTIFIER
MFSTTSLL EQU   *-MFSTS
         END
```

The code shown for a ready-to-send message after it was processed by FMT/ESA is only an example. Alternatively, you could save the message class set by FMT/ESA, insert your code, then restore the saved message class, as shown below. Note that the code is also valid if FMT/ESA did not set a message class.

```
⋮

************************************************************************
*/* AFTER READY-TO-SEND MESSAGE WAS PROCESSED BY FMT/ESA:
************************************************************************
MU00021W DS    0H
         CLI   CPLMUXF,@UXFRTS         READY-TO-SEND MESSAGE ?
         BNE   MU00022W                NO, CHECK FOR OTHER FUNCTION
         MVC   UXMSGCLS,8(R4)          SAVE MESSAGE CLASS SET BY FMT/ESA
*/* IF REQUESTED, INSERT YOUR CODE FOR READY-TO-SEND MESSAGE HERE.
         MVC   8(2,R4),UXMSGCLS        RESTORE SAVED MESSAGE CLASS
         B     MFSGOOD                 RETURN TO MERVA ESA MFS

⋮

*------------------------------------------------------------------------
*        USER EXIT WORK FIELDS
*------------------------------------------------------------------------
MFSTS    DSECT                         MFS TEMP STORAGE (CONTINUED)

⋮

UXMSGCLS DS    CL2                     MESSAGE CLASS SET BY FMT/ESA
MFSTTSLL EQU   *-MFSTS

⋮
```

The routine shown above (EKAMU000) is a valid MERVA Link MFS user exit, and is contained in the MERVA ESA installation material. If you use it, provide code only for the section where you want to influence the processing of FMT/ESA. Coding examples are contained in the source code of the user-exit routines EKAMU010 or EKAMU133.

The logical places where MERVA Link calls an MFS user exit are:

- For a ready-to-send message
- For an outgoing message
- For a confirmed message
- For an incoming status report
- For an incoming application message

- For a recovered or rerouted message

These places are described in "Support of the MFS User Exits" on page 203 and, in more detail, in *MERVA for ESA Advanced MERVA Link*. "CICS Commands in an MFS User Exit" on page 207 describes how to issue CICS commands in the user exit.

To test EKAMU000:

1. Modify parameter MFSEXIT in the partner table used for the installation verification. For more information about this table, refer to the *MERVA for ESA Installation Guide*. Specify **EKAPT TYPE=ASP,...,MFSEXIT=7000,...**
2. Install the partner table.
3. Run FMT/ESA.

The following table shows the results when FMT/ESA finishes processing at the end of each logical place and returns to its caller. It provides you with information for coding your own user exit. The table gives you another view of the message flow explained in Figure 132 on page 268 and Figure 133 on page 271.

Table 10. FMT/ESA Processing Results for Each Logical Place

Logical Place	Processing Result
Ready-to-Send Message	For a SWIFT input message without the MSGACK field: • Session number '0000' inserted • ISN incremented and inserted • Message checked • Message authenticated • Message written to the MERVA ESA journal • ISN written to the ISN control queue For a SWIFT input message containing the MSGACK field: • SWIFT output message and delivery notification message routed from the ISN control queue to the target queue(s) For a delivery notification message: • SWIFT output message routed from the ISN control queue to the target queue(s) Error indication: • Register 15 contains return code greater than 0. • Register 15 contains 0 and the first two bytes of the TOF field working buffer addressed by register 4 with offset 8 containes the characters 'SE' or 'LR'. MERVA Link interprets 'SE' or 'LR' as message class. It routes the message to a target queue depending on the message class. The target queue for the message with class 'LR' does not exist. However, this message was routed before to an error queue with class 'IE'.
Outgoing Message	For a SWIFT input message containing the MSGACK field: • MERVA Link receipt return code field EKARECRC set to '00'. This indicates a MERVA Link status report. • MERVA Link receipt diagnostic code field EKARECDC set to the first six characters of the MSGACK field. • MSGACK field and SWIFT trailers contained in field SWTRAIL written to data areas of field EKARDATA. For a SWIFT input message without the MSGACK field and containing the EKAPDUPM field with 'PDM' (indicates that message was recovered): • PDE trailer generated by the SWIFT Link inserted, if available.

Table 10. FMT/ESA Processing Results for Each Logical Place (continued)

Logical Place	Processing Result
Confirmed Message	For a SWIFT input message containing the EKAPDUPM field with 'PDM' (indicates that message was recovered): • Message used for PDE trailer processing with the SWIFT Link deleted from the ISN control queue. • Message containing a PDE trailer written to the MERVA ESA journal. For CONFIRM=NO or CONFIRM=NON specified in the MERVA Link partner table EKAPT: • SWIFT acknowledgment written to the MSGACK field • SWIFT acknowledgment written to the MERVA ESA journal Error indication: • Register 15 contains return code greater than 0 • Register 15 contains 0 and both the following are true: – An error message starting with **EKA8** is written to the MSGACK field – Field EKARECRC is set to '00'
Incoming Status Report	• Data areas of field EKARDATA written to fields MSGACK and SWTRAIL • SWIFT acknowledgment written to the MERVA ESA journal • Data areas of field EKARDATA deleted (except for the MERVA Link control message type MCTL) Error indication: • Register 15 contains return code greater than 0 • Register 15 contains 0 and both the following are true: – An error message starting with 'EKA8' is written to the MSGACK field – Field EKARECDC is deleted

Table 10. FMT/ESA Processing Results for Each Logical Place (continued)

Logical Place	Processing Result
Incoming Application Message	For a SWIFT input message: • Input message transformed to a SWIFT output message • OSN incremented and inserted • Output message authenticated • Output message written to the MERVA ESA journal • SWIFT acknowledgment written to the MERVA ESA journal • OSN written to the OSN control queue • If field EKAACKRQ contains '1' and if a delivery notification message is not to be created, field EKAACKRQ set to '0' • If field EKAACKRQ contains '2' or if a delivery notification message is to be created, output message written to the ISN control queue • If field SWAHIDM of the input message contains '2' or '3' and if field EKAACKRQ contains '1' or '2', delivery notification message created • If field EKAACKRQ contains '2', delivery notification message written to the ISN control queue • If field EKAACKRQ contains '2', SWIFT acknowledgment written to the MSGACK field For a delivery notification message: • OSN incremented and inserted • Message written to the MERVA ESA journal • SWIFT acknowledgment written to the MERVA ESA journal • OSN written to the OSN control queue • Field EKAAMSID deleted Error indication: • Register 15 contains return code greater than 0 • Register 15 contains 0 and Error message starting with **EKA8** written to the MSGACK field
Recovered Message	For a SWIFT input message containing field EKAPDUPM with 'PDM': • If field EKACLASS contains 'RC', message without SWIFT PDE trailer written to the ISN control queue • If field EKACLASS contains 'RC', name of the ISN control queue written to the field EKAAMBSL • PDE trailer written to the field SWTRAIL For a SWIFT input message not containing field EKAPDUPM with 'PDM' or not containing field EKACLASS with 'RC': • Message without SWIFT PDE trailer deleted from the ISN control queue

Using FMT/ESA with MERVA-MQI Attachment

The FMT/ESA message flow when using MERVA-MQI Attachment is similar to that described in “FMT/ESA Message Flow with MERVA Link” on page 266, except that MERVA-MQI Attachment takes the place of MERVA Link. FMT/ESA with MERVA-MQI Attachment counts each sent and received message in a message counter data set. This is the same data set that is used by FMT/ESA with MERVA Link, and uses the same counter indices (hexadecimal 60 and 61).

Customizing MERVA-MQI Attachment for Use with FMT/ESA

Use the MERVA-MQI Attachment process table DSLKPROC to specify the following for FMT/ESA:

- The names of the queues in which input sequence numbers (ISNs) and output sequence numbers (OSNs) are to be stored (**ISNCTLQ=isn_control_queue** and **OSNCTLQ=osn_control_queue**). These queues are described in "Queues for FMT/ESA with MERVA-MQI Attachment".
- The name of the exit used to call FMT/ESA. The EXIT parameter is mandatory and must be set to **EXIT=8044**.
- Where to generate a SWIFT acknowledgment and a delivery notification (MT 011):

  COAWQ=coa_wait_queue
  A SWIFT ACK is generated locally (by the sending MERVA ESA). No delivery notification is generated. No COD report is requested from MQSeries.

  COAWQ=coa_wait_queue *and* **CODWQ=#**
  A SWIFT ACK is generated locally (by the sending MERVA ESA). If a delivery notification is requested by the received SWIFT input message, it is generated by the receiving MERVA ESA. No COD report is requested from MQSeries.

  COAWQ=coa_wait_queue *and* **CODWQ=cod_wait_queue**
  A SWIFT ACK is generated locally (by the sending MERVA ESA). If a delivery notification is requested by the received SWIFT input message, it is generated by the receiving MERVA ESA. In addition to the delivery notification, a COD report is requested from MQSeries.

  ACKWQ=ack_wait_queue
  A SWIFT ACK is generated remotely (by the receiving MERVA ESA). If a delivery notification is requested by the received SWIFT input message, it is generated by the receiving MERVA ESA. If the COAWQ and CODWQ parameters are specified additionally, then a COA report, a COD report, or both are requested from MQSeries.

  There is a direct relationship between these parameters and the CONFIRM parameter of the MERVA Link partner table EKAPT (described in "Customizing MERVA Link for Use with FMT/ESA" on page 286):
  - COAWQ corresponds to CONFIRM=NO
  - COAWQ and CODWQ correspond to CONFIRM=NON
  - ACKWQ corresponds to CONFIRM=ALL
- Whether to authenticating SWIFT input and output messages (**AUTHENT=YES** or **NO**).
- Whether to use the Message Format Service (MFS) to check SWIFT input and output messages (**CHECK=YES** or **NO**).
- The IDs of the MERVA ESA journal records in which the sent and received SWIFT messages are to be stored (set by the **JIDSENT** and **JIDRCVD** parameters).

Queues for FMT/ESA with MERVA-MQI Attachment

The same queues that were used to verify the installation of MERVA-MQI Attachment can be used for FMT/ESA. These queues are described in the *MERVA*

for ESA Installation Guide. In addition to these, you will need one or two additional queues for each MERVA-MQI Attachment send and receive process that is part of FMT/ESA:

- An input sequence number (ISN) control queue, the name of which is specified by the **ISNCTLQ** parameter. This parameter is mandatory for a send process (DSLKPROC TYPE=SEND) and receive process (DSLKPROC TYPE=RECEIVE).
- An output sequence number (OSN) control queue, the name of which is specified by the **OSNCTLQ** parameter. This parameter is mandatory for a receive process (DSLKPROC TYPE=RECEIVE).

The ISN control queue must be defined with two keys in the MERVA ESA function table DSLFNTT:

- KEY1 represents the name of the send process contained in the TOF field DSLKPNM. Specify KEY1=(DSLKPNM,8).
- KEY2 represents the contents of the field MsgId from the MQI control block MQMD contained in the TOF field DSLKMSID. Specify KEY2=(DSLKMSID,24,,NOMOD).

When MERVA-MQI Attachment processes a SWIFT input message for a specific send process, it passes the message to FMT/ESA, which increments the current ISN and inserts the new ISN into the input message. Similarly, when MERVA-MQI Attachment processes a SWIFT output message or delivery notification, FMT/ESA increments the current OSN and inserts the new OSN into the message.

Routing

The sample routing table DSLKQRT affects the message routing in the following ways:

- The value specified in the TOF field DSLKAKRQ at the sending side affects how FMT/ESA handles delivery notifications and SWIFT acknowledgments:

  0 No delivery notification or SWIFT acknowledgment is generated by the FMT/ESA at the receiving side.

  1 If requested by the SWIFT input message at the sending side, a delivery notification is generated by the FMT/ESA at the receiving side. The DSLKAKRQ field of the notification is assigned the value 1, and the delivery notification is routed to a MERVA send queue of the MERVA-MQI Attachment at the receiving side, sent to the sending side, assigned the value 0, and routed to a MERVA receive queue there.

  2 Both a SWIFT acknowledgment and (if requested by the SWIFT input message at the sending side) a delivery notification are generated by the FMT/ESA at the receiving side. The DSLKAKRQ field of the notification is assigned the value 1; the DSLKAKRQ field of the acknowledged SWIFT input message is assigned the value 2. Both the notification and the acknowledged SWIFT input message are routed to a MERVA send queue of the MERVA-MQI Attachment at the receiving side. The notification is sent to the sending side, assigned the value 0, and routed to a MERVA receive queue there. The acknowledgment is sent as an MQI reply message to the sending side, and correlated with the waiting SWIFT input message in the acknowledgment wait queue specified in parameter ACKWQ. The correlated SWIFT input message is routed to a MERVA receive queue at the sending side.

  Note: These values are also placed in the MQI control block MQMD, in the next to the last byte of the field ApplIdentityData. When a message is

displayed, the MERVA ESA command SHOW KCOV displays each field of the MQMD and shows the contents of ApplIdentityData.

- The message statuses contained in TOF field DSLKSTAT, each of which corresponds to an error situation:

  ACKER An error occurred in a MERVA-MQI Attachment receive process after successful correlation with the received reply message.

  COAER An error occurred in a MERVA-MQI Attachment receive process after successful correlation with the received COA report.

  ERSND An error occurred in a MERVA-MQI Attachment send process. The message cannot be transmitted to the receiving MERVA ESA.

  ERSWO An error occurred in a MERVA-MQI Attachment send process. The generated SWIFT output message cannot be routed to its target queue.

  SWIER An error occurred in a MERVA-MQI Attachment receive process concerning a SWIFT input message.

  A message with any of these statuses is routed to an error queue.

The following excerpt from the routing table DSLKQRT shows how messages sent via FMT/ESA can be handled at the receiving side (the message status in field DSLKSTAT is RCVD):

```
RCVD     DSLROUTE TYPE=DEFINE,FIELD=(ACKREQ,DSLKAKRQ,,,,,VFIRST),      *
               FOUND=FMTRCVD                                  @LEI0004   [1]
         DSLROUTE TYPE=TEST,COND=(MSGTYP,'8',EQ,SHORT),TRUE=RCVDQ
         DSLROUTE TYPE=SET,TARGET='DSLMRSQ2'
RCVDQ    DSLROUTE TYPE=SET,TARGET='DSLMRRQ1',GOTO=END
FMTRCVD  DSLROUTE TYPE=TEST,COND=(ACKREQ,'0',EQ),TRUE=FMTRCVDQ @LEI0004   [2]
         DSLROUTE TYPE=SET,TARGET='DSLMRSQ2',GOTO=END          @LEI0004   [3]
FMTRCVDQ DSLROUTE TYPE=SET,TARGET='DSLMRRQ1',GOTO=END          @LEI0004   [4]
```

Notes:

[1] The field DSLKAKRQ is defined as ACKREQ. If the field is found in the TOF, processing continues at the FMTRCVD label.

[2] If ACKREQ contains the value 0, the message is a delivery notification that is to be routed to a receive queue. Processing continues at the FMTRCVDQ label.

[3] If ACKREQ does not contain the value 0 (in which case it contains the value 1 or 2, where 1 represents a delivery notification, and 2 an acknowledged SWIFT input message), the message is routed to the send queue DSLMRSQ2, and processing stops at the END label.

[4] The delivery notification from step 2 is routed to the target queue DSLMRRQ1, and processing stops at the END label.

Chapter 9. MERVA-MQI Attachment

MERVA-MQI Attachment provides a means of communication between MERVA ESA and MQSeries for MVS/ESA and VSE/ESA. This chapter describes the facilities available to adapt the functions of MERVA-MQI Attachment to the requirements of the message transfer in a specific installation.

The *MERVA for ESA Concepts and Components* book describes the basic principles of MERVA-MQI Attachment. It is recommended that you read the appropriate section in this book.

Customizing the Send and Receive Processes

You use the macro DSLKPROC to define one or more send and receive processes (see the *MERVA for ESA Macro Reference* for details). These definitions are collected in the process table DSLKPROC. The name DSLKPROC of the process table cannot be changed.

A send process definition contains the characteristics of the message transfer from MERVA ESA to the MQSeries. It is therefore called MERVA-to-MQI send process. Each send process is identified by a unique name. A receive process definition contains the characteristics of the message transfer from the MQSeries to MERVA ESA. It is therefore called MQI-to-MERVA receive process. Each receive process is identified by a unique name. More than one send or receive processes can be active at a time.

In the following the effect of specified DSLKPROC parameters on the message exchange between MERVA ESA and MQSeries is explained.

Setting the MQI Message Types

MERVA-MQI Attachment sends three message types defined by the MQSeries:

- Datagram
- Request message
- Reply message

A datagram is sent if the ACKWQ parameter of DSLKPROC TYPE=SEND is omitted.

A request message is sent if the name of a MERVA ESA acknowledgment wait queue is specified in the ACKWQ parameter. The acknowledgment is a reply message.

A reply message is constructed using the control information contained in the appropriate request message. Therefore a reply message can only be sent if MERVA-MQI Attachment finds a received request message in a send queue. This requires that the customer application routes a received request message to a suitable send queue of MERVA-MQI Attachment.

MERVA-MQI Attachment sets the message type in the field *MsgType* of the MQI control block MQMD. The type of a received message can always be identified using the *MsgType* field. In addition to the three MQI message types, MERVA-MQI Attachment also receives report messages as the fourth MQI message type.

Defining the Message Data Structure

The structure of a datagram, request, or reply message that is to be sent can be influenced by a user exit. The user exit is specified in the EXIT parameter of DSLKPROC TYPE=SEND (see "Writing a User Exit" on page 332 for details on user exits called by MERVA-MQI Attachment).

- The contents of a datagram or request message built without a user exit is shown below.

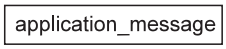

Figure 137. Datagram or Request Message without Additional Fields

The application message can be a SWIFT, telex, or user-defined message.

The data provided by a user exit changes the contents of a datagram or request message as follows:

Figure 138. Datagram or Request Message Containing Additional Fields

The data parts within a datagram or request message are now each preceded by a 4-byte field LLLL. This field contains the length, in bytes, of the following data part including its own length (4 bytes).

The data parts following the application message represent the additional fields provided by the user exit. A field consists of the field ID and the field data. The fixed length field ID of 8 bytes contains the name of the additional field. The variable length field data of up to 28672 bytes contains the data of the additional field.

- A reply message built without a user exit contains no data and so has a length of zero. This is a valid form of a reply message.
- The layout of a reply message consisting of one field provided by a user exit is shown below.

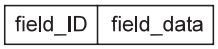

Figure 139. Reply Message Containing Data in One Field

The meaning of the field ID and field data is the same as for the additional fields in a datagram or request message. There is no length field required for data in one field.

- The contents of a reply message consisting of more than one field provided by a user exit is shown below.

Figure 140. Reply Message Containing Data in More Than One Field

The data parts within such a reply message are each preceded by a 4-byte field LLLL. The meanings of the terms LLLL, field ID, and field data are the same as for the additional fields in a datagram or request message.

Recognizing the Message Data Structure

The application receiving a datagram, request, or reply message must be able to recognize the nature of the data in the message. This information is provided by the MQFMT parameter of DSLKPROC TYPE=SEND. MQFMT specifies the name of an MQI data format. MERVA-MQI Attachment uses the format name as an indicator for the structure of the message data:

- Names beginning with ''MQ'' indicate that the message does **not** contain a length field LLLL to separate its data parts.

  These names represent MQI built-in formats.
- Names beginning with other than ''MQ'' indicate that the message contains a length field LLLL.

  These names represent application-defined formats.

When the MQFMT parameter is omitted, MERVA-MQI Attachment uses the following default format names:

- MQSTR if the message does not contain length fields or there is no message data at all (reply message only).

  This is the value of the built-in format MQFMT_STRING.
- Blanks if the message contains length fields.

When the specified format name and the message data structure do not match, MERVA-MQI Attachment ignores the format name and sets a new name. This can occur due to one of the following reasons:

- The datagram or request message to be sent has no additional fields as the user exit was not specified in the EXIT parameter, or the user exit did not provide any data. The message structure is according to Figure 137 on page 302.

  The name of an application-defined format was specified.

  Result: MERVA-MQI Attachment sets the format name MQSTR.
- The reply message to be sent has no data as the user exit was not specified in the EXIT parameter, or the user exit did not provide any data.

  Any format name other than MQSTR was specified.

  Result: MERVA-MQI Attachment sets the format name MQSTR.
- The reply message to be sent has data without a length field. The message structure is as shown in Figure 139 on page 302.

  The name of an application-defined format was specified.

  Result: MERVA-MQI Attachment sets the format name MQSTR.
- The datagram, request, or reply message to be sent consists of data parts preceded by a length field. The message structure is as shown in Figure 138 on page 302 or Figure 140 on page 302.

  The name of an MQI built-in format was specified, for example, MQSTR.

  Result: MERVA-MQI Attachment sets the format name to blanks or uses the alternate format name if an alternate format name for MQSTR is specified in the MQFMT parameter.

MERVA-MQI Attachment sets the data format name in the field *Format* of the MQI control block MQMD. The format of a received message can always be identified using the *Format* field. Thus the receiving application gets the information whether it has to consider the length field LLLL for the processing of the received message.

Defining the Groups of MERVA ESA Messages

MERVA-MQI Attachment sends three groups of MERVA ESA messages as a datagram or request message:

- SWIFT
- Telex
- User-defined messages

The format code in the FORMAT parameter of DSLKPROC TYPE=SEND identifies each group. The format code applies when the message has to be sent in the external line format defined in an MCB. The format code is used in the MCB to format the message.

The following format codes must be defined in the appropriate MCBs to indicate the groups:

W SWIFT message

P Telex message

U User-defined message

When MERVA-MQI Attachment sends messages, it uses the field *ApplIdentityData* of the MQI control block MQMD to transmit control information. It sets the first character of *ApplIdentityData* to one of the following:

- If using external line format, the format code
- If using MERVA ESA queue format, the letter **Q**

The receiving application can check *ApplIdentityData* and determine the group of the MERVA ESA message contained in the datagram or request message.

When MERVA-MQI Attachment receives messages, it checks the first character of *ApplIdentityData*. If the value is:

W, P, or U It uses the corresponding format code.

Q It uses the MERVA ESA queue format.

anything else It uses the specification made in the FORMAT parameter of DSLKPROC TYPE=RECEIVE.

When MERVA-MQI Attachment receives messages in external line format, if the FORMAT parameter of DSLKPROC TYPE=RECEIVE specifies a blank message type, MERVA-MQI Attachment uses the MERVA ESA MFS user exit DSLMU054 to determine the message type. For user-defined messages (format code U), the appropriate program selection must be coded in DSLMU054.

For sending and receiving telex messages, the user exit DSLKQ100 is required. DSLKQ100 is defined to MERVA-MQI Attachment when EXIT=8100 is specified for the appropriate send and receive process in the DSLKPROC macro.

Setting the MQI Report Options

When MERVA-MQI Attachment sends a datagram, request message, or reply message, it can specify that it expects one or more of the following report messages:

- COA report
- COD report
- Exception report

A COA report is requested if the name of a MERVA ESA COA wait queue is specified in the COAWQ parameter of DSLKPROC TYPE=SEND.

A COD report is requested if the name of a MERVA ESA COD wait queue is specified in the CODWQ parameter of DSLKPROC TYPE=SEND.

An exception report is requested if EXCEPT=YES is specified in the EXCEPT parameter of DSLKPROC TYPE=SEND.

MERVA-MQI Attachment sets the report options in the field *Report* of the MQI control block MQMD. Each option specifies that up to 100 characters from the beginning of the original request message are included in the report message.

When reports are requested for a reply message to be sent, MERVA-MQI Attachment uses the characters 1 to 16 of the MQMD control block field *ApplIdentityData*. The characters 17 to 32 of *ApplIdentityData* are available for the application that sends the appropriate request message.

Note: If MERVA-MQI Attachment is running under MVS and sends messages to MQSeries for VSE/ESA, or if it is running under VSE, an exception report cannot be requested.

Authorizing the Use of Queues

MERVA-MQI Attachment sends and receives messages using both MERVA ESA and MQI queues. A group of users must be authorized to work with these queues.

Defining an Alternate User Identifier

Note: An alternate user identifier can only be defined if MERVA-MQI Attachment is running under MVS. MQSeries for VSE/ESA does not support an alternate user identifier.

The authorization for the MQI send and receive queues is not restricted to the user identifier that MERVA-MQI Attachment is currently running under. If nothing else is specified, the queue manager uses the following identifiers:

- For CICS, the user ID associated with the task
- For IMS MPP regions, one of:
  - The signed-on user ID associated with the message
  - The logical terminal (LTERM) name
  - The user ID from the region JES JOB card
  - The TSO user ID
  - The PSB name

These identifiers are called current identifiers.

The ALTUID parameter of DSLKPROC specifies an alternate user identifier. The first 8 characters of the alternate user ID are used to check the authorization for the open of the MQI send or receive queues. The current user ID must be authorized in MQSeries to specify the particular alternate user ID. All 12 characters of the alternate user ID are used for this check.

For a datagram or request message to be sent, MERVA-MQI Attachment sets the alternate user ID in the field *UserIdentifier* of the MQI control block MQMD.

Defining an Authorized MERVA-MQI Attachment User

A MERVA ESA operator can start one or more send and receive processes using the command SF. This is an example for a command that an operator can issue against a MERVA ESA function associated with a transaction. Other commands of

this kind are CF and HF. MERVA-MQI Attachment uses MERVA ESA send and start queues which are associated with a transaction. Therefore the unauthorized commands CF, HF, and SF should become authorized before they can be issued against MERVA ESA queues used by MERVA-MQI Attachment.

The authorization requires definitions in two MERVA ESA resources, in:

- The user file
- The function table DSLFNTT

In the MERVA ESA user file, a user must be defined either as a master user (user type 'M') or as a MERVA-MQI Attachment user (user type 'K'). In the MERVA ESA function table, the parameter MQI=YES must be set for those queues with an associated transaction which are to be used as send and start queues for MERVA-MQI Attachment.

When a MERVA ESA operator is neither a master user nor a MERVA-MQI Attachment user and issues one of the commands CF, HF, or SF against a queue defined with MQI=YES, the command is rejected.

Defining the Send Queues

Sending a message from MERVA ESA to the MQSeries means that MERVA-MQI Attachment retrieves the MERVA ESA message from a MERVA ESA send queue and puts it as an MQI message to an MQI send queue. The association between a MERVA ESA and an MQI send queue can be specified in the ALLSNDQ parameter of DSLKPROC TYPE=SEND.

Up to 10 send queue pairs per send process can be defined. This allows to order the send queue pairs according to the priority scheme of MQSeries where 0 is the lowest priority and 9 is the highest priority. Several MERVA ESA send queues can be associated to the same MQI send queue.

A MERVA ESA send queue is defined in the MERVA ESA function table DSLFNTT. An MQI send queue is defined in MQSeries either as a local queue or as the local definition of a remote queue:

- When it is defined as a local queue, the following attributes must be specified:
  - For MQSeries for MVS/ESA, use the DEFINE QLOCAL command:
    ```
    DEFINE QLOCAL PUT(ENABLED) NOTRIGGER ...
    ```
  - For MQSeries for VSE/ESA:
    - On the LOCAL QUEUE DEFINITION screen: Put Enabled: Y
    - On the Queue Extended Definition screen: Trigger Enable: N
- When it is defined as the local definition of a remote queue, the following attributes must be specified:
  - For MQSeries for MVS/ESA, use the DEFINE QREMOTE command:
    ```
    DEFINE QREMOTE PUT(ENABLED) XMITQ(...) ...
    ```
  - For MQSeries for VSE/ESA, on the REMOTE QUEUE DEFINITION screen:
    - Put Enabled: Y
    - TRANSMISSION Q NAME: ...

For details see the *MQSeries Command Reference* or the *MQSeries for VSE/ESA System Management Guide*.

A MERVA-to-MQI send process is started automatically by MERVA ESA or by an operator using the command SF. The send process is started automatically:

- When a message is written to a MERVA ESA send queue which is in status NOHOLD
- During MERVA ESA startup when a send queue is in status AUTO

If you want to start the send process via the SF command, there are two ways to collect MERVA ESA messages in several MERVA ESA send queues before starting the send process:

- Define the MERVA ESA send queues in the MERVA ESA function table DSLFNTT using the parameters TRAN=DSLS and STATUS=HOLD. This prevents the transaction DSLS of the MERVA-to-MQI send process program DSLKQS from being started automatically. The held function can be released using the SF command.
- Define the MERVA ESA send queues in the DSLFNTT and omit the parameter TRAN=DSLS. This inhibits an automatic start of the DSLKQS program.

Refer to "Defining the Start Queue" on page 310 for more information on the operator-controlled start of a send process.

Defining the Receive Queues

Receiving a message from the MQSeries means that MERVA-MQI Attachment retrieves the MQI message from an MQI receive queue and puts it as a MERVA ESA message to a MERVA ESA control queue. The MQIRCVQ parameter of DSLKPROC TYPE=RECEIVE contains the names of the MQI receive queues.

You can define up to 10 receive queues per receive process. As with the send queue pairs, this lets you order the receive queues according to the priority scheme of MQSeries. In MQSeries, an MQI receive queue is always defined as a local queue. In MQSeries for MVS/ESA, the attribute GET(ENABLED) must be specified in the DEFINE QLOCAL command. In MQSeries for VSE/ESA, specify **Get Enabled: Y** on the LOCAL QUEUE DEFINITION screen.

An MQI-to-MERVA receive process is started in one of the following ways:

Automatically by the MQSeries queue manager

The automatic start by the queue manager is called *triggering*. The receive process is triggered for an MQI receive queue when both the following are true:

- Triggering is enabled for that queue.
- Certain predetermined conditions are satisfied; for example, a message with a priority greater than or equal to the message priority trigger number specified for that queue arrives in the queue.

To enable triggering, the following attributes must be specified:

- For MQSeries for MVS/ESA, use the DEFINE QLOCAL command:
  ```
  DEFINE QLOCAL INITQ(...) PROCESS(...) TRIGGER TRIGTYPE(FIRST) ...
  ```
- For MQSeries for VSE/ESA, on the Queue Extended Definition screen:
  - Trigger Enable: Y
  - Trigger Type: F
  - Allow Restart of Trigger: Y
  - Trans ID: ...

By MERVA ESA

MERVA ESA can start a receive process during startup. This occurs when a MERVA ESA start queue is assigned to the receive process and the start

queue is in status AUTO. The start queue can be set to status AUTO by the MERVA ESA function table definition DSLFNT STATUS=AUTO.

By an operator using the command SF

If you want to start the receive process via the SF command, you can collect MQI messages in several MQI receive queues before starting the receive process. Switch off triggering for these receive queues as follows:

- For MQSeries for MVS/ESA, use the DEFINE QLOCAL command:
  `DEFINE QLOCAL NOTRIGGER ...`
- For MQSeries for VSE/ESA, on the Queue Extended Definition screen:
  - Trigger Enable: N

If required, set the start queue to status NOHOLD by specifying DSLFNT STATUS=NOHOLD. Refer to "Defining the Start Queue" on page 310 for more information on the operator-controlled start of a receive process.

Defining the Reply-to Queue

The reply-to queue is an MQI receive queue that must be defined in a send process. In the REPLYTQ parameter of DSLKPROC TYPE=SEND the name of the reply-to queue is specified. As a receive queue it must also be named in the MQIRCVQ parameter of DSLKPROC TYPE=RECEIVE.

The reply-to queue receives reply or report messages. Therefore it must be specified in combination with the ACKWQ, COAWQ, CODWQ, and EXCEPT=YES parameters of DSLKPROC TYPE=SEND. The queue can be shared between several send processes.

The reply-to queue is defined in the MQSeries as a local queue. The same attributes apply to reply-to queues as for other receive queues.

MERVA-MQI Attachment sets the name of the reply-to queue in the field *ReplyToQ* of the MQI control block MQMD.

Defining the Control Queues

Control queues are required both in a send process and receive process. They assure the message integrity for the message transfer from MERVA ESA to the MQSeries and vice versa.

Control Queues in a Send Process

A send process uses two types of control queues:

- MERVA ESA control queue
- MQI control queue

Before a MERVA ESA message is put to the MQI send queue, it is written to the MERVA ESA control queue. The message contains a sequence number starting with 1 and ending with the number of messages on the MQI send queue to be committed. If an error occurs and the send process fails to put each message from the MERVA ESA send queue to the MQI send queue, the messages in the MERVA ESA control queue can be processed again when the send process is started the next time.

Before the messages in the MQI send queue are committed, MERVA-MQI Attachment puts a control message to the MQI control queue. The control message contains the number of MQI messages put to the MQI send queue since the last

commit. During message recovery MERVA-MQI Attachment uses this information to decide which of the messages in the MERVA ESA control queue have to be put again to the MQI send queue.

The MERVA ESA control queue name is specified in the MRVCTLQ parameter of DSLKPROC TYPE=SEND. The control queue can be shared between several send processes. The control queue is defined in the MERVA ESA function table DSLFNTT with KEY1 and KEY2 (refer to "Using the Keys for the MERVA ESA Queues" on page 322 for details).

The MQI control queue name is specified in the MQICTLQ parameter of DSLKPROC TYPE=SEND. The control queue can be shared between several send processes if MERVA-MQI Attachment is running under MVS. The control queue is defined in the MQSeries as a local queue. The following attributes must be specified:

- For MQSeries for MVS/ESA, use the DEFINE QLOCAL command:
  ```
  DEFINE QLOCAL GET(ENABLED) PUT(ENABLED) NOTRIGGER ...
  ```
- For MQSeries for VSE/ESA:
  - On the LOCAL QUEUE DEFINITION screen:
    - Get Enabled: Y
    - Put Enabled: Y
  - On the Queue Extended Definition screen: Trigger Enable: N

Control Queue in a Receive Process

A receive process requires only a MERVA ESA control queue. MERVA-MQI Attachment writes each message retrieved from an MQI receive queue to the control queue. The message on the control queue gets a unique key, the contents of the field *MsgId* from the MQI control block MQMD. After commit, the retrieved messages are deleted from the MQI receive queue.

If an error should occur during processing, the queue manager backs out the changes on the MQI receive queue since the last commit. That is, it restores the uncommitted messages to the MQI receive queue. When the receive process is started the next time, the unique key assures that MERVA-MQI Attachment processes the messages from the MQI receive queue only once. MERVA-MQI Attachment ignores the messages from the MQI receive queue which are already in the MERVA ESA control queue (see also "Keys for a Receive Queue" on page 322).

The MERVA ESA control queue name is specified in the MRVCTLQ parameter of DSLKPROC TYPE=RECEIVE. The control queue can be shared between several receive processes. It is defined in the MERVA ESA function table DSLFNTT with KEY1 and KEY2 (refer to "Using the Keys for the MERVA ESA Queues" on page 322 for details).

Note: The control queue **cannot** be shared by the receive processes when an MQI reply message requests one or more of the following MQI reports:

- COA
- COD
- Exception (MVS only)

In this case, two receive processes are required:

- One to receive the MQI report and reply messages associated with the previously sent MQI request message

- One to receive the MQI report messages associated with the MQI reply message that was sent on behalf of the MQI request message

Defining the Start Queue

An operator can use the start queue to initiate one or more of the MERVA-to-MQI send processes and one or more of the MQI-to-MERVA receive processes. This is an alternative to the automatic start of a send or receive process.

The parameter MRVSTAQ specifies the name of the MERVA ESA start queue. The queue must be defined in the MERVA ESA function table DSLFNTT with the following parameters:

- MQI=YES
- QUEUE=DUMMY
- TRAN=DSLS for a send process, or TRAN=DSLR for a receive process

An operator can initiate send or receive processes by entering the command SF followed by the name of the start queue. The operator must be authorized to apply the SF command to a function defined with MQI=YES.

Note: To enable an operator-controlled start, do not specify the parameter STATUS=AUTO for the start queue.

When the start queue is specified for one send process or receive process only, it processes all send or receive queues of the send process or receive process. The queues are processed in the order of their appearance in the list of send queue pairs (parameter ALLSNDQ of DSLKPROC TYPE=SEND) or receive queues (parameter MQIRCVQ of DSLKPROC TYPE=RECEIVE).

When the start queue is specified for more than one send process or receive process, it processes all send or receive queues of all the involved send processes or receive processes. The messages in the queues are processed in the order of the send or receive processes in the process table DSLKPROC. That is, the messages of the first send or receive process are processed in the order of the queues in the list of send queue pairs or receive queues. Then the messages of the second send or receive process are processed and so on until the messages of the last send or receive process have been processed.

Enabling a Disabled Receive Process

If triggering is enabled for an MQI receive queue, MERVA-MQI Attachment processes the messages on this queue automatically. However, when an error occurs during processing, MERVA-MQI Attachment running under MVS issues an error message and disables triggering for the queue before it stops processing.

Under MQSeries for MVS/ESA, disabling the trigger mechanism is required. Otherwise, the queue manager could call MERVA-MQI Attachment again when other messages arrive on the queue satisfying the conditions for triggering. As MERVA-MQI Attachment cannot remove the erroneous message from the queue, the receive process would never come to an end.

When a start queue is defined for a receive process, triggering can be enabled again. This occurs when the operator enters the SF command followed by the start queue name. It is not required when MERVA-MQI Attachment runs under VSE.

Notes:

1. The problem that caused MERVA-MQI Attachment to disable triggering must have been solved before the SF command can enable triggering.

2. MERVA-MQI Attachment writes the indicator **DSLKQRbb** as data to the trigger message (b represents a blank). The TRIGDATA attribute of the queue shows this data when you issue the DISPLAY QUEUE command. If the data starts with the indicator **DSLKQRbb** as the first 8 characters, you must not change the indicator when you issue the ALTER QLOCAL command specifying the TRIGDATA attribute with new data. The SF command can enable triggering only when this indicator is available.

Defining the Error Queue

After MERVA-MQI Attachment has successfully received a message from an MQI receive queue, errors can occur during further processing; for example:

- Accessing the MERVA ESA control queue for getting or putting a message might fail.
- Length fields in the received message may be invalid.
- Mapping the message to the internal TOF format might fail.

Without an error queue, MERVA-MQI Attachment stops processing when such an error occurs. Under MQSeries for MVS/ESA, triggering is inhibited. When an error queue is available, MERVA-MQI Attachment writes the received message to this queue, and continues processing.

The error queue can be either the MQSeries dead-letter queue or a user-defined MQI queue. This can be specified in the MQIERRQ parameter of DSLKPROC TYPE=RECEIVE. In MQSeries for MVS/ESA, specifying MQIERRQ=* requests that the dead-letter queue be used.

MERVA-MQI Attachment always puts the dead-letter header MQDLH before the message, even if it writes the message to a user-defined error queue. The *Reason* field of the MQDLH contains a code describing the error. For an explanation of the codes, refer to *MERVA for ESA Messages and Codes*.

Defining the Commit Frequency

MERVA-MQI Attachment commits the changes on the MQI send or receive queues after a given number of messages from the MERVA ESA send queue or MQI receive queue has been successfully processed. MERVA-MQI Attachment uses the number specified in the COMMIT parameter as a limit to perform the commit.

The considerable overhead connected with a commit affects the overall performance. Committing a large number of messages requires less commits, but the commit takes more time. Committing a small number of messages requires more commits, but the commit is faster. Varying the value of the COMMIT parameter can help to find an acceptable compromise.

Defining the Scheduling Frequency

When MERVA-MQI Attachment is running under IMS, the number of messages to be processed in a single scheduling can be defined.

For a send process, MERVA-MQI Attachment uses either the value defined in the MERVA ESA customization parameter module DSLPRM, or the value defined for a MERVA ESA send or start queue in the function table DSLFNTT. The value is specified in the MSGLIM parameter of the macros DSLPARM or DSLFNT. If specified in the function table, MERVA-MQI Attachment uses this value.

For a receive process, it depends on how the receive process was started.

- If it was triggered, MERVA-MQI Attachment uses either the value defined in the module DSLPRM, or the value defined in the USERDATA parameter of an MQI PROCESS definition. If specified in the PROCESS definition, MERVA-MQI Attachment uses this value.

  The characters 1 to 12 of the USERDATA parameter can contain the following specification:

  MSGLIM=nnnnn.

  nnnnn represents the value for MSGLIM in the range 1 to 65535. Leading zeros need not be specified.

  The characters 13 to 16 of the USERDATA parameter are reserved.
- If an operator started the receive process or if it was started automatically during MERVA ESA startup, MERVA-MQI Attachment uses either the value defined in the module DSLPRM, or the value defined for the start queue in the function table DSLFNTT. If specified in the function table, MERVA-MQI Attachment uses this value.

The minimum value of MSGLIM and COMMIT determines the commit frequency.

After MERVA-MQI Attachment has processed the specified number of messages, it inserts a message containing control information into the IMS message queue. The message is:

- The TUCB if a MERVA ESA send or start queue is involved
- The MQI trigger message if the receive process was started by the queue manager

Then MERVA-MQI Attachment terminates and is available for rescheduling by IMS. When IMS schedules MERVA-MQI Attachment again, it uses the control information from the retrieved TUCB or trigger message to determine the appropriate queue of the send or receive process.

Defining the Wait Interval for Message Retrieval

When MERVA-MQI Attachment is triggered by the queue manager, it sets up an MQI-to-MERVA receive process and retrieves all the messages from the MQI receive queue until the queue is empty.

The condition for an empty queue can be made time dependent. Assume that the last retrieval did not supply a message from the queue. The GETWAIT parameter of DSLKPROC TYPE=RECEIVE specifies the maximum time MERVA-MQI Attachment waits for a message to arrive. If no message has arrived after this time has elapsed, MERVA-MQI Attachment considers the queue empty and terminates the current receive process.

The overhead for triggering and terminating MERVA-MQI Attachment in short periods can have an impact on the performance. This can occur if numerous small groups of messages arrive on the queue discontinuously. If the GETWAIT interval is shorter than the time between the arrivals of most of the messages, MERVA-MQI Attachment is often triggered and terminated. On the other hand, a long GETWAIT interval allocates the resources of MERVA ESA and MQSeries longer than necessary in many cases. There is no general answer for what is the right GETWAIT interval. Each installation has to find an appropriate value based on the frequency and amount of its message transfer.

Defining the Next Processing Step

In the NEXT parameter of DSLKPROC TYPE=SEND, the handling of an MQI datagram or request message in the next processing step can be specified, after the datagram or request message has been processed. The message has either been received from an application or correlated with a reply or report message.

The following options are available:

- STANDARD

  A datagram is sent again as a datagram. How a request message is handled depends on its current status: correlated or received. A correlated request message can be sent as a request message. Sending a correlated request message as a datagram is not recommended as some control fields are kept which are not appropriate to a datagram. A received request message is transformed to, and sent as, a reply message.

- NEW

  Using this option, the MQI message type can be changed independently of the current status of the message. That is, a datagram and a request message can be sent using the current message type or the changed one. Thus a datagram can be sent as a request message and a request message can be sent as a datagram. The new message type is determined by the ACKWQ parameter of DSLKPROC TYPE=SEND. A datagram is sent if the ACKWQ parameter is omitted. A request message is sent if the name of a MERVA ESA acknowledgment wait queue is specified in the ACKWQ parameter. A received request message is not transformed to a reply message.

- FORWARD

  This option preserves the MQI message type and, with some exceptions, the control information associated with a datagram and request message. The current status, received or correlated, is not relevant. A datagram can be forwarded as a datagram. A request message can be forwarded as a request message. The message attributes contained in the MQI message descriptor MQMD are kept with the following exceptions:

  – For a received message, the encoding is set to the native machine encoding after the message was converted.

  – For a received message, the coded character set identifier (CCSID) is set to the requested CCSID after the message was converted.

  – If a user exit is specified in the EXIT parameter of DSLKPROC TYPE=SEND, the format name is set depending on the MQFMT parameter of DSLKPROC TYPE=SEND and additional data provided by the user exit.

  The received message can already contain additional data, as shown in Figure 138 on page 302. In this case, at least the binary length fields must have been converted or be in the native machine encoding. In order to avoid checking errors when the message is mapped, the character data should have been converted to EBCDIC.

  If a user exit is not specified in the EXIT parameter, any existing additional data remain unchanged when the message is forwarded. A user exit can supply another additional data. It is up to the user exit whether it appends the new additional data to the old one, or overwrites the old additional data with the new one.

Requesting Message Conversion

In the DSLKPROC table, the following parameters control the conversion of message data consisting of characters and binary length fields:

CNVDEST This parameter can be specified for a send process when MERVA-MQI Attachment is running under VSE. It specifies the name of the destination platform when messages are to be fully or partially converted before they are sent.

CONVERT This parameter can be specified for a receive process. It enables or disables the conversion of received messages.

The different approaches are described in "Converting the Message Data" on page 336.

Requesting Message Security

MERVA-MQI Attachment offers the following security services:

- Encrypting and creating an authentication checksum for messages to be sent
- Decrypting and authenticating received messages

It uses proprietary algorithms to encrypt, decrypt, and authenticate the message data. How you request these security services depends on your environment:

- For MVS, you activate MQSeries channel exits
- For VSE, you set a parameter in the DSLKPROC table

Activating MQSeries Channel Exits (MVS only)

When MERVA-MQI Attachment is running under MVS, use one of the following two MQSeries for MVS/ESA channel exits:

The message exit (DSLKMEA)

You can use this exit for all types of MQSeries channels except client-connection and server-connection channels. If, after decryption, DSLKMEA detects an authentication error in a received message, MERVA-MQI Attachment assigns status AUTER to the message and routes it to an error queue. The sample routing table DSLKQRT shows how to handle messages with the status AUTER.

Specify the name of this exit in the MSGEXIT parameter of the DEFINE CHANNEL command: **`MSGEXIT('DSLKMEA')`**

For messages to be sent to MQSeries for VSE/ESA, also specify **`MSGDATA('VSE')`** for the DEFINE CHANNEL command. If an outgoing message requests a COA or COD report with data from MQSeries, this causes DSLKMEA to modify the appropriate report option, thereby requesting a report without data.

The send and receive exit (DSLKSREA)

You can use this exit for all types of MQSeries channels; however, it is recommended that you use it only for channel types that DSLKMEA cannot handle (that is, client-connection and server-connection channels). If, after decryption, DSLKSREA detects an authentication error in a received message, it requests that MQSeries close the channel. This is usually less desirable than taking advantage of the error handling provided by DSLKMEA.

For a channel connection to MQSeries for VSE/ESA, DSLKSREA cannot be used.

Specify the name of this exit in the DEFINE CHANNEL command:

SENDEXIT('DSLKSREA')	For a send exit
RCVEXIT('DSLKSREA')	For a receive exit

Except for a client-connection and server-connection channel, also specify one of the following parameters for the DEFINE CHANNEL command:

SENDDATA('ALL')	For a send exit
RCVDATA('ALL')	For a receive exit

The channel exits DSLKMEA and DSLKSREA must be contained in the non-authorized libraries defined by a CSQXLIB DD statement in the JCL of the started task for the MQSeries channel initiator.

For more information about the DEFINE CHANNEL command, refer to the *MQSeries Command Reference*.

Parameter in DSLKPROC (VSE only)

When MERVA-MQI Attachment is running under VSE, you request encryption, decryption, and authentication of message data by specifying the DSLKPROC parameter SECURE=AUTHENCR:

- When specified for a send process, MERVA-MQI Attachment encrypts and creates an authentication checksum for the outgoing messages. If an outgoing message requests a COA or COD report with data from MQSeries, MERVA-MQI Attachment modifies the appropriate report option, thereby requesting a report without data.
- When specified for a receive process, MERVA-MQI Attachment decrypts the incoming messages, recalculates the checksum, and compares it with the checksum sent with the message. If it detects an authentication error, MERVA-MQI Attachment assigns status AUTER to the message and routes it to an error queue.

Writing the MQI Message Types to the MERVA ESA Journal

Each MQI message type processed in a MERVA-to-MQI send process or MQI-to-MERVA receive process can be selected to be written to the MERVA ESA journal. The following message types are common for a send and receive process:

- Datagram
- Request message
- Reply message

They are written to the MERVA ESA journal if the parameters of DSLKPROC are specified as follows:

JRNDGRM=YES	Datagram
JRNRQST=YES	Request message
JRNRPLY=YES	Reply message

The following MQI report message types are handled in a receive process only:

- COA
- COD
- Exception

They are written to the MERVA ESA journal if the parameters of DSLKPROC TYPE=RECEIVE are specified as follows:

JRNRCOA=YES	COA
JRNRCOD=YES	COD
JRNREXC=YES	Exception (supported only when MERVA-MQI Attachment is running under MVS)

The journal record identifiers for the MQI message types are described in *MERVA for ESA Concepts and Components*.

Notes:

1. The processing of the MQI message types including the reports must have been requested in a send process. For example, if the ACKWQ parameter of DSLKPROC TYPE=SEND was not specified, the send process can only send datagrams. Specifying JRNDGRM=NO and JRNRQST=YES prevents a journal entry as no request message can be sent.
2. Other reports than COA, COD, or exception are always written to the journal. MERVA-MQI Attachment does not support these report message types and writes them to the journal whenever they occur.

Issuing the MERVA ESA Operator Messages

MERVA-MQI Attachment issues messages to inform the MERVA ESA operator on the current state of the send and receive processes. MERVA-MQI Attachment can write the operator messages to two output media of MERVA ESA:

- The display message table
- The journal

The OPMSDM parameter of DSLKPROC directs the messages to the display message table, the OPMSJRN parameter of DSLKPROC directs them to the journal.

Three levels can be chosen for the operator messages:

NONE	No messages
SUBSET	Part of the messages
FULL	All messages

The NONE level suppresses the output of information messages to the display message table or the journal. Error messages, however, are always issued to the display message table or the journal.

The SUBSET level informs when a MERVA ESA send queue or MQI receive queue was started for being processed or ended processing. This level is suited to inform on the activities of automatically started queues in a send or receive process.

The FULL level causes MERVA-MQI Attachment to issue all available operator messages. It includes the SUBSET level. Additionally it informs when a send or receive process was started or ended. This level is suited to inform on the processing of one or more send and receive processes started by an operator.

Using the Display Message Table

The display message table contains messages issued from MERVA-MQI Attachment and from other components of MERVA ESA. The operator command DM shows the messages contained in the display message table. In order to select the messages issued by MERVA-MQI Attachment, the operator is recommended to enter the prefix DSL6 as a parameter of DM. For example:

- DM DSL6

- DM FIRST DSL6
- DM LAST DSL6

Messages issued for a send process start with 'MQSND:', messages issued for a receive process start with 'MQRCV:'.

The size of the display message table can be adjusted using the DM parameter of the DSLPARM macro.

Setting MERVA ESA Traces

MERVA-MQI Attachment writes two types of traces:

- The debugging trace
- The processing trace

MERVA ESA provides a debugging trace for MFS and TOF services. This trace can be used to debug problems when MERVA-MQI Attachment uses MFS and TOF services.

The parameter TRACMFS=YES of DSLKPROC activates the MFS trace. The parameter TRACTOF=YES of DSLKPROC activates the TOF trace. Both types of trace can be requested for a send and receive process.

The major modules of MERVA-MQI Attachment also provide processing trace information. They write status entries into the trace table. The TRACE parameter of the DSLPARM macro controls the MERVA ESA processing trace status. When TRACE=EXT is set, the trace table is in main storage and is written to the MERVA ESA journal.

Refer to the *MERVA for ESA Diagnosis Guide* for details.

Sample Process Tables DSLKPSAM (MVS) and DSLKPSMV (VSE)

Figure 141 on page 318 shows a part of the sample process tables DSLKPSAM (for MVS) and DSLKPSMV (for VSE). Each of these tables contains the definitions for one send and one receive process.

```
DSLKPROC TYPE=INITIAL                                      [1]
DSLKPROC TYPE=SEND,                                  *     [2]
      NAME=SPROC1,                                   *     [3]
      ALLSNDQ=((DSLMRSQ1,DSL.MQI.SNDRCV)),           *     [4]
      COAWQ=DSLMRCOA,                                *     [5]
      CODWQ=DSLMRCOD,                                *     [6]
      ACKWQ=DSLMRAWQ,                                *     [7]
      MQICTLQ=DSL.MQI.CONTROL,                       *     [8]
      MRVCTLQ=(DSLMRCQS,CONTINUE),                   *     [9]
      MRVSTAQ=DSLMRSTS,                              *    [10]
      REPLYTQ=DSL.MQI.REPLY_TO_Q,                    *    [11]
      EXIT=8001,                                     *    [12]
      MQFMT=BLANK,                                   *    [13]
      OPMSDM=SUBSET                                       [14]
DSLKPROC TYPE=RECEIVE,                               *    [15]
      NAME=RPROC1,                                   *    [16]
      MQIRCVQ=(DSL.MQI.REPLY_TO_Q),                  *    [17]
      MRVCTLQ=DSLMRCQR,                              *    [18]
      MRVSTAQ=DSLMRSTR,                              *    [19]
      EXIT=8002,                                     *    [20]
      OPMSDM=SUBSET                                       [21]
DSLKPROC TYPE=FINAL                                        [22]
END
```

Figure 141. Coding Example of a MERVA-MQI Attachment Process Table

Notes:

[1] TYPE=INITIAL

This must be the first macro. The generated name of the process table is always DSLKPROC. The process table must be link-edited using this name. Under CICS, a program definition for DSLKPROC in the CICS system definition file CSD is also required.

[2] TYPE=SEND

This macro defines a MERVA-to-MQI send process.

[3] NAME=SPROC1

Specifies the send process name SPROC1. When more than one send process is defined, this name must be unique for all send processes.

[4] ALLSNDQ=((DSLMRSQ1,DSL.MQI.SNDRCV))

Specifies one MERVA ESA and MQI send queue pair. The MERVA ESA send queue DSLMRSQ1 is associated to the MQI send queue DSL.MQI.SNDRCV. MERVA-MQI Attachment retrieves the messages to be sent from the MERVA ESA queue DSLMRSQ1 and puts them to the MQI queue DSL.MQI.SNDRCV.

[5] COAWQ=DSLMRCOA

Specifies the COA wait queue name DSLMRCOA. This indicates to MERVA-MQI Attachment that it has to request a COA report message for each message to be sent. MERVA-MQI Attachment stores messages waiting for a COA report in the MERVA ESA queue DSLMRCOA.

[6] CODWQ=DSLMRCOD

Specifies the COD wait queue name DSLMRCOD. This indicates to MERVA-MQI Attachment that it has to request a COD report message for each message to be sent. MERVA-MQI Attachment stores messages waiting for a COD report in the MERVA ESA queue DSLMRCOD.

[7] ACKWQ=DSLMRAWQ

Specifies the acknowledgment wait queue name DSLMRAWQ. This indicates to MERVA-MQI Attachment that it has to send MQI request messages in this send process. MERVA-MQI Attachment stores request messages waiting for an acknowledgment in the MERVA ESA queue DSLMRAWQ. The acknowledgment is an MQI reply message from the receiving application.

[8] MQICTLQ=DSL.MQI.CONTROL

Specifies the MQI control queue name DSL.MQI.CONTROL. The MQI control queue is required to assure message integrity. Before the messages in the MQI send queue DSL.MQI.SNDRCV are committed to the MQSeries, MERVA-MQI Attachment stores control information in the MQI control queue DSL.MQI.CONTROL. Thus the control information is also committed to the MQSeries.

[9] MRVCTLQ=(DSLMRCQS,CONTINUE)

Specifies the MERVA ESA control queue name DSLMRCQS and the option CONTINUE. The MERVA ESA control queue is required to assure message integrity. Before a MERVA ESA message is put to the MQI send queue DSL.MQI.SNDRCV, MERVA-MQI Attachment retrieves it from the MERVA ESA send queue DSLMRSQ1 and writes it to the MERVA ESA control queue DSLMRCQS.

The option CONTINUE applies only when MERVA-MQI Attachment has to recover the messages in the queue DSLMRCQS. CONTINUE indicates to MERVA-MQI Attachment to remove the messages from the queue DSLMRCQS if an error occurs during message recovery. MERVA-MQI Attachment routes the messages to an error queue and continues to process the messages in the MERVA ESA send queue DSLMRSQ1.

[10] MRVSTAQ=DSLMRSTS

Specifies the MERVA ESA start queue name DSLMRSTS. An authorized MERVA ESA operator can enter the command SF DSLMRSTS in order to start the send process SPROC1.

[11] REPLYTQ=DSL.MQI.REPLY_TO_Q

Specifies the MQI reply-to queue name DSL.MQI.REPLY_TO_Q. MERVA-MQI Attachment provides this name to the local MQSeries which passes it on to the remote MQSeries and the receiving application. Thus the remote MQSeries and the receiving application get the information on the destination for their MQI report and reply messages. The reply-to queue DSL.MQI.REPLY_TO_Q stores the COA and COD report messages sent from the remote MQSeries, and the reply messages sent from the receiving application.

Note: A remote MQSeries is not required. The local MQSeries can also generate COA and COD report messages to confirm the arrival and delivery of the received request messages.

[12] EXIT=8001

Specifies the MFS user exit number 8001. This number represents the sample MFS user exit DSLKQ001. The exit provides additional data for the MQI request messages to be sent. Although written in Assembler, the exit complies to the high-level language interface for MFS user exits.

[13] MQFMT=BLANK

Specifies that the MQI data format name consists entirely of blanks. As the user exit DSLKQ001 provides additional message data, MERVA-MQI Attachment structures the data in the MQI request messages using 4-byte fields, as shown in Figure 138 on page 302. The blank character format name beginning other than "MQ" indicates to the receiving application that the request messages contain length fields.

[14] OPMSDM=SUBSET

Specifies that MERVA-MQI Attachment issues a subset of operator messages adding them to the MERVA ESA display message table. These messages show when the MERVA ESA send queue DSLMRSQ1 was started for processing or ended processing. The operator can enter the command DM to see the messages in the display message table.

[15] TYPE=RECEIVE

This macro defines an MQI-to-MERVA receive process.

[16] NAME=RPROC1

Specifies the receive process name RPROC1. When more than one receive process is defined, this name must be unique for all receive processes.

[17] MQIRCVQ=(DSL.MQI.REPLY_TO_Q)

Specifies the MQI receive queue name DSL.MQI.REPLY_TO_Q. This is the name of the reply-to queue specified in the REPLYTQ parameter of DSLKPROC TYPE=SEND. The MQI queue DSL.MQI.REPLY_TO_Q stores the received COA and COD report messages, and the received reply messages until MERVA-MQI Attachment has processed them.

[18] MRVCTLQ=DSLMRCQR

Specifies the MERVA ESA control queue name DSLMRCQR. The MERVA ESA control queue is required to assure message integrity. Before each received message can be routed to its MERVA ESA target queue, MERVA-MQI Attachment retrieves it from the MQI receive queue DSL.MQI.REPLY_TO_Q and writes it to the MERVA ESA control queue DSLMRCQR.

[19] MRVSTAQ=DSLMRSTR

Specifies the MERVA ESA start queue name DSLMRSTR. An authorized MERVA ESA operator can enter the command SF DSLMRSTR in order to start the receive process RPROC1.

[20] EXIT=8002

Specifies the MFS user exit number 8002. This number represents the sample MFS user exit DSLKQ002. After correlation of the waiting request message with the received reply message, the exit reads the reply message data and writes it to one or more target fields. Although written in Assembler, the exit complies to the high-level language interface for MFS user exits.

[21] OPMSDM=SUBSET

Specifies that MERVA-MQI Attachment issues a subset of operator messages adding them to the MERVA ESA display message table. These messages show when the MQI receive queue DSL.MQI.REPLY_TO_Q was

started for processing or ended processing. The operator can enter the command DM to see the messages in the display message table.

[22] TYPE=FINAL

This must be the last macro and is followed by the Assembler END statement.

Using the Keys for Message Identification and Correlation

MERVA-MQI Attachment uses one or two keys to uniquely identify messages on the MQI and MERVA ESA queues.

Using the Keys for the MQI Queues

MERVA-MQI Attachment uses two fields of the MQI control block MQMD as key fields for the messages on the MQI queues:

- *MsgId*
- *CorrelId*

The field contents is different depending on the MQI queue and the type of the message in the queue.

Keys for a Send Queue

This queue can contain datagrams, request, and reply messages. MERVA-MQI Attachment sets the key fields for these MQI messages as follows:

- Datagram and request message:

  MsgId The queue manager generates a unique key.

  CorrelId Name of the current send process (bytes 1 to 8), followed by the name of the MERVA ESA send queue (bytes 9 to 16), followed by blanks (bytes 17 to 24).

  The send process name is specified in the NAME parameter of DSLKPROC. The MERVA ESA send queue name is specified in the ALLSNDQ parameter of DSLKPROC.

- Reply message:

  MERVA-MQI Attachment sets *MsgId* and *CorrelId* depending on the report options specified for the received MQI request message. The report options are contained in the *Report* field of the MQI control block MQMD.

  The following options are supported:

  Copy *MsgId* **to** *CorrelId*
  MERVA-MQI Attachment copies the *MsgId* of the request message to the *CorrelId* of the reply message.

  Pass *CorrelId*
  MERVA-MQI Attachment copies the *CorrelId* of the request message to the *CorrelId* of the reply message.

  New *MsgId*
  The queue manager generates a new *MsgId* for the reply message.

  Pass *MsgId*
  MERVA-MQI Attachment copies the *MsgId* of the request message to the *MsgId* of the reply message.

Keys for a Control Queue

This queue can contain control messages. MERVA-MQI Attachment uses only the *CorrelId* field to identify the control messages. The *MsgId* field can have any identifier to match.

CorrelId contains the name of the current send process (bytes 1 to 8), followed by the name of the MERVA ESA send queue (bytes 9 to 16), followed by blanks (bytes 17 to 24). The send process name is specified in the NAME parameter of DSLKPROC. The MERVA ESA send queue name is specified in the ALLSNDQ parameter of DSLKPROC.

Keys for a Receive Queue

This queue can contain datagrams, request, reply, and report messages. MERVA-MQI Attachment always retrieves all messages contained in a receive queue. Therefore there are no selection criteria required. The fields *MsgId* and *CorrelId* can have any identifier to match.

Note: The application sending a datagram or request message to a receive queue owned by MERVA-MQI Attachment **must provide a unique identifier** in the *MsgId* field. MERVA-MQI Attachment uses the unique *MsgId* to assure the message integrity for the MQI-to-MERVA receive processes.

The contents of the *MsgId* and *CorrelId* for a received reply and report message are determined by the report options for the datagram and request message sent by MERVA-MQI Attachment. The application sending a reply message (or even a report message) to a receive queue owned by MERVA-MQI Attachment must follow the report option instructions; that is, the application must support the following report options:

- Copy *MsgId* to *CorrelId*
- Pass *CorrelId*
- New *MsgId*
- Pass *MsgId*

Using the Keys for the MERVA ESA Queues

The following MERVA ESA queues require one or two key fields when they are defined in the MERVA ESA function table DSLFNTT:

- The control queue
- The acknowledgment wait queue
- The COA wait queue
- The COD wait queue

Keys for the Control Queue

The control queue is defined in DSLFNTT with KEY1=(DSLKKEY1,16,,NOMOD) and KEY2=(DSLKMSID,24,,NOMOD). DSLKMSID is a subfield of the TOF field DSLKMQMD. The TOF fields DSLKKEY1 and DSLKMQMD are described in "Using the Control Fields" on page 323.

- When used for a send process, KEY1 contains the name of the send process followed by the name of the MERVA ESA send queue. The name of the send process is specified in the NAME parameter of DSLKPROC TYPE=SEND. The name of the MERVA ESA send queue is specified in the ALLSNDQ parameter of DSLKPROC TYPE=SEND.

  When used for a receive process, KEY1 contains the name of the receive process followed by an index for the MQI receive queue. The name of the receive

process is specified in the NAME parameter of DSLKPROC TYPE=RECEIVE. The index is an 8-digit number in the range 00000001 to 00000010. It represents an MQI receive queue on position 1 to 10 in the receive queue list of the MQIRCVQ parameter of DSLKPROC TYPE=RECEIVE.

KEY1 must have a length of 16 bytes. The subparameter NOMOD is required.

- KEY2 contains the contents of the field *MsgId* from the MQI control block MQMD. KEY2 must have a length of 24 bytes. The subparameter NOMOD is required.

Key for the Wait Queues

The wait queues are defined in DSLFNTT with KEY1=(DSLKMSID,24,,NOMOD). DSLKMSID is a subfield of the TOF field DSLKMQMD. The TOF field DSLKMQMD is described in "Using the Control Fields".

KEY1 contains the contents of the field *MsgId* from the MQI control block MQMD. KEY1 must have a length of 24 bytes. The subparameter NOMOD is required.

Correlating MQI Report and Reply Messages

Message correlation means searching in one of the wait queues for the message that is waiting for a particular report or reply message. The message that is waiting can be a datagram, a request message, or a reply message. A reply message can wait only for a report message. When the waiting message is found, the relevant data of the report or reply message is written to that message. It is possible that a reply message contains no data. The only data in a report message is the feedback or reason code.

The correlation requires the following:

Send process determination
MERVA-MQI Attachment obtains the name of the send process from the field *CorrelId* of the MQI control block MQMD. When the report was requested by a reply message, the MQMD field *ApplIdentityData* contains the send process name (see "Setting the MQI Report Options" on page 304).

Message identification in one of the wait queues
After the send process has been determined, MERVA-MQI Attachment retrieves the message from the appropriate wait queue using the MQMD field *MsgId* as KEY1.

Using the Control Fields

MERVA-MQI Attachment writes control information to the TOF describing the message transfer in a MERVA-to-MQI send process and an MQI-to-MERVA receive process.

List of Control Fields

The following list shows the fields which contain the control information. MERVA-MQI Attachment writes the fields to the TOF using the nesting identifier 0. The fields are defined in the MERVA ESA field definition table DSLFDTT. An asterisk (*) indicates that a field contains subfields.

DSLKDATA Additional message fields (*)

DSLKKEY1 KEY1 for MERVA ESA control queue

DSLKMAWQ MERVA ESA ACK wait queue name

DSLKMCNT Counter of uncommitted messages

DSLKMDRQ MQI control block MQMD used in a reply message

DSLKMGMO MQI control block MQGMO (*)

DSLKMPMO MQI control block MQPMO (*)

DSLKMQMD MQI control block MQMD (*)

DSLKMQOD MQI control block MQOD

DSLKMSQN MERVA ESA send queue name

DSLKPNAM Send or receive process name

DSLKPRTY MQI message priority

DSLKRPLY MQI reply message data (*)

DSLKRPRT MQI report message data

DSLKSTAT Message status

DSLKTYPE MQI message type

FMT/ESA with MERVA-MQI Attachment writes the following fields:

DSLKAKRQ Request for an acknowledgment

DSLKPNM Name of the send process

DSLKISN Input sequence number (ISN)

DSLKOSN Output sequence number (OSN)

Some of the fields are used in a routing table to control the message flow in a send and receive process. The sample routing table DSLKQRT shows how the MQI message types can be handled as MERVA ESA messages (refer to "Sample Routing Table DSLKQRT" on page 328 for details).

MERVA-MQI Attachment writes the following fields both for a send and receive process:

DSLKAKRQ Request for an acknowledgment (FMT/ESA only)

DSLKKEY1 KEY1 for MERVA ESA control queue

DSLKMQMD MQI control block MQMD

DSLKPNAM Send or receive process name

DSLKPRTY MQI message priority

DSLKRPLY MQI reply message data

DSLKSTAT Message status

DSLKTYPE MQI message type

MERVA-MQI Attachment writes the following fields for a send process only:

DSLKMAWQ MERVA ESA ACK wait queue name

DSLKMCNT Counter of uncommitted messages

DSLKMDRQ MQI control block MQMD used in a reply message

DSLKMPMO MQI control block MQPMO

DSLKMQOD MQI control block MQOD

DSLKMSQN MERVA ESA send queue name

MERVA-MQI Attachment writes the following fields for a receive process only:

DSLKDATA Additional message fields

DSLKMGMO MQI control block MQGMO

DSLKRPRT MQI report message data

Using Message Status Information

The field DSLKSTAT contains the status of the currently processed message. This information is used in a routing table to distinguish between a send and receive process, and to structure the processing of the MQI message types.

Recognizing the Message Status

The status information in DSLKSTAT can have a length of up to five characters. The following values can occur:

ACK Request message correlated with reply message

ACKC Reply message successfully correlated

ACKER Error after correlation with the reply message (written by FMT/ESA)

ACKNC Reply message could not be correlated

AUTER Authentication error in a received datagram or request, reply, or report message

COA Datagram/request/reply message correlated with COA report

COAC COA report successfully correlated

COAER Error after correlation with the COA report (written by FMT/ESA)

COANC COA report could not be correlated

COD Datagram/request/reply message correlated with COD report

CODC COD report successfully correlated

CODNC COD report could not be correlated

ERANY Any error during message recovery

ERCHK MERVA ESA MFS message checking error

ERCNT Counter error during message recovery

ERSND Error while the message is being sent (written by FMT/ESA)

ERSWO Error while routing a SWIFT message (written by FMT/ESA)

EXC Datagram/request/reply message correlated with exception report (MVS only)

EXCC Exception report successfully correlated (MVS only)

EXCNC Exception report could not be correlated (MVS only)

FORWD Datagram or request message was forwarded

RCVD Datagram or request message was received

SENT Datagram/request/reply message was sent

SWIER Error regarding a SWIFT input message (written by FMT/ESA)

UNSUP Unsupported report message was received

Recognizing a Send or Receive Process

The message belongs to a send process when DSLKSTAT contains one of the following values:

- ERANY
- ERCHK
- ERCNT
- ERSND
- ERSWO
- FORWD
- SENT

The other DSLKSTAT values indicate a receive process. Thus it is possible to use a common routing table for send and receive processes.

Recognizing the Message Type

Some values of DSLKSTAT apply to several MQI message types. For example, the values RCVD and SENT. The field DSLKTYPE contains the necessary information to identify the message type.

MERVA-MQI Attachment uses the scheme defined by MQSeries for message type identifiers and provides the following values in DSLKTYPE:

1 Request message

2 Reply message

4 Report message

8 Datagram

Recognizing MFS Errors

Regular SWIFT messages are transported via request messages (DSLKTYPE=1) or datagrams (DSLKTYPE=8). If, for such a message, a MERVA ESA Message Format Service (MFS) error occurs, MERVA-MQI Attachment writes to subfield MSGTRERR of field MSGTRACE the character 0 (zero) followed by the three-character reason code. This value can be used to route erroneous messages to an error queue.

For example, the routing table might contain the following macros to test the contents of MSGTRERR:

```
DSLROUTE TYPE=DEFINE,FIELD=(MSGERR,MSGTRERR,,,,VFIRST,      *
      LASTDA),NOTFND=ERROR,EMPTY=ERROR
DSLROUTE TYPE=TEST,COND=(MSGERR,'0000',NE),TRUE=ERROR
```

MSGTRERR can have a non-zero value only when the status field DSLKSTAT contains one of the following values:

- For a send process, **ERANY** or **ERCHK**
- For a receive process, **RCVD**

Displaying MQI Control Block Data

When MERVA-MQI Attachment puts a MERVA ESA message to an MQI send queue or retrieves a message from an MQI receive queue, two of the following three MQI control blocks are always affected:

MQMD Message descriptor

MQGMO Get-message options

MQPMO Put-message options

MQSeries provides in these control blocks the complete information on the message and the options used for getting it from, or putting it to, the MQI queue (see the *MQSeries Application Programming Reference* for details). MERVA-MQI Attachment writes this control data to the following fields:

DSLKMGMO MQGMO

DSLKMPMO MQPMO

DSLKMQMD MQMD

In a send process, MERVA-MQI Attachment writes the fields:

- DSLKMPMO
- DSLKMQMD

In a receive process, MERVA-MQI Attachment writes the fields:

- DSLKMGMO
- DSLKMQMD

When using the MERVA ESA cover MCB DSLKCOV provided for MERVA-MQI Attachment, the data in the fields DSLKMGMO, DSLKMPMO, and DSLKMQMD can be displayed together with the other data of the sent or received message. There are two ways to display the MQI control block data:

- Using the operator command SHOW
- Specifying the cover MCB in the MERVA ESA function table

When a SWIFT, telex, or user message is displayed without the MQI control block data, the operator can enter the SHOW command. SHOW KCOV or SHOW MQCOVER (the MERVA ESA message type synonym) displays the data as far as it is available for the current message.

The cover MCB can be associated to those queues in the function table DSLFNTT which contain messages processed by MERVA-MQI Attachment. When the parameter MSGID=KCOV is specified in the DSLFNT macro, the cover MCB DSLKCOV is associated to this queue and the MQI control block data can be shown.

Recognizing the Feedback from a Report Message

The feedback or reason code indicates the nature of the report. After correlation, the feedback or reason code from the report message is available in the message having waited for the report. The subfield DSLKFBCK of the field DSLKMQMD contains this code. For a list of feedback and reason codes, refer to the *MQSeries Application Programming Reference*.

Note: When more than one report message was correlated with the waiting message, DSLKFBCK contains the feedback or reason code of the last received report message. For example, when both a COA and COD report message were correlated, DSLKFBCK usually contains the feedback code of the COD report.

The MFS Editing Exits DSLME910 and DSLME911

Each subfield of the fields DSLKMGMO, DSLKMPMO, and DSLKMQMD contains the data of the corresponding field in the appropriate MQI control block. The MFS

editing exits DSLME910 and DSLME911 prepare the external representation of selected control block fields. For example, when a subfield contains binary data, the exits convert it to displayable decimal and hexadecimal characters.

The source code of the exits is provided. If required, an installation can change the external representation of the MQI control block data. The MCB DSLKCTL contains references to the exits DSLME910 and DSLME911. In this MCB, the subfields are defined with the EDIT=910 and EDIT=911 parameter of the DSLLDFLD statement. DSLKCTL is embedded in the cover MCB DSLKCOV by the IMBED=KCTL parameter of the DSLLEXIT statement.

Sample Routing Table DSLKQRT

The following is an excerpt from the sample routing table DSLKQRT. This routing table can be used both for send and receive processes, and is associated with the MERVA ESA control queue specified in the MRVCTLQ parameter of the DSLKPROC macro.

```
DSLKQRT  DSLROUTE TYPE=DEFINE,FIELD=(STATUS,DSLKSTAT,,,,,VFIRST),   *  [1]
               NOTFND=END
         DSLROUTE TYPE=DEFINE,FIELD=(MSGTYP,DSLKTYPE,,,,,VFIRST)       [2]
*-------------------------------------------------------------------*
*        DISTINGUISH BETWEEN SEND AND RECEIVE PROCESSES             *
*-------------------------------------------------------------------*
*
         DSLROUTE TYPE=TEST,COND=(STATUS,'SENT',EQ,SHORT),TRUE=SENT    [3]
         DSLROUTE TYPE=TEST,COND=(STATUS,'FORWD',EQ),TRUE=FWD          [4]
         DSLROUTE TYPE=TEST,COND=(STATUS,'ER',EQ,SHORT),TRUE=ER        [5]
*
         DSLROUTE TYPE=TEST,COND=(STATUS,'RCVD',EQ,SHORT),TRUE=RCVD    [6]
         DSLROUTE TYPE=TEST,COND=(STATUS,'ACK',EQ,SHORT),TRUE=ACK      [7]
         DSLROUTE TYPE=TEST,COND=(STATUS,'COA',EQ,SHORT),TRUE=COA      [8]
         DSLROUTE TYPE=TEST,COND=(STATUS,'COD',EQ,SHORT),TRUE=COD      [9]
         DSLROUTE TYPE=TEST,COND=(STATUS,'EXC',EQ,SHORT),TRUE=EXC      [9]
         DSLROUTE TYPE=TEST,COND=(STATUS,'UNSUP',EQ),TRUE=UNSUP,    *  [9]
               FALSE=END
*-------------------------------------------------------------------*
*        MERVA-TO-MQI SEND PROCESS                                  *
*-------------------------------------------------------------------*
*
SENT     DSLROUTE TYPE=TEST,COND=(MSGTYP,'1',EQ,SHORT),TRUE=ACKWAIT   [10]
         DSLROUTE TYPE=TEST,COND=(MSGTYP,'2',EQ,SHORT),TRUE=DELREPCM  [11]
         DSLROUTE TYPE=SET,TARGET='DSLMRSNT',GOTO=END                 [12]
ACKWAIT  DSLROUTE TYPE=DEFINE,FIELD=(ACKWQ,DSLKMAWQ,,,,,VFIRST)       [13]
         DSLROUTE TYPE=SET,TARGET=ACKWQ,GOTO=END                      [14]
DELREPCM DSLROUTE TYPE=SET,TARGET='DSLMRDMY',GOTO=END                 [15]
FWD      DSLROUTE TYPE=SET,TARGET='DSLMRSNT',GOTO=END                 [16]
*
ER       ...
*-------------------------------------------------------------------*
*        MQI-TO-MERVA RECEIVE PROCESS                               *
*-------------------------------------------------------------------*
*
RCVD     DSLROUTE TYPE=TEST,COND=(MSGTYP,'8',EQ,SHORT),TRUE=RCVDQ     [17]
         DSLROUTE TYPE=SET,TARGET='DSLMRSQ2'                          [18]
RCVDQ    DSLROUTE TYPE=SET,TARGET='DSLMRRQ1',GOTO=END                 [19]
*
ACK      DSLROUTE TYPE=TEST,COND=(STATUS,'ACK  ',EQ),TRUE=ACKCORR     [20]
         DSLROUTE TYPE=TEST,COND=(STATUS,'ACKNC',EQ),TRUE=ACKNCORR    [21]
         DSLROUTE TYPE=SET,TARGET='DSLMRDMY',GOTO=END                 [22]
ACKCORR  DSLROUTE TYPE=SET,TARGET='DSLMRRQ1',GOTO=END                 [23]
ACKNCORR DSLROUTE TYPE=SET,TARGET='DSLMRNCO',GOTO=END                 [24]
*
COA      DSLROUTE TYPE=TEST,COND=(STATUS,'COA  ',EQ),TRUE=COACORR     [25]
         DSLROUTE TYPE=TEST,COND=(STATUS,'COANC',EQ),TRUE=COANCORR    [26]
```

```
         DSLROUTE TYPE=SET,TARGET='DSLMRDMY',GOTO=END                  [27]
COACORR  DSLROUTE TYPE=DEFINE,FIELD=(ACKWQ,DSLKMAWQ,,,,,VFIRST),    * [28]
               FOUND=COAAWQ
         DSLROUTE TYPE=SET,TARGET='DSLMRRQ1',GOTO=END                  [29]
COAAWQ   DSLROUTE TYPE=SET,TARGET=ACKWQ,GOTO=END                       [30]
COANCORR DSLROUTE TYPE=SET,TARGET='DSLMRNCO',GOTO=END                  [31]
*
COD      ...
EXC      ...
UNSUP    ...
*
END      DSLROUTE TYPE=FINAL
         END
```

Notes:

[1] The field DSLKSTAT is defined as STATUS for testing its contents in later DSLROUTE macros. STATUS contains the status of the message. If the field DSLKSTAT is not found in the TOF, processing stops at the END label. Checking the existence of DSLKSTAT can be used to separate MERVA-MQI Attachment-related parts from other processing logic in a routing table.

[2] The field DSLKTYPE is defined as MSGTYP for testing its contents in later DSLROUTE macros. MSGTYP contains the MQI message type identifiers.

[3] If the first four characters of the status value are **SENT**, the message belongs to a send process. This status value indicates that an MQI datagram, request, or reply message was sent. Processing continues at the SENT label.

[4] If the status value contains **FORWD**, the message also belongs to a send process. This status value indicates that an MQI datagram or request message was forwarded. Processing continues at the FWD label.

[5] If the first two characters of the status value are **ER**, the message also belongs to a send process. The values ERANY, ERCHK, and ERCNT indicate an error during message checking or recovery; the values ERSND and ERSWO indicate an error during FMT/ESA processing. Processing continues at the ER label. The processing logic at the ER label is not shown in this example.

[6] If the first four characters of the status value are **RCVD**, the message belongs to a receive process. This status value indicates that an MQI datagram or request message was received. Processing continues at the RCVD label.

[7] If the first three characters of the status value are **ACK**, the message also belongs to a receive process. The values ACK, ACKC, and ACKNC indicate whether a waiting request message could be correlated with the received reply message; the value ACKER indicates whether an error occurred during FMT/ESA processing after the correlation with the reply message. Processing continues at the ACK label.

[8] If the first three characters of the status value are **COA**, the message also belongs to a receive process. The values COA, COAC, and COANC indicate whether a waiting datagram, request, or reply message can be correlated with the received COA report message; the value COAER indicates whether an error occurred during FMT/ESA processing after the correlation with the COA report. Processing continues at the COA label.

[9] The processing logic concerning the status values beginning with COD and EXC and the status value UNSUP is not shown in this example. A message with any of these status values also belongs to a receive process.

Note: Only a MERVA-MQI Attachment running under MVS can produce status values beginning with **EXC**.

[10] A datagram, request, or reply message was sent. If MSGTYP contains 1 as the first character, the message sent is a request message. Processing continues at the ACKWAIT label.

[11] If MSGTYP contains 2 as the first character, the message sent is a reply message. Processing continues at the DELREPCM label.

[12] The message sent is neither a request nor a reply message. Therefore it is a datagram, as MERVA-MQI Attachment never sends a report message. The datagram is routed to the target queue DSLMRSNT, and processing stops at the END label.

[13] The message sent is a request message. The field DSLKMAWQ is defined as ACKWQ for using its contents in later DSLROUTE macros. ACKWQ contains the name of the acknowledgment wait queue as defined in the ACKWQ parameter of the DSLKPROC macro.

[14] The request message is routed to the acknowledgment wait queue whose name is contained in ACKWQ, and processing stops at the END label. The request message waits in the acknowledgment wait queue for the correlation with the reply message sent from the receiving application.

[15] The message sent is a reply message. As the reply message needs not to wait for any report message, it is routed to the dummy queue DSLMRDMY. Routing to a dummy queue deletes the reply message, and processing stops at the END label.

[16] The message forwarded is a datagram or request message. The message is routed to the target queue DSLMRSNT, and processing stops at the END label.

[17] A datagram or request message was received. If MSGTYP contains 8 as the first character, the message received is a datagram. Processing continues at the RCVDQ label.

[18] As it is not a datagram, the message received is a request message. It is routed to the target queues DSLMRSQ2 and DSLMRRQ1, and processing stops at the END label.

MERVA-MQI Attachment uses the queue DSLMRSQ2 as a send queue in another send process. The send queue name must be specified in the ALLSNDQ parameter of the DSLKPROC macro. When a received request message is routed to a send queue and the next processing step is specified or defaults to NEXT=STANDARD in the other send process, MERVA-MQI Attachment creates a reply message and sends it to the originator of the request message.

[19] The message received is a datagram. It is routed to the target queue DSLMRRQ1, and processing stops at the END label.

[20] A reply message was received. If STATUS contains the string `'ACK  '`, the waiting request message was correlated with the reply message. Processing continues at the ACKCORR label.

[21] If STATUS contains ACKNC, the received reply message could not be correlated with the waiting request message. Processing continues at the ACKNCORR label.

[22] As STATUS does not contain either of the strings `'ACK  '` or `'ACKNC'`, it must contain either ACKC or ACKER:

- ACKC indicates that the received reply message could be correlated with the waiting request message. The reply message is routed to the dummy queue DSLMRDMY. Routing to a dummy queue deletes the reply message, and processing stops at the END label.
- The processing logic for the status ACKER is not shown in this example.

[23] The correlated request message containing the reply message data is routed to the target queue DSLMRRQ1, and processing stops at the END label.

[24] The reply message which could not be correlated with the waiting request message is routed to the target queue DSLMRNCO, and processing stops at the END label.

[25] A COA report message was received. If STATUS contains the string 'COA ', the waiting datagram, request, or reply message was correlated with the COA report message. Processing continues at the COACORR label.

[26] If STATUS contains COANC, the received COA report message could not be correlated with the waiting datagram, request, or reply message. Processing continues at the COANCORR label.

[27] As STATUS does not contain either of the strings `'COA '` or `'COANC'`, it must contain either COAC or COAER:

- COAC indicates that the received COA report message could be correlated with the waiting datagram, request, or reply message. The COA report message is routed to the dummy queue DSLMRDMY. Routing to a dummy queue deletes the COA report message, and processing stops at the END label.
- The processing logic for the status COAER is not shown in this example.

[28] The datagram, request, or reply message was correlated with the COA report message and contains now the feedback or reason code from the report message. The field DSLKMAWQ is defined as ACKWQ for using its contents in later DSLROUTE macros. ACKWQ contains the name of the acknowledgment wait queue as defined in the ACKWQ parameter of the DSLKPROC macro. If the field DSLKMAWQ is found in the TOF, processing continues at the COAAWQ label.

[29] As the field DSLKMAWQ is not available in the TOF, the correlated message must be either a datagram or a reply message. The correlated datagram or reply message containing the feedback or reason code from the COA report message is routed to the target queue DSLMRRQ1, and processing stops at the END label.

[30] As the field DSLKMAWQ is available in the TOF, the correlated message must be a request message. The correlated request message containing the feedback or reason code from the COA report message is routed to the acknowledgment wait queue whose name is contained in ACKWQ, and processing stops at the END label. The request message waits in the acknowledgment wait queue for the correlation with the reply message sent from the receiving application.

[31] The COA report message which could not be correlated with the waiting datagram, request, or reply message is routed to the target queue DSLMRNCO, and processing stops at the END label.

Writing a User Exit

MERVA-MQI Attachment calls a user exit to add data to a datagram, request, or reply message. The data cannot be provided by MERVA-MQI Attachment itself. The effect of the user exit data on the three MQI message types was described in "Defining the Message Data Structure" on page 302.

Functions of the User Exit

A message in the TOF is presented to the user exit. The exit reads data from, and writes new data to, the TOF thus modifying the message.

The Request Types

The exit is called for a send and receive process. When called for a send process, the exit processes the following requests of MERVA-MQI Attachment:

PUTDATA The exit writes additional message data to the TOF field DSLKDATA. This request applies to a datagram and request message.

On return, MERVA-MQI Attachment reads the data from the field DSLKDATA and creates a message structure as shown in Figure 138 on page 302.

PUTREPLY The exit writes the complete message data to the TOF field DSLKRPLY. This request applies to a reply message.

On return, MERVA-MQI Attachment reads the data from the field DSLKRPLY and creates a message structure as shown in Figure 139 on page 302 or Figure 140 on page 302.

When called for a receive process, the exit processes the following requests of MERVA-MQI Attachment:

GETDATA The exit reads additional message data from the TOF field DSLKDATA and writes it to one or more target fields in the TOF. This request applies to a datagram and request message.

GETREPLY The exit reads the complete message data from the TOF field DSLKRPLY and writes it to one or more target fields in the TOF. This request applies to a request message after correlation with the received reply message.

A single user exit can process these four requests. It is not necessary to write an exit for a send process and another one for a receive process.

The TOF Fields DSLKDATA and DSLKRPLY

Both fields can consist of up to 16 data areas. Each data area can contain data with a length of up to 28680 bytes. Note that the number of data areas can be increased in the MERVA ESA field definition table DSLFDTT. The maximum length of a data area, however, must not be increased.

A data area consists of the area for the field identifier followed by the area for the field data. The area for the field identifier has a fixed length of 8 bytes. The area for the field data has a variable length of up to 28672 bytes. The field identifier is the name of the TOF field that contained the field data. For the request types PUTDATA and PUTREPLY, the exit provides the contents of the field identifier and field data. The field identifier can be set entirely to blanks if the origin of the field data is not relevant.

Interface to MERVA ESA and to MERVA-MQI Attachment

An exit can be written as either:

An HLL MFS user exit
High-level language (HLL) MFS user exits exchange data with MERVA ESA via the MERVA ESA application programming interface (API). They can be written in Assembler, C/370, COBOL, or PL/I. An HLL user exit can use all API functions except INIT and TERM. If written in Assembler, an HLL user exit can use MERVA ESA macros directly, although this is not recommended.

A macro-level MFS user exit
Macro-level MFS user exits use MERVA ESA macros directly. They can be written in Assembler only.

When running under CICS, either type of user exit can issue EXEC CICS calls if required.

Interface to MERVA ESA

The exit is defined to MERVA ESA using the MFS program table DSLMPTT. The following parameters of the DSLMPT macro are used:

LANG The programming language (omit this parameter for a macro-level user exit)

LINK Always LINK=NO

NAME The name of the exit

NUMBER The exit number of up to 5 digits

TYPE Always TYPE=U

The DSLMPTT must be link-edited to the module DSLMMFS. The exit must be link-edited as a separate load module. Under CICS, a program definition for the exit in the CICS system definition file CSD is also required.

Interface to MERVA-MQI Attachment

The exit is defined to MERVA-MQI Attachment using the process table DSLKPROC. Parameter EXIT of the DSLKPROC macro must contain the exit number of up to 5 digits. MERVA-MQI Attachment passes a parameter list to the user exit:

- The parameter list passed to an HLL exit contains the following parameters (in this order):
  - The address of the API interface working storage INTWSTOR
  - The address of the MFS parameter list

  For more information about this parameter list, refer to the *MERVA for ESA Application Programming Interface Guide*.
- The parameter list passed to a macro-level exit follows the conventions for the macro

  ```
  DSLMFS MF=START,TYPE=USER,...
  ```

  For more information about this parameter list, refer to the *MERVA for ESA System Programming Guide*.

The User Exit Control Block (KQCBLOCK): The field **MFSLFLD** in the MFS parameter list contains the address of the user exit control block KQCBLOCK. This control block contains the following fields, which are input fields for the exit:

KQREQID Request identifier. Length is 8 bytes.

This field contains one of the following values:

- GETDATA
- GETREPLY
- PUTDATA
- PUTREPLY

KQMRSNDQ Name of the MERVA ESA send queue. Length is 8 bytes.

For the request identifiers GETDATA and GETREPLY, this field contains blanks.

KQMQUEUE Name of the MQI send or receive queue. Length is 48 bytes.

For the request identifier PUTREPLY, this field contains blanks.

KQTOFBUF Address of the TOF field buffer. Length is 4 bytes.

KQKPROC Address of the process table DSLKPROC. Length is 4 bytes.

KQKENTRY Address of the send or receive process entry in table DSLKPROC. Length is 4 bytes.

The exit must address the following data areas in this order:

1. For an HLL exit, the API interface working storage INTWSTOR using the first address on the passed parameter list.
2. For an HLL exit, the MFS parameter list using the second address on the passed parameter list.
3. The user exit control block KQCBLOCK using the field MFSLFLD of the MFS parameter list.
4. The TOF field buffer KQTOFBFF using the field KQTOFBUF of the user exit control block.
5. The process table DSLKPROC using the field KQKPROC of the user exit control block.

   Addressing the process table is optional and depends on the requirements for the exit.
6. The send or receive process entry in table DSLKPROC using the field KQKENTRY of the user exit control block.

   Addressing the process table entry is optional and depends on the requirements for the exit.

After all required data areas have been addressed, the exit can check the request identifier contained in the field KQREQID of the user exit control block. The request identifier determines the processing of the user exit as described in "The Request Types" on page 332.

The exit uses the TOF field buffer KQTOFBFF when it reads data from, or writes data to, the TOF. The first 8 bytes of KQTOFBFF contain the MERVA ESA buffer prefix. The next 8 bytes contain the TOF field identifier. The remaining 28672 bytes provide the space for the variable length TOF field data. Before writing data to the TOF, the exit must set the second length field of the buffer prefix to the correct length. This length field must contain the length of the TOF field data plus 8 (length of the TOF field identifier) plus 4. The exit can decide whether it provides the TOF field identifier, or sets this area to blanks.

Before the exit returns control to MERVA-MQI Attachment, it must set the following fields in the MFS parameter list:

MFSLRET MFS return code. The return code 0 indicates no error.

MFSLREAS MFS reason code. The reason code 0 indicates no error.

If the exit sets MFSLRET not equal to 0, MERVA-MQI Attachment checks the contents of the MFS error message buffer in field MFSPEMSG. If the exit cleared the buffer to X'00' or set it to blanks, MERVA-MQI Attachment issues an own error message. Otherwise, it issues the error message contained in field MFSPEMSG.

Language-Specific Control Blocks: The following language-specific copy books and macros are available to access the data provided by MERVA-MQI Attachment:

DSLKCBLK User exit control block and TOF field buffer. Available for Assembler, COBOL, and PL/I.

DSLKPROC Process table entry. Available for Assembler (macro), COBOL, and PL/I.

DSLKEXIC User exit control block, TOF field buffer, and process table entry. Available for C/370.

The copy books and macros are included in the user exit as follows:

Assembler
```
COPY DSLKCBLK
DSLKPROC DSLKPROC TYPE=DSECT
```

C/370
```
#INCLUDE "DSLKEXIC.H"
```
(specify immediately following the #INCLUDE for the DSLAPC copy book)

COBOL
```
COPY DSLKCBLK.
COPY DSLKPROC.
```
(specify in the linkage section)

PL/I
```
%INCLUDE DSLKCBLK;
%INCLUDE DSLKPROC;
```

Sample User Exits

The following sample HLL user exits are provided:

- DSLKQ001
- DSLKQ002
- DSLKQ100

The user exits show how the described control blocks and data areas can be addressed. They also show how the MERVA ESA API can be used to read data from, and to write data to, the TOF.

The first two exits demonstrate the principal setup for such a user written program. They are optional programs. The third exit, DSLKQ100, is required when MERVA-MQI Attachment sends and receives telex messages using the format code P. Refer to “Defining the Groups of MERVA ESA Messages” on page 303.

The sample user exits are written in Assembler language. Their source code is available.

Converting the Message Data

When messages are exchanged between applications on different platforms, it is often necessary to convert the message data. Otherwise, the receiving application cannot interpret and process the message contents correctly. Conversion applies both to character and binary data.

MERVA-MQI Attachment provides message conversion programs. Each is written as an exit program to be invoked by MQSeries for MVS/ESA or MERVA-MQI Attachment:

- MQI Data-Conversion Exit (MVS)
- Attachment-Conversion Exit (VSE)

When MERVA-MQI Attachment is running under MVS, you can use the data-conversion exit. For this exit type, MQSeries for MVS/ESA supports the conversion of character data for a lot of code pages. The conversion from or to double-byte character sets (DBCS) is also supported.

When MERVA-MQI Attachment is running under VSE, you can use the attachment-conversion exit. The attachment-conversion exit is always invoked by MERVA-MQI Attachment, even for the conversion of messages before they are sent. Under MQSeries for MVS/ESA, the conversion of messages before being sent is controlled by a message channel agent (MCA) which invokes the data-conversion exit.

Depending on your environment and preferences, refer to "Data-Conversion Exit (MVS)", or "Attachment-Conversion Exit (VSE)" on page 338.

Data-Conversion Exit (MVS)

When MQSeries for MVS/ESA recognizes during the processing of an MQGET call that a data-conversion exit is to be invoked, it checks whether the message consists of character data only. If so, MQSeries converts the message without using the data-conversion exit. That is, a data-conversion exit always converts messages consisting of characters and binary length fields.

The exit can be activated using control information associated with the message. The name of the exit must be equal to the format name in the MQI message descriptor MQMD. This applies both to converting at the sending and receiving side.

See the *MQSeries Application Programming Guide* and the *MQSeries Application Programming Reference* for details on data-conversion exits.

Accessing the Data-Conversion Exit

There are two functionally equivalent modules. The source code for each module is available. The modules are used as follows:

DSLKCDCC Under CICS/ESA, it converts received messages

DSLKCDCM It does the following:

- Under IMS, it converts received messages
- Under both CICS/ESA and IMS, for DQM without CICS, it converts messages to be sent

The modules reside in the following libraries:

- Under CICS/ESA, DSLKCDCC must be contained in a library in the DFHRPL concatenation of the startup job. A program definition specifying **EXECKEY(CICS)** in the CICS system definition file CSD is required.
- Under IMS, DSLKCDCM must be contained in a library in the STEPLIB concatenation of the MPP job.
- For DQM without CICS, DSLKCDCM must be contained in the non-authorized libraries defined by a CSQXLIB DD statement in the JCL of the started task for the MQSeries channel initiator.

Converting Messages at the Receiving Side

Messages should always be converted at the receiving side. The message-sending application needs not to convert the message for the receiving application. This is especially important when the messages pass one or more intermediate nodes (MQSeries installations) before they arrive at the final node.

Messages are converted if the following conditions are met:

- **CONVERT=YES** must be specified or defaulted in the receive process entry in table DSLKPROC.
- The format name in the MQI message descriptor MQMD of the received message must be equal to **DSLKCDCC** (for CICS/ESA) or **DSLKCDCM** (for IMS).

Messages with a format name MQSTR in the MQMD need not be converted by the exit. The format name MQSTR identifies a message consisting of character data only. These messages are converted by MQSeries if CONVERT=YES is specified, either explicitly or by default.

Both the exit and MQSeries carry out the conversion of character data only if the CCSID specified in the received message differs from the receive CCSID. The receive CCSID can be specified using the parameter CCSIDP of the macro DSLKPROC TYPE=RECEIVE. If it is not specified there, the value of the parameter CCSID of the macro DSLKPROC TYPE=INITIAL is used. The exit converts binary integers only if the encoding for the binary length fields in the received message differs from the native encoding 785 for MVS.

The message-sending application is responsible for the correct format name. It should try to set the format name to DSLKCDCC or DSLKCDCM. If this is not possible, link-edit the modules DSLKCDCC or DSLKCDCM using an alias that matches the required format name.

Note: You cannot use a data-conversion exit if the format name consists of blanks.

Converting Messages at the Sending Side

Convert messages at the sending side only when the receiving application is not able to convert the messages.

Under CICS/ESA, if you want to convert messages containing binary length fields, link-edit DSLKCDCM using the alias DSLKCDCC into the library with the CSQXLIB DD statement in the JCL of the started task for the MQSeries channel initiator.

Messages are converted if the following conditions are met:

- **CONVERT(YES)** is specified in the DEFINE CHANNEL command for the sending channel.

- **DSLKCDCM** (for IMS) or **DSLKCDCC** (for CICS/ESA) is specified as the format name or alternate format name in the MQFMT parameter of DSLKPROC TYPE=SEND. If the receiving application expects another format name (except of a blank name), DSLKCDCM can be link-edited using an alias, and this alias must be specified in the MQFMT parameter.

If the message to be sent consists of character data only, MERVA-MQI Attachment sets the format name MQSTR independent of the specification in the MQFMT parameter. Messages with a format name MQSTR in the MQMD need not be converted by the exit. These messages are converted by the MCA if CONVERT(YES) is specified in the DEFINE CHANNEL command.

Both the exit and the MCA carry out the conversion of character data only if the CCSID of the message in the local MQSeries (the send CCSID) differs from the CCSID on the destination platform. The send CCSID can be specified in the parameter CCSIDP of the macro DSLKPROC TYPE=SEND. If it is not specified there, the value specified for CCSID of the macro DSLKPROC TYPE=INITIAL is used. The exit converts binary integers only if the encoding on the destination platform differs from the native encoding 785 for MVS.

Data-Conversion Exit Provided Error Information

When the data-conversion exit detects an error, it provides standard MQI reason codes and exit specific reason codes. For details, refer to *MERVA for ESA Messages and Codes*.

If MERVA-MQI Attachment invokes the exit and the conversion fails, MERVA-MQI Attachment stops processing and inhibits triggering for the MQI receive queue. The unconverted message remains in the MQI receive queue. If the MCA invokes the exit and the conversion fails, the message is sent to the dead-letter queue (undelivered-message queue). If the message cannot be sent to the dead-letter queue, the channel is closed and the unconverted message remains in the transmission queue.

Attachment-Conversion Exit (VSE)

MERVA-MQI Attachment invokes the exit DSLKCVSE when it is running under VSE. DSLKCVSE converts messages that consist of characters only, or of characters and 32-bit binary integers. The characters must be single-byte characters. The exit distinguishes these message categories by checking the contents of the MQI *Format* field (for details refer to "Defining the Message Data Structure" on page 302 and "Recognizing the Message Data Structure" on page 303).

Note: The exit ignores all MQI built-in format names beginning with **MQ**, except for **MQSTR**. When it detects such a built-in format name, the exit immediately returns control to MERVA-MQI Attachment without converting the message data.

The exit DSLKCVSE must be contained in a library of the LIBDEF search chain in the CICS startup job. A program definition for DSLKCVSE in the CICS system definition file CSD is required.

Converting Messages at the Receiving Side

MERVA-MQI Attachment calls the exit at the message-receiving side if CONVERT=YES is specified or defaulted in the appropriate receive process of the DSLKPROC table.

Messages should always be converted at the receiving side. The message-sending application needs not to convert the message for the receiving application. This is especially important when the messages pass one or more intermediate nodes (MQSeries installations) before they arrive at the final node.

The exit converts character data according to the contents of an internal table: the coded character set ID (CCSID) table. The exit converts binary integers to the encoding for VSE, that is, to 785. Table 11 shows the layout of an entry in the internal CCSID table. The table can contain as many entries as required.

Table 11. Internal CCSID Table Entry

Offset Decimal (Hex)	Length in Bytes	Description
0 (0)	4	CCSID of the character data in the received message
4 (4)	8	Name of the conversion table (or blanks)

The exit uses the CCSID specified in the received message as a key to locate, in the CCSID table, the conversion table to be used. The CCSID specified in the received message must have a corresponding entry in the CCSID table, otherwise an error is returned.

The exit carries out the conversion of character data only if the CCSID specified in the received message differs from the receive CCSID. The receive CCSID can be specified using the parameter CCSIDP of the macro DSLKPROC TYPE=RECEIVE. If it is not specified there, the value of the parameter CCSID of the macro DSLKPROC TYPE=INITIAL is used.

The exit converts binary integers only if the encoding for the binary length fields in the received message differs from the native encoding 785 for VSE. If both the CCSID and the encoding are identical, the exit immediately returns control to MERVA-MQI Attachment without converting the message data.

Input for the attachment-conversion exit: The exit requires the following input to convert messages at the receiving side:

- The **external conversion table**, which describes the conversion of character data from the CCSID specified in the received message to the receive CCSID. An example is the conversion from ASCII to EBCDIC. The sample conversion table DSLKATOE is provided to convert single-byte ASCII character data to the CCSID 500 for EBCDIC.

  The conversion table must be link-edited as a separate phase. Like the exit itself, the conversion table must be contained in a library of the LIBDEF search chain in the CICS startup job. A program definition for the conversion table in the CICS system definition file CSD is required.
- The **internal CCSID table**, which is located at the label CM01TBAE in the source code of the conversion exit. For a new CCSID, another entry must be added to the table. The entry is added preceding the label CM01AEL. After this modification, the conversion exit must be assembled and relink-edited.

  The following example shows the internal CCSID table in the sample conversion exit. The table contains the following entries:

```
CM01TBAE DS    0F                 CCSID TABLE
         DC    A(437),CL8'DSLKATOE'    CCSID 437, CONVERSION TABLE NAME
         DC    A(819),CL8'DSLKATOE'    CCSID 819, CONVERSION TABLE NAME
         DC    A(850),CL8'DSLKATOE'    CCSID 850, CONVERSION TABLE NAME
CM01AEL  EQU   4+8                LENGTH OF TABLE ENTRY
CM01AETL EQU   *-CM01TBAE         TOTAL LENGTH OF CCSID TABLE
```

In this example, the three CCSIDs 437, 819, and 850 are associated with the same conversion table, the sample table DSLKATOE. Modify or extend the CCSID table according to your needs.

Converting Messages at the Sending Side

MERVA-MQI Attachment calls the exit at the message-sending side if the CNVDEST parameter of DSLKPROC TYPE=SEND specifies the name of the destination platform.

Convert messages at the sending side only when the receiving application is not able to convert them. The exit converts both character data and binary integers according to the contents of an internal table, the destination table. Table 12 shows the layout of an entry in the internal destination table. The table can contain as many entries as required.

Table 12. Internal Destination Table Entry

Offset Decimal (Hex)	Length in Bytes	Description
0 (0)	8	Name of the destination platform.
8 (8)	8	Name of the conversion table.
16 (10)	4	CCSID of the character data on the destination platform.
20 (14)	4	Encoding for the binary integers on the destination platform.

The value of the CNVDEST parameter provides the key for the destination table. CNVDEST must contain the name of the destination platform. When the exit finds the name of the destination platform in a table entry, the associated table entry items describe the conversion of characters and binary integers for the destination platform.

The encoding for the binary integers on the destination platform indicates to the exit whether the bytes of the integers are arranged in normal (unchanged) order or have to be arranged in reverse order.

When the exit does not find the name of the destination platform in the destination table, it assumes that the destination platform is an IBM RS/6000 running the operating system AIX. Then the following defaults apply for the table entry items:

DSLKETOA Name of the conversion table.

850 CCSID of the character data on the destination platform.

273 Encoding for the binary integers on the destination platform.

A partial conversion can be requested when the CCSID of the character data on the destination platform is set to 0. This means that the character data is not converted. Only the binary length fields will be rearranged if the encoding for the binary integers on the destination platform requires it. This simplifies the conversion at the message-receiving side. Only character data has to be converted there using the correct length information.

Input for the attachment-conversion exit: The exit requires the following input to convert messages at the sending side:

- The **CNVDEST parameter value**, which is provided when the DSLKPROC table is defined. The CNVDEST parameter value must specify the name of the destination platform. The exit uses this name to search in the internal destination table for the appropriate table entry.
- The **external conversion table**, which describes the conversion of character data from the CCSID of the message in the local MQSeries (the send CCSID) to the CCSID on the destination platform. An example is the conversion from EBCDIC to ASCII: the sample conversion table DSLKETOA is provided to convert single-byte EBCDIC character data to the CCSID 850 for ASCII.

  The send CCSID can be specified in the parameter CCSIDP of the macro DSLKPROC TYPE=SEND. If it is not specified there, the value specified for CCSID of the macro DSLKPROC TYPE=INITIAL is used.

  The conversion table must be link-edited as a separate phase. Like the exit itself, the conversion table must be contained in a library of the LIBDEF search chain in the CICS startup job. A program definition for the conversion table in the CICS system definition file CSD is required.
- The **internal destination table**, which is located at the label CM01TBEA in the source code of the conversion exit. When a new destination requires new conversion parameters, another entry must be added to the table. The entry is added preceding the label CM01EAL. After this modification, the conversion exit must be assembled and relink-edited.

  The following example shows the internal destination table in the sample conversion exit. The table contains the following entries:

```
CM01TBEA DS    0F                  DESTINATION TABLE
         DC    CL8'AIX     ',CL8'DSLKETOA',A(850),A(273) BINARY & CHAR.
         DC    CL8'AIXBIN  ',CL8'        ',A(0),A(273)   BINARY ONLY
         DC    CL8'OS2     ',CL8'DSLKETOA',A(850),A(546) BINARY & CHAR.
         DC    CL8'OS2BIN  ',CL8'        ',A(0),A(546)   BINARY ONLY
         DC    CL8'WINNT   ',CL8'DSLKETOA',A(850),A(546) BINARY & CHAR.
         DC    CL8'WINNTBIN',CL8'        ',A(0),A(546)   BINARY ONLY
CM01EAL  EQU   8+8+4+4             LENGTH OF TABLE ENTRY
CM01EATL EQU   *-CM01TBEA          TOTAL LENGTH OF DESTINATION TABLE
```

  The first entry shows the conversion parameters for the destination platform name AIX. They correspond to the default values when the exit cannot find the name of the destination platform in the table.

  The third entry shows the conversion parameters for the destination platform name OS2. This name represents an IBM workstation running the operating system OS/2. The name of the conversion table (sample table DSLKETOA) and the CCSID 850 are the same as for the destination platform name AIX. The conversion parameter for binary integers is different, the encoding is 546.

  The fifth entry shows the conversion parameters for the destination platform name WINNT. This name represents an IBM workstation running the operating system Windows NT. The conversion parameters are the same as for the destination OS2.

  The second, fourth, and sixth entries show the conversion parameters for a partial conversion. This is indicated by the names AIXBIN, OS2BIN, and WINNTBIN, respectively. The name of the conversion table is set to blanks and the target CCSID is set to 0. Only the target encoding is required. Depending on the destination platform, it is 273 or 546.

Modify or extend the destination table according to your needs.

Attachment-Conversion Exit Provided Error Information

When the exit detects an error, it provides a return code for the calling attachment. MERVA-MQI Attachment issues an error message containing this return code.

If the conversion fails, MERVA-MQI Attachment stops processing unless an MQI error queue is specified in the MQIERRQ parameter of DSLKPROC TYPE=RECEIVE. If an error queue (including the dead-letter queue) is available, MERVA-MQI Attachment sets a reason code in the appropriate field of the dead-letter queue header MQDLH, puts the unconverted received message including the MQDLH to the error queue, and continues processing.

For an explanation of the return and reason codes, refer to *MERVA for ESA Messages and Codes.*

Part 2. Defining Fields and Messages

The main purpose of MERVA ESA is to process messages for external networks such as SWIFT or telex, for display and printer terminals, and for system printers. To set the various formats for these external devices, all messages are processed in a MERVA ESA internal format called Tokenized Format (TOF). TOF assumes that a message consists of fields in a predefined order. TOF allows access to any field in a message directly by using a symbolic name. It also allows the transformation of a message to one of the external formats.

The fields of a message, their order in a message, and their presentation on external devices are defined using MERVA ESA macros.

A field definition specifies the attributes of a field such as length and position. The field definitions are contained in the Field Definition Table (FDT).

A message definition specifies the sequence of fields in a message and how these fields must be presented on the external device. For example it specifies if there are separators between fields on an external network, or if there must be additional text on screens or printers. The message definitions are contained in a message control block (MCB).

Both fields and messages are defined by using the assembler macros supplied with MERVA ESA.

Chapter 10. Message Control Blocks (MCBs)

The layout of each message type is described in a separate message control block (MCB). The different message types are identified by a message type indicator which can be up to seven digits long. SWIFT messages use a 3-digit identifier. For example, SWIFT MT 100 is a customer transfer message. The SWIFT MT 100 is used in the examples of this chapter to show how to code the MCBs.

A message control block can refer to other MCBs which describe a part of a message. This means that parts of messages that are similar for different message types can be defined in a separate MCB and referred to by the MCB for a message type.

A message (for example, a SWIFT message) that is displayed on on a terminal screen, or is printed on a terminal printer or system printer, consists of the following parts:

1. The top frame (DSL0TOP), which shows information such as:
   - Message type “MT S100”
   - Message title “Customer Transfer”
   - Page number “Page 00001”
   - Processing Function “Func L1DE0”
2. The message part. For example, for a SWIFT message the message part consists of:
   - The message header
   - Message text containing several message fields dependent on the message type
   - Trailer field

   The message part shows as many fields as is possible on the lines that are left by the top and bottom frames. Fields that do not fit on one page can be seen on the next or following pages.
3. The bottom frame (DSL0BOT), which shows information such as:
   - The error message line
   - The command line
   - The PF key information lines

These parts are illustrated in Figure 148 on page 354.

All three parts of a message panel must be defined by the Message Definition Facility.

General Message Control Block Structure

Each message control block (MCB) can contain one or more device descriptions. There are five different device types available in an MCB, and they can all be described in the same MCB. The device types available are:

- TYPE=MESSAGE shows the sequence of fields in a message type and their reference in the TOF. It must be present in the first MCB that is used to process

the message. When this device type description is used it should be at the beginning of the MCB. The message description can be omitted in the following cases:

  – In MCBs that are referred to in another MCB
  – In an MCB that uses only MERVA ESA system fields

- TYPE=SCREEN shows the presentation of a message on a terminal screen. There can be several screen descriptions using different layouts or languages in the same MCB.
- TYPE=HARDCOPY shows the presentation of a message on a terminal printer. There can be several hardcopy descriptions using different layouts or languages in the same MCB. The hardcopy description can also refer to a screen description in the same MCB if the layout and language used is the same.
- TYPE=SYSP shows the presentation of a message on a system printer. There can be several system printer descriptions using different layouts or languages in the same MCB. The system printer description can also refer to a screen or hardcopy description in the same MCB if the layout and language used is the same.
- TYPE=NET shows the presentation of a message on an external network line (such as the SWIFT network, ID=W), or on a screen in NOPROMPT mode (ID=X in MERVA ESA). The net format is also used for the sequential data sets processed by the MERVA ESA batch programs DSLSDI and DSLSDO.

The Message Definition Macroinstructions

The message definition macros are all coded using the rules of Assembler described in the *High Level Assembler Language Reference.* The output produced by the Assembler is the control block as defined by the message definition macros. If errors are detected in the input, the control block produced is incomplete and should be discarded. An error message indicating the cause of any error is printed in the output listing.

The macros used to define messages are described in the *MERVA for ESA Macro Reference*.

DSLLMCB
Identifies the beginning of a message control block (MCB) and gives it a name.

DSLLDEV
Identifies the device type and operational options.

DSLLGRP
Defines the start of a logical group of fields.

DSLLUNIT
Defines the beginning of a unit of message fields.

DSLLMFLD, DSLLDFLD, DSLLNFLD
Define a message field, display device field, and net field, respectively. Here, DSLLxFLD is used to show that all three types of field definition macros are being referred to.

DSLLUEND
Defines the end of a unit.

DSLLGEN
Shows the end of a message control block and completes the MCB generation.

The following list shows the hierarchical structure of an MCB:

- Device level

These are the definitions starting with a DSLLDEV macro and ending with another DSLLDEV macro or a DSLLGEN macro.

- Group level

  These are all the definitions starting with a DSLLGRP macro and ending with another DSLLGRP, DSLLDEV, or a DSLLGEN macro. If no DSLLGRP macro is defined for a device, the group level is the same as the device level.

- Unit level

  These are all the definitions starting with a DSLLUNIT macro and ending with a DSLLUEND, DSLLGRP, DSLLDEV, or DSLLGEN macro. If no DSLLUNIT is defined within a group, the unit level is the same as the group level. If no DSLLUNIT and no DSLLGRP macros are defined for a device, the unit level is the same as the device level.

- Field level

  These are all the DSLLxFLD definitions.

The following list shows the macros that control the flow of processing within the same MCB or to other MCBs:

DSLLCOND
: Specifies the conditions under which processing is to continue, and at which point in the MCB it is to continue.

DSLLEXIT
: Imbeds another MCB identified explicitly by a message identification or implicitly by an exit field into the processing of a message for the same device.

MCB Coding Examples

The following sections give examples on how to code the MCB for all device types. The SWIFT MT 100 (Customer Transfer) is used in the examples of the coding (Module DWS100).

Example for TYPE=MESSAGE

Figure 142 on page 348 shows how to code an MCB for TYPE=MESSAGE. The statements marked by the numbers in parentheses are commented after the figure.

```
         TITLE    'M T   1 0 0   -   CUSTOMER TRANSFER'          [1]
         COPY     DSLCOLOR                                       [2]
DWS100   DSLLMCB                                                 [3]
MESSAGE  DSLLDEV  TYPE=MESSAGE                                   [4]
         COPY     DWSMHEAD                                       [5]
GRN005   DSLLGRP  GRPNUM=5                                       [6]
SW20     DSLLMFLD MAND=YES                                       [7]
SW32     DSLLMFLD MAND=YES,OPTLIST=(A), Expand=1003              [8]
SW50     DSLLMFLD MAND=YES
SW52     DSLLMFLD OPTLIST=(A,D)                                  [9]
SA52     DSLLMFLD                                               [10]
SW53     DSLLMFLD OPTLIST=(A,B,D)
SA53     DSLLMFLD
SW54     DSLLMFLD OPTLIST=(A,B,D)
SA54     DSLLMFLD
SW56     DSLLMFLD OPTLIST=(A),EXPAND=1003                       [11]
SA56     DSLLMFLD
SW57     DSLLMFLD OPTLIST=(A,B,D)
SA57     DSLLMFLD
SW59     DSLLMFLD MAND=YES
SW70     DSLLMFLD
SW71     DSLLMFLD OPTLIST=(A),DAMAX=1,LENGTH=(3,,F),EXPAND=1003 [12]
SW72     DSLLMFLD
         COPY     DWSMTRL                                        [13]
```

Figure 142. MCB of SWIFT MT 100, TYPE=MESSAGE

Notes:

[1] The assembler TITLE statement.

[2] The assembler COPY statement and includes the color definitions (see Figure 145 on page 350) used for screen display.

[3] DSLLMCB

This is the first macro statement in the message control block. It starts the MCB, and its label DWS100 is used as the MCB name. DSLLMCB can be used only once in an MCB.

[4] DSLLDEV

Defines the device type. TYPE=MESSAGE can be used only once in an MCB and must be specified as the first device if used.

[5] This is the assembler COPY statement and includes the message header DWSMHEAD (see Figure 143 on page 349).

[6] DSLLGRP

The DSLLGRP macro assigns a unique name, GRN005, to the group. The group number GRPNUM (1 to 255) must be unique and in ascending order. The group GRN005 is the first group of the Message Text and has the group number 5 (GRPNUM=5) randomly chosen (GRPNUM=1 is reserved for the message header). The following groups are numbered automatically.

This group number is defined for later reference in another device description.

[7] DSLLMFLD

This defines the SWIFT field 20 (transaction reference number) that was given the name SW20 in the Field Definition Table. MAND=YES shows that this field is mandatory for MT 100.

[8] OPTLIST=(A)

In MT 100 only the option 'A' is allowed for field SW32. In different message types, other options can be valid with SWIFT field 32, such as 'B','M','S'.

[9] OPTLIST=(A,D)

The field SW52 can have the option A or D.

[10] SA52

This shows that a SWIFT address in field SW52 can be expanded to the correspondent name in the field SA52 if a MERVA ESA function is processed that specifies address expansion in the function table DSLFNTT.

[11] EXPAND=1003

Specifies the SWIFT Link expansion routine 1003. This routine sets or removes the option for optional fields having only one option specified depending on whether other data for this field were supplied.

[12] DAMAX=1,LENGTH=(3,,F)

This length specification differs from the general definition of field 71 in the Field Definition Table; therefore the appropriate definition for field 71 in MT 100 must be specified in the MCB.

DAMAX and LENGTH override the general definition of field 71 in the Field Definition Table. For example, in MT 100, only one data area with a fixed length of 3 characters is allowed.

[13] This is the Assembler COPY statement and includes the message trailer (see Figure 144 on page 350).

Note: The device TYPE=MESSAGE is ended by the next DSLLDEV macro (see Figure 146 on page 351).

Copies Included in the Sample MCB MT 100 for TYPE=MESSAGE

In the following, the two copies included in the TYPE=MESSAGE part of the MCB for MT 100 are presented. The examples refer to the message header (COPY DWSMHEAD) and the message trailer (COPY DWSMTRL).

The use of such copies for general parts of a message simplifies the MCB definitions and makes changing general message parts easier later on. Then only the copies are changed and all the related MCBs are assembled. This is more efficient than changing all the related MCBs and assembling them.

```
         DSLLCOND O1=(TEST=DSLCOND),EQ=NO,O2='NO',GOTO=MSG10     [1]
         DSLLEXIT IMBED=SHEAD                                    [2]
MSG10    DSLLCOND                                                [3]
```

Figure 143. Example of the Message Header Copy for TYPE=MESSAGE

Notes:

[1] DSLLCOND

This checks a condition during execution time. The first operand O1 is compared with the second operand O2 by the relational operator EQ (equal). If the result of the comparison is true, then the code branches to the label MSG10. Otherwise the next line is processed. This condition is

used to omit the normal SWIFT message headers where the MT 100 is nested in another message type (192, 195, or 196).

[2] DSLLEXIT

During execution time, the MCB DWSHEAD is embedded. This contains the field definitions for the header of SWIFT messages. SHEAD is the message identification of this MCB and is expanded into the full MCB name in the MERVA ESA message type table.

[3] DSLLCOND

As no parameters are specified, this DSLLCOND macro is used only to specify the label for statement **[1]**. No condition is evaluated.

```
DSLLCOND O1=(TEST=DSLCOND),EQ=NO,O2='NO',GOTO=SCREEN    [1]
DSLLEXIT IMBED=STRL                                     [2]
```

Figure 144. Example of the Trailer Copy for TYPE=MESSAGE

Notes:

[1] DSLLCOND

As for the condition in the header, the embed of the SWIFT message trailer is bypassed when the message type 100 is embedded in one of the MTs 192, 195 or 196.

[2] DSLLEXIT

The SWIFT message trailer is embedded, DWSTRL is the MCB that defines the trailer field.

Example for Color Definitions

```
          GBLC    &LITERAL                                  [1]
          GBLC    &TITLE
          GBLC    &DATA
          GBLC    &WATCH
          GBLC    &BELL
&LITERAL  SETC    'BLUE'                                    [2]
&TITLE    SETC    'NEUTRAL'
&DATA     SETC    'GREEN'
&WATCH    SETC    'YELLOW'
&BELL     SETC    'RED'
```

Figure 145. Example of the Color Copy

Notes:

[1] GBLC

These instructions define global variables.

[2] SETC

These instructions assign color attributes to the global variables. These color attributes are used for screen display items, for example, literals are displayed in blue, the title is displayed in a neutral color (that is, the default color), normal data-entry fields are displayed with green color,

retyped fields are displayed with yellow color, and error messages (for example, in the MCB DWSERROR) are displayed with red color.

Example for TYPE=SCREEN

In screen device definitions, each field defined by a DSLLDFLD statement is preceded by a screen attribute byte. This should be considered when calculating the position of a field. It is recommended that you start on the screen position line 1, column 2 (POS=(1,2)).

The next free position after a field with a length of x bytes and the position POS=(ll,cc) is calculated as shown below:

```
POS=(ll+(cc+x+1)/80,(cc+x+1)//80)
```

where "/" denotes integer division and "//" the remainder from integer division.

```
SCREEN   DSLLDEV  TYPE=SCREEN,ID=E                          [1]
         COPY     DWSCHEAD
         DSLLGRP  GROUP=GRN005                             [2]
         DWSSW20  MSGTYP=XXX,MANDCH=*                      [3]
         DWSSW32A MSGTYP=XXX,MANDCH=*,ONEOPT=A
         DWSSW50  MSGTYP=XXX,MANDCH=*
         DWSSW52  MSGTYP=111
         DWSSW53  MSGTYP=111
         DWSSW54  MSGTYP=111
         DWSSW56  MSGTYP=111,ONEOPT=A
         DWSSW57  MSGTYP=111
         DWSSW59  MSGTYP=XXX,MANDCH=*
         DWSSW70
         DWSSW71A MSGTYP=XXX,ONEOPT=A                      [4]
         DWSSW72  MSGTYP=XXX
         COPY     DWSCTRL
```

Figure 146. MCB of SWIFT Message Type 100, TYPE=SCREEN

Notes:

[1] DSLLDEV

This describes the device to be used, TYPE=SCREEN. The language identifier E (English) specifies that all users that have the language identifier in their user-file record, or select with the **form** command, use this section of the screen device description.

[2] DSLLGRP

This instruction assigns the following field definitions to the group GRN005 as defined in the TYPE=MESSAGE part.

[3] MANDCH=*

The parameter MANDCH=* specifies the mandatory option for the field macro DWSSW20. This macro defines the literal and field definitions for the SWIFT field 20 (SW20) for screen and printer devices.

[4] DWSSW71A

This calls the macro DWSSW71A that contains the literal and field definitions for the SWIFT field 71 (SW71), for screen and printer devices. The parameters MSGTYP=XXX passes the message type, and ONEOPT=A passes the only allowed option A to the macro DWSSW71A. See Figure 147.

Sample of a Field Macro for Screen and Printer Devices

Figure 147 shows how an assembler macro is coded for a field used in several message types. The example uses the SWIFT field 71 (SW71). The use of such macros simplifies the MCB definitions, especially when a field is changed. Only the macro is changed and all the related MCBs are assembled. This is more efficient than changing all the related MCBs and assembling them.

```
         MACRO                                                          [1]
         DWSSW71A &MSGTYP=,&MANDCH=,&ONEOPT=                            [2]
         GBLC     &LITERAL                                              [3]
         GBLC     &TITLE
         GBLC     &DATA
         GBLC     &WATCH
         GBLC     &BELL
         LCLC     &MANDIND                                              [4]
         AIF      (T'&MANDCH NE 'O').AMAND1
&MANDIND SETC     ' '
         AGO      .AMAND2
. AMAND1  ANOP
&MANDIND SETC     '*'
. AMAND2  ANOP
         DSLLUNIT COMMENT=Y                                             [5]
         AIF      ('&MSGTYP' EQ 'XXX').A071AA                           [6]
         AIF      ('&MSGTYP' EQ '740').A071AB
. A071AA  ANOP
         DSLLDFLD 'Details of Charges  &MANDIND.71',POS=(NEXT,02),    *[7]
               COLOR=&LITERAL,COMMENT=Y
         AGO      .A071AZ
. A071AB  ANOP
         DSLLDFLD 'Reimbursing Charges &MANDIND.71',POS=(NEXT,02),    *
               COLOR=&LITERAL,COMMENT=Y
         AGO      .A071AZ
. A071AZ  ANOP
         AIF      (T'&ONEOPT EQ 'O').ONEOPT1                            [8]
         DSLLDFLD '&ONEOPT',POS=(,NEXT),COLOR=&LITERAL,COMMENT=Y
         AGO      .ONEOPT2
. ONEOPT1 ANOP
           DSLLDFLD FLD=SW71,OPTION=YES,POS=(,NEXT),LENGTH=1,COLOR=&DATA,*;
                COMMENT=Y
         AGO      .ONEOPT2
. ONEOPT2 ANOP
         DSLLDFLD ':',COLOR=&LITERAL,COMMENT=Y
         DSLLDFLD FLD=SW71,LENGTH=3,COLOR=&DATA,COMMENT=Y               [9]
         DSLLUEND COMMENT=Y                                             [5]
         MEND                                                          [10]
```

Figure 147. Example of the Macro for the SWIFT Field SW71A

Notes:

[1] MACRO

The start of the field macro.

[2] DWSSW71A

The name of the macro. It defines the parameters MSGTYP, MANDCH, and ONEOPT.

MSGTYP Shows the message type of the MCB that uses the macro to process message-type dependent information for the field.

MANDCH Shows if the field is mandatory, dependent on the message type. MANDCH=* causes an asterisk to be displayed in front of the field to specify that it is mandatory (see also 4).

ONEOPT Defines the only option for this field specified (see 8).

[3] GBLC &LITERAL

The following instructions define global variables.

[4] &MANDIND

In this and the following lines the variable &MANDIND is either set to '*' or ' ' according to the parameter MANDCH. The variable is used in the literal preceding the field SW71 on the output device. ' ' means the field is optional, '*' means the field is mandatory.

[5] DSLLUNIT ..... DSLLUEND

These enclose a unit of message fields. Each field is enclosed by unit statements to allow for unit compression. The field and its accompanying text (literals) are only displayed when one or more filled data areas exist.

COMMENT=Y

Specifies that the macro is to be shown as MNOTE after macro expansion.

[6] &MSGTYP

In this section the literal text preceding the field is chosen according to the message type in which the macro is used. The variable contains either the MT number for specific literals or 'XXX' when the same literal is used for many message types.

[7] DSLLDFLD

This macro is used for screens, hard-copy, and system printers to define the device input and output fields or literals. The literal 'Details ...' is shown on the next line beginning on column 2, with the color as defined for the global variable &LITERAL.

[8] &ONEOPT

In this section the field option is displayed as a literal, (if ONEOPT is specified in the MCB TYPE=SCREEN), or as an input field to be entered on the screen.

[9] DSLLDFLD

FLD=SW71

The contents of TOF field SW71 are shown on the same line, with a length of 3 bytes (LENGTH=3), beginning in the next column as defined in the macro DSLLDFLD referring to the field option, POS=(,NEXT).

[10] MEND

The macro end.

Sample of a Screen Panel

The following figure shows the result of the previous TYPE=SCREEN definition for SWIFT MT 100.

```
MT S100                    Customer Transfer                    Page 00001
                                                                Func L1DE0
                                                                UMR  00002002
Basic Header        F  01 TIBMBEAAAXXX 0000 000000
Application Header  I 100              N
User Header         Service Code    103:
                    Bank. Priority  113:
                    Msg User Ref.   108:
TRN                 *20   :
Date/Cur/Amount     *32 A : Date        Currency     Amount
Ordering Customer   *50   : _____________________________________
                            _____________________________________
                            _____________________________________
                            _____________________________________
Ordering Inst.       52 _ : / _ / _________________________________
                            _____________________________________
                            _____________________________________
                            _____________________________________
                            _____________________________________




Command =====> ____________________________________________________________
PF 1=Help      2=Retrieve   3=EOM        4=Repeat     5=Get Next   6=Requeue
PF 7=Page -1   8=Page +1    9=Hardcopy  10=Pro Line  11=Prompt    12=Escape
```

Figure 148. SWIFT MT 100 with SWIFT II Input Header, Page 1

Notes:

- The first two lines 'MT ..... Func L1DE0' are created by the MCB DSL0TOP (see "The Frame MCBs for Screen and Printer Panels" on page 362).
- The line 'Basic Header...' and the following are created by the macros: DSLLEXIT IMBED=SBHEAD DSLLEXIT IMBED=SAHEAD DSLLEXIT IMBED=SUHEAD

  This is included in the MCB by the copy book DWSCHEAD (see Figure 146 on page 351).
- The line 'TRN ...' is created by the DWSSW20 macro. The lines following the TRN are created by the macros that follow the DWSSW20 macro.
- The line 'Command ...', the previous and the two lines following are created by the MCB DSL0BOT (see "The Frame MCBs for Screen and Printer Panels" on page 362).

Example for TYPE=HARDCOPY

This example shows the coding for a hardcopy device. The example uses the same definitions that were used for the screen terminal, except for the DSLLDEV statement.

```
HARDCOPY DSLLDEV  TYPE=HARDCOPY,ID=E,LIKE=SCREEN
```

The parameter TYPE=HARDCOPY specifies that the layout for a hardcopy printer is to be defined. The LIKE parameter refers to the label SCREEN and specifies the use of the same layout as defined in TYPE=SCREEN.

Example for TYPE=SYSP

This coding example shows the coding example for a system printer device. The example uses the same definitions as the screen terminal.

```
PRINTER  DSLLDEV  TYPE=SYSP,ID=E,LIKE=SCREEN
```

The parameter TYPE=SYSP specifies that the layout for the system printer is defined. The LIKE parameter refers to the label SCREEN and specifies the use of the same layout as defined in TYPE=SCREEN.

Example for the SWIFT Line with TYPE=NET

```
 LINES   DSLLDEV  TYPE=NET,ID=S,SEP=X'0D25'                      [1]
         COPY     DWSSHEAD
         DSLLGRP  GROUP=GRN005
         DSLLNFLD FLD=SW20,TAG=':20:'
         DSLLNFLD FLD=SW32,TAG=':32'-':'
         DSLLNFLD FLD=SW50,TAG=':50:'
         DSLLNFLD FLD=SW52,TAG=':52'-':'
         DSLLNFLD FLD=SW53,TAG=':53'-':'
         DSLLNFLD FLD=SW54,TAG=':54'-':'
         DSLLNFLD FLD=SW56,TAG=':56'-':'
         DSLLNFLD FLD=SW57,TAG=':57'-':'
         DSLLNFLD FLD=SW59,TAG=':59:'
         DSLLNFLD FLD=SW70,TAG=':70:'
         DSLLNFLD FLD=SW71,TAG=':71'-':'                       [2]
         DSLLNFLD FLD=SW72,TAG=':72:'
         COPY     DWSSTRL
LINEW    DSLLDEV  TYPE=NET,ID=W,SEP=X'0D25',LIKE=LINES         [3]
```

Figure 149. MCB of SWIFT MT 100, TYPE=NET, for the SWIFT Line

Notes:

[1] TYPE=NET,ID=S,SEP=X'0D25'

Defines the layout of the message for the communication line to the SWIFT network.

The message formats for SWIFT I are still supported in MERVA ESA for compatibility reasons. However, the SWIFT I network support has been dropped as there is no SWIFT I network any longer. The differences between SWIFT I and SWIFT II network are handled in the copy books DWSSHEAD and DWSSTRL (see comment 3). It can also be used by the MERVA ESA batch programs DSLSDI and DSLSDO.

As this layout is defined by SWIFT, a change of these definitions must only be made when SWIFT requests it.

The operand SEP defines a default character string that most commonly separates the fields for this device definition.

[2] DSLLNFLD

FLD=SW71

Specifies the SWIFT field 71 by its TOF field name SW71.

TAG=':71'-':'

Specifies that the tag for SW71 consists of the character string ':71', the option ('-') and ':'.

[3] DSLLDEV

ID=W

Is the line definition for the SWIFT II format. The parameter LIKE=LINES specifies that the device description for the SWIFT I format should be taken for the mapping process.

Example for the Screen NOPROMPT Mode with TYPE=NET

```
 LINEX   DSLLDEV  TYPE=NET,ID=X,SEP=X'0D25'                      [1]
         COPY     DWSXHEAD
         DSLLGRP  GROUP=GRN005
         DSLLNFLD FLD=SW20,TAG=':20:'
         DSLLNFLD FLD=SW32,TAG=':32'-':'
         DSLLNFLD FLD=SW50,TAG=':50:'
         DSLLNFLD FLD=SW52,TAG=':52'-':'
         DSLLNFLD FLD=SA52,TAG=':SX52:'                        [2]
         DSLLNFLD FLD=SW53,TAG=':53'-':'
         DSLLNFLD FLD=SA53,TAG=':SX53:'
         DSLLNFLD FLD=SW54,TAG=':54'-':'
         DSLLNFLD FLD=SA54,TAG=':SX54:'
         DSLLNFLD FLD=SW56,TAG=':56'-':'
         DSLLNFLD FLD=SA56,TAG=':SX56:'
         DSLLNFLD FLD=SW57,TAG=':57'-':'
         DSLLNFLD FLD=SA57,TAG=':SX57:'
         DSLLNFLD FLD=SW59,TAG=':59:'
         DSLLNFLD FLD=SW70,TAG=':70:'
         DSLLNFLD FLD=SW71,TAG=':71'-':'
         DSLLNFLD FLD=SW72,TAG=':72:'
         COPY     DWSXTRL
LINEY    DSLLDEV  TYPE=NET,ID=Y,SEP=X'0D25',LIKE=LINEX         [3]
GEN      DSLLGEN                                               [4]
         END                                                   [5]
```

Figure 150. MCB of SWIFT MT 100, TYPE=NET, Screen NOPROMPT Mode

Notes:

[1] DSLLDEV

TYPE=NET,ID=X

Defines the layout of the message for the screen terminal in NOPROMPT mode. This format cannot be used for the communication line to the SWIFT network as it contains fields that are not known to SWIFT (see 2.), and it uses some separators that do not conform to the SWIFT specifications.

ID=X is similar to the ID=S except for the address fields SA5x.

[2] SA52

This field is used for address expansion and contains the correspondent name and control information for the SWIFT address of the SWIFT field 52, SW52.

[3] DSLLDEV

ID=Y

Defines the layout for the screen terminal in NOPROMPT mode for the SWIFT II format. The same device description as for the SWIFT I format is used. The differences are handled in the copy books for the SWIFT header and trailer (DWSXHEAD and DWSXTRL).

[4] DSLLGEN

This macro ends the message definitions and completes the message control block generation.

Note: As it is the last device description for MT 100, the statements indicating the end of the MCB are shown here.

[5] END

This Assembler statement must follow immediately and must be the last statement.

Examples for the SWIFT Message Trailer with TYPE=NET

```
LINEX   DSLLDEV  TYPE=NET,ID=X,SEP=X'0D25'
        DSLLGRP  GROUP=GRPTRAIL
        DSLLCOND INPUT,01=BUFFER,EQ=NO,02=':5:',GOTO=LINEX9            [1]
        DSLLNFLD FLD=SW00BL5,TAG=(':5:',ALWAYS),SEP=(X'0D25',ALWAYS)   [2]
        DSLLNFLD FLD=SWTRAIL      NO TAG, ACCEPT ANYTHING AFTER :5:
LINEX9  DSLLCOND
LINEY   DSLLDEV  TYPE=NET,ID=Y,SEP=X'0D25',LIKE=LINEX                  [3]
```

Figure 151. Example of the Extended SWIFT Message Trailer for TYPE=NET, Screen NOPROMPT Mode for the SWIFT II Network

This definition is part of the MCB DWSTRAIL for SWIFT II network trailer mapping.

Notes:

[1] DSLLCOND

The defined tag for a NOPROMPT trailer is ':5:'. When this sequence is not in the input buffer, the trailer processing is skipped.

[2] DSLLNFLD FLD=SW00BL5,TAG=(':5:',ALWAYS),SEP=(X'0D25',ALWAYS)

This is the tag ':5:' and an internal field SW00BL5 that allows input for this tag line in NOPROMPT mode.

[3] DSLLDEV TYPE=NET,ID=Y

The external line format Y used to indicate the SWIFT II format. In this case both formats X and Y map for the SWIFT II network. The selection that is to be mapped for both networks is defined in the copy book DWSXTRL.

```
LINEX   DSLLDEV  TYPE=NET,ID=X,SEP=X'0D25'
        DSLLGRP  GROUP=GRPTRL
        DSLLNFLD FLD=SWTRL,TAG=('-',ALWAYS),SEP=(X'0D25',ALWAYS)  [1]
        DSLLNFLD FLD=MSGACK1,NI=0,GROUP=1,TAG=':AK:'              [2]
        DSLLNFLD FLD=MSGTRACE,NI=0,GROUP=1,TAG=':TR:'             [3]
```

Figure 152. Example of the Extended SWIFT Message Trailer for TYPE=NET, Screen NOPROMPT Mode

In this example, the SWIFT message trailer is extended by the two fields MSGACK1 and MSGTRACE. These fields are added after the SWIFT trailer. When these fields are found in the TOF, their data is displayed in NOPROMPT mode. The same technique can be used to map user fields to a sequential output data set using the batch program DSLSDO, or from a sequential input data set into a MERVA ESA queue using the batch program DSLSDI.

Notes:

[1] DSLLNFLD

FLD=SWTRL

The message trailer field is mapped into the buffer or displayed on a terminal or printer in NOPROMPT mode.

TAG=('-',ALWAYS),SEP=(X'0D25',ALWAYS)

When the trailer field is empty, the tag and the separator are mapped or displayed regardless. This is necessary so that the following fields are not appended to the trailer.

[2] DSLLNFLD

FLD=MSGACK1

The MSGACK1 field (SWIFT acknowledgment in SWIFT I format or the SWIFT Link diagnostic message) is mapped into the buffer or displayed on a terminal or printer in NOPROMPT mode. For example, a SWIFT acknowledgment in SWIFT I format is mapped like this:

```
:AK:ACK/1215/03TIBMBEBBAXXX00012
```

NI=0,GROUP=1

This is the index specification to address the field in the TOF. This specification is necessary to avoid the use of the default values NI=1 and the current group 255 of the SWIFT trailer.

[3] DSLLNFLD

FLD=MSGTRACE

The MSGTRACE field (message trace) is mapped into the buffer or displayed on a terminal or printer in NOPROMPT mode. Each data area of the MSGTRACE field is mapped in a separate line, for example:

```
:TR:USER1   L1DE0   0000930615074803LTERM1
USER2   L1VE0   0000930615081213LTERM2
USER3   L1AI0   0000930615112354LTERM3
```

Description of Functions Not Contained in MT 100

There are functions for MCB coding not used in MT 100. One function refers to the MCB coding of repeatable sequences, such as MT 210. Figure 153 on page 359 shows MCB macros for a repeatable sequence.

```
         TITLE    'M T   2 1 0   -   NOTICE TO RECEIVE'

⋮

MESSAGE  DSLLDEV  TYPE=MESSAGE

⋮

GRN006   DSLLGRP                                                    [1]
         DSLLUNIT REPSEQ=(1,10)                                     [2]

⋮

         DSLLUEND                                                   [2]

⋮

SCREEN   DSLLDEV  TYPE=SCREEN,ID=E

⋮

         DSLLGRP  GROUP=GRN006                                      [3]
         DSLLCOND PAGE=NEW,LINES=17                                 [4]
         DSLLUNIT REPSEQ=(1,10)                                     [5]
         DWSSWREP                                                   [6]
         DWSSW21  MSGTYP=XXX,MANDCH=*
         DWSSW32B MSGTYP=XXX,MANDCH=*,ONEOPT=B
         DWSSW50  MSGTYP=XXX
         DWSSW52  MSGTYP=III
         DWSSW56  MSGTYP=III
         DSLLCOND PAGE=NEW,LINES=17                                 [4]
         DSLLUEND                                                   [5]

⋮

GEN      DSLLGEN
         END
```

Figure 153. MCB Macroinstructions for a Repeatable Sequence

Notes:

[1] GRN006 DSLLGRP

The DSLLGRP macro assigns the name GRN006 to the next field group.

[2] REPSEQ=(1,10)

This describes a repeatable sequence within a field group. All the fields between DSLLUNIT and DSLLUEND can be repeated from 1 to 10 times.

[3] DSLLGRP GROUP=GRN006

This refers to the DSLLGRP macro in TYPE=MESSAGE (label name GRN006).

[4] PAGE=NEW

Displays the following fields beginning on the next page of the device if less than 17 lines are free at the bottom of the page. This results in a sequence of fields starting on a new page, when the whole sequence does not fit on the current page.

[5] REPSEQ=(1,10)

This is the macro of the DSLLDEV TYPE=SCREEN part corresponding to the macro (02) explained for the DSLLDEV TYPE=MESSAGE part.

[6] DWSSWREP

This calls the macro DWSSWREP that contains the literal definitions for the headers used for repeatable sequences. See Figure 154.

Repeatable Sequence Header Macro for Screen and Printer Devices

```
         MACRO
         DWSSWREP
         GBLC     &LITERAL
         GBLC     &TITLE
         GBLC     &DATA
         GBLC     &WATCH
         GBLC     &BELL
         LCLC     &LB
&LB      SETC     'SWRS'.'&SYSNDX'(2,3)                                  [1]
         DSLLDFLD '* * Repeatable Sequence',POS=(NEXT,2),           *   [2]
               COLOR=&LITERAL,COMMENT=Y
         DSLLDFLD FLD=DSLACTR,NI=0,LENGTH=3,POS=(,NEXT),COLOR=&LITERAL,* [3]
               PROT=YES,RSNUM=1,COMMENT=Y
         DSLLDFLD '* * * * * * * * * * * * * *  Occurrence',         * [2]
               COLOR=&LITERAL,COMMENT=Y
         DSLLCOND O1=(TEST=DSLACTA),EQ=YES,O2='A',GOTO=&LB;A,COMMENT=Y   [4]
         DSLLDFLD FLD=DSLACCO,NI=0,LENGTH=5,POS=(,NEXT),COLOR=&LITERAL,*;
               PROT=YES,RSNUM=1,COMMENT=Y
         DSLLCOND GOTO=&LB;B,COMMENT=Y
&LB.A    DSLLDFLD FLD=DSLACCO,NI=0,LENGTH=5,POS=(,NEXT),COLOR=&TITLE,  *
               PROT=YES,RSNUM=1,DISP=HIGH,COMMENT=Y
&LB.B    DSLLCOND COMMENT=Y
         MEND
```

Figure 154. Coding Example for a Macro Used for Repeatable Sequences

Notes:

[1] SETC

This instruction is used to create a unique label, consisting of 'SWRS' and a 3-digit character string generated for every new macro call.

[2] DSLLDFLD

These instructions create the literal parts of the header of each occurrence in a repeatable sequence.

[3] FLD=DSLACTR

This field shows the number of the repeatable sequence to be displayed.

NI=0

Sets the nesting identifier to 0. This is necessary because DSLACTR is a MERVA ESA system field (see DSLFDTTC).

PROT=YES

This instruction specifies that the field displayed is protected against entering of data.

RSNUM=1

Sets the occurrence number of this field to 1. This is necessary because the occurrence number is increased with each pass through a sequence. DSLACTR is not within the repeatable sequence, but a MERVA ESA system field with only one occurrence.

[4] DSLLCOND

This condition is required to find out whether the occurrence to be displayed is the "active" one on the screen. Inactive occurrences are displayed normally (COLOR=&LITERAL), whereas active occurrences are displayed highlighted (DISP=HIGH for monochrome and extended color feature screen devices, COLOR=&TITLE for normal color screen devices).

Processing New or Changed MCBs

Note: Changing message control blocks or Field Definition Tables can cause a discrepancy between the description in the *MERVA for ESA User's Guide* and your customized installation.

The following steps should be carried out when designing or coding a new message type, changing an existing message type, or when changing or coding function panels:

- Determine the fields that are required for the new message type. See "Chapter 13. Field Definition Table (DSLFDTT)" on page 385.

  Keep in mind that, when new subfields are required in an MCB, they must be specified in the Field Definition Table. If an MCB refers to a subfield not found in the Field Definition Table, the field is not recognized as a subfield. On a screen or printer panel such a field cannot be displayed.

  The sequence of fields of a message must be defined in the message description in the MCB (DSLLMCB TYPE=MESSAGE). Status information already specified in the Field Definition Table need not be specified again in the message description.

  Note: When you change the status information of a field in an MCB, or assign a different exit to the field, this information is used only for fields in messages created after the alteration. To assign new features to fields in old messages, you must map these messages again by reprocessing the message in NOPROMPT mode.
- Define the screen, hardcopy, and system printer layouts.

  Code one device description for each device and different language or format.

  If the printer layout, or part of it, is the same as the screen layout, the LIKE parameter of the DSLLDEV macro can be used. Using the LIKE parameter reduces the size of the MCB.
- Define the external line format or formats.

  Design tags and separators for all fields that appear in the external line format. Code one device description for each communication line or screen NOPROMPT format.
- Assemble the new or changed MCBs.
- Add the new message types in the message type table, or change the MCB for an existing message type.
- All new separation, edit, checking, default setting routines must be installed. The Message Format Service (MFS) program table (DSLMPTT) must be changed. That is, existing entries might be changed or new entries added for the new programs or MCBs. When an edit checking or default setting routine is to be

added it can be named DSLMEnnn, DSLMCnnn, or DSLMDnnn, where nnn is the exit number. Then the DSLMPTT need not be modified, because the exit is dynamically loaded by DSLMMFS.

- If one of the programs or MCBs linked to Message Format Service is modified, or if new MCBs or programs are added and defined to be linked to Message Format Service, then the program DSLMMFS must be link-edited.

The Frame MCBs for Screen and Printer Panels

A screen terminal, hardcopy printer, or system printer page is divided into three areas:

- The top frame or title
- The message area, such as fields of the message header, message text, and message trailer
- Bottom frame or command line and program function key lines for screen terminals, and error message lines for all display devices

The MCBs for the message area are described earlier.

The frame MCBs are to be handled like all other MCBs. There is either one MCB for the top frame and the bottom frame, or two MCBs one for each frame. On the following pages there is a description of two MCBs, one for the top frame and one for the bottom frame. The advantage of using one MCB for the top frame and one MCB for the bottom frame is to avoid the coding of conditions in one MCB covering the top frame and bottom frame.

The fields and subfields used in the frame MCB or MCBs must be defined in the Field Definition Table (DSLFDTT).

The frame MCB names must be defined:

- In the MERVA ESA function table (DSLFNT Macro) with the FRAME parameter, such as:

  ```
  FRAME=(0TOP,0BOT)
  ```

- The MFS program table (DSLMPTT), for example:

  ```
  DSLMPT NAME=DSL0TOP
  ```

  This specification is necessary only when the MCB should be link-edited to DSLMMFS for performance reasons.

- In the MERVA ESA message type table (DSLMTTT):

  ```
  DSLMTT MTYPE=0TOP,MCB=DSL0TOP,PANEL=YES
  ```

The Top Frame

```
         TITLE    ' TOP-FRAME MCB '
         COPY     DSLCOLOR
DSL0TOP  DSLLMCB
SCREEN   DSLLDEV  TYPE=SCREEN,ID=E
         DSLLCOND O1=(TEST=DSLMSTAT),EQ=NO,O2='MSG',GOTO=GEN             [1]
SCRMT    DSLLDFLD 'MT',POS=(1,2),COLOR=&LITERAL                          [2]
         DSLLDFLD FLD=DSLMIDNS,LENGTH=9,POS=(,NEXT),COLOR=&LITERAL,    * [3]
               NI=1,PROT=YES
         DSLLDFLD FLD=DSLMIDN,LENGTH=50,COLOR=&TITLE,PROT=YES,NI=1,    * [4]
               DISP=HIGH
         DSLLCOND O1=(TEST=DSLACTS),EQ=YES,O2='L',GOTO=SCRLINE           [5]
SCRPAGE  DSLLDFLD 'Page',POS=(,NEXT+1),COLOR=&LITERAL                    [6]
         DSLLDFLD FLD=DSLACTP,LENGTH=5,POS=(,NEXT),COLOR=&LITERAL,     *
               PROT=YES,NI=1
         DSLLCOND GOTO=SCRACK
SCRLINE  DSLLDFLD 'Line',POS=(,NEXT+1),COLOR=&LITERAL
         DSLLDFLD FLD=DSLLINE,LENGTH=5,POS=(,NEXT),COLOR=&LITERAL,     *
               PROT=YES,NI=1
SCRACK   DSLLCOND O1=(TEST=MSGACK,NI=0),EQ=YES,O2=',GOTO=SCRFUNC        [7]
         DSLLDFLD FLD=MSGACK,LENGTH=69,POS=(NEXT,2),COLOR=&TITLE,NI=0, *
               DISP=HIGH,PROT=YES
         DSLLCOND GOTO=SCRFUN
SCRFUNC  DSLLDFLD 'Func',COLOR=&LITERAL,POS=(NEXT,67)
SCRFUN   DSLLDFLD FLD=DSLFUN,LENGTH=8,POS=(,72),COLOR=&LITERAL,        * [8]
               PROT=YES
SCRPAC   DSLLCOND O1=(TEST=MSGPAC,NI=0),EQ=YES,O2=',GOTO=SCRNOPAC
         DSLLDFLD FLD=MSGPAC,LENGTH=69,POS=(NEXT,2),COLOR=&TITLE,NI=0, * [9]
               DISP=HIGH,PROT=YES
         DSLLCOND O1=(TEST=DSLNUMR),EQ=YES,O2='NO',GOTO=GEN
         DSLLCOND O1=(TEST=DSLUMRNO),EQ=YES,O2=',GOTO=GEN
         DSLLCOND GOTO=SCRUMRN
SCRNOPAC DSLLCOND
         DSLLCOND O1=(TEST=DSLNUMR),EQ=YES,O2='NO',GOTO=GEN
         DSLLCOND O1=(TEST=DSLUMRNO),EQ=YES,O2=',GOTO=GEN
SRCUMR   DSLLDFLD 'UMR',COLOR=&LITERAL,POS=(NEXT,67)                     [10]
SCRUMRN  DSLLDFLD FLD=DSLUMRNO,LENGTH=8,POS=(,72),COLOR=&LITERAL,      * [11]
               PROT=YES
HARDCOPY DSLLDEV  TYPE=HARDCOPY,ID=E

⋮

HARDCOP0 DSLLDEV  TYPE=HARDCOPY,ID=0

⋮

HARDCOP1 DSLLDEV  TYPE=HARDCOPY,ID=1

⋮

PRINTER  DSLDEV  TYPE=SYSP,ID=E

⋮

PRINTER0 DSLLDEV  TYPE=SYSP,ID=0

⋮

GEN      DSLLGEN
```

Figure 155. Top Frame MCB, Screen Part

Notes:

[1] DSLLCOND

If the variable DSLMSTAT has the value 'MSG', this TOP Frame variable DSLMSTAT is generated; otherwise the following MCB instructions are ignored (GOTO=GEN).

[2] 'MT'

This instruction specifies the first literal of the TOP frame.

[3] FLD=DSLMIDNS

DSLMIDNS contains the net identifier, 'S', and the message type, '100' (for nested messages, net identifier and message type of the last message nested are appended).

[4] FLD=DSLMIDN

DSLMIDN contains the descriptive message header, for example, for a SWIFT MT 100, it contains “Customer Transfer”. You can customize the descriptive message headers in the message type table (MTT) with the DESCR parameter of the DSLMTT macro.

[5] DSLLCOND

This instruction checks whether line or page mode is used.

[6] DSLLDFLD

This and the following four instructions specify 'Page' or 'Line' and its current value at the end of the first header line.

[7] DSLLDFLD

This and the following three instructions specify either the contents of the field MSGACK (message acknowledgment) or the literal 'Func' on the first part of the second header line.

[8] FLD=DSLFUN

DSLFUN contains the name of the function currently in use and is displayed at the end of the second header line.

[9] FLD=MSGPAC

The contents of the field MSGPAC is shown if there is data present for the field. The MSGPAC field contains the result of the PAC calculation for SWIFT input and output messages.

[10] DSLLDFLD 'UMR'

The literal 'UMR' is displayed only when MSGPAC is not filled. If MSGPAC is filled, the literal is dropped because there is not enough space on the screen line.

[11] DSLLDFLD

This definition shows the Unique Message Reference (UMR) of the message, if available.

```
HARDCOPY DSLLDEV  TYPE=HARDCOPY,ID=E
         DSLLCOND O1=(TEST=DSLMSTAT),EQ=NO,O2='MSG',GOTO=HCPTOP2          [1]
HCPTOP1  DSLLDFLD FLD=DSLDATE,LENGTH=7,POS=(NEXT,2)                       [2]
         DSLLDFLD FLD=DSLTIME,LENGTH=8,POS=(,11)
         DSLLDFLD 'Logical Terminal',POS=(,55)
         DSLLDFLD FLD=DSLLTERM,LENGTH=8,POS=(,72)
HCP1MT   DSLLDFLD 'MT',POS=(NEXT,2),COLOR=&LITERAL                        [3]
         DSLLDFLD FLD=DSLMIDNS,LENGTH=9,POS=(,NEXT),COLOR=&LITERAL,     *
               NI=1,PROT=YES
         DSLLDFLD FLD=DSLMIDN,LENGTH=50,COLOR=&TITLE,PROT=YES,NI=1,     *
               DISP=HIGH
         DSLLCOND O1=(TEST=DSLACTS),EQ=YES,O2='L',GOTO=HCP1LINE
HCP1PAGE DSLLDFLD 'Page',POS=(,NEXT+1),COLOR=&LITERAL
         DSLLDFLD FLD=DSLACTP,LENGTH=5,POS=(,NEXT),COLOR=&LITERAL,      *
               PROT=YES,NI=1
         DSLLCOND GOTO=HCP1FUNC
HCP1LINE DSLLDFLD 'Line',POS=(,NEXT+1),COLOR=&LITERAL
         DSLLDFLD FLD=DSLLINE,LENGTH=5,POS=(,NEXT),COLOR=&LITERAL,      *
               PROT=YES,NI=1
HCP1FUNC DSLLDFLD 'Func',COLOR=&LITERAL,POS=(NEXT,67)
         DSLLDFLD FLD=DSLFUN,LENGTH=8,POS=(,72),COLOR=&LITERAL,         *
               PROT=YES
         DSLLCOND GOTO=HCPEND
HCPTOP2  DSLLDFLD FLD=DSLDATE,LENGTH=7,POS=(NEXT,2),COLOR=&LITERAL        [4]
         DSLLDFLD FLD=DSLTIME,LENGTH=8,POS=(,11),COLOR=&LITERAL
         DSLLDFLD 'Logical Terminal',POS=(,22),COLOR=&LITERAL
         DSLLDFLD FLD=DSLLTERM,LENGTH=8,POS=(,NEXT),COLOR=&LITERAL
         DSLLCOND O1=(TEST=DSLACTS),EQ=YES,O2='L',GOTO=HCP2LINE
HCP2PAGE DSLLDFLD 'Page',POS=(,51),COLOR=&LITERAL
         DSLLDFLD FLD=DSLACTP,LENGTH=5,POS=(,NEXT),COLOR=&LITERAL
         DSLLCOND GOTO=HCP2FUNC
HCP2LINE DSLLDFLD 'Line',POS=(,51),COLOR=&LITERAL
         DSLLDFLD FLD=DSLLINE,LENGTH=5,POS=(,NEXT),COLOR=&LITERAL
HCP2FUNC DSLLDFLD 'Function',POS=(,62),COLOR=&LITERAL
         DSLLDFLD FLD=DSLFUN,LENGTH=8,POS=(,NEXT),COLOR=&LITERAL
HCPEND   DSLLCOND
HARDCOP0 DSLLDEV  TYPE=HARDCOPY,ID=0                                      [5]
         DSLLCOND GOTO=HARDCOP1
         DSLLEXIT IMBED=DUMMY
HARDCOP1 DSLLDEV  TYPE=HARDCOPY,ID=1                                      [6]
         DSLLDFLD ' ',POS=(3,2),COLOR=&LITERAL
```

Figure 156. Top Frame MCB, Hardcopy Part

Notes:

[1] DSLLCOND

If the variable DSLMSTAT has the value 'MSG' the instructions starting from the label HCPTOP1 until the label HCPTOP2 are used, otherwise a GOTO to the label HCPTOP2 is performed.

[2] DSLLDFLD

This and the following three instructions specify the first header line, containing date, time, and logical terminal used.

[3] HCP1MT

This and the following instructions specify the same layout of the next header lines as described for the first two lines of the screen part, except that the content of the MSGACK is not shown.

[4] HCPTOP2

This and the following instructions specify the layout of the top line for printouts of MCBs other than those containing DSLMSTAT='MSG', such as Help-MCBs.

[5] HARDCOP0

This and the following instruction specifies a hardcopy printout without top frame. To avoid having an empty MCB definition, the DSLLEXIT macro with the IMBED=DUMMY parameter is used.

[6] HARDCOP1

Specifies a hardcopy printout with a top frame of 3 blank lines (POS=(3,2)).

```
PRINTER  DSLLDEV  TYPE=SYSP,ID=E                                          [1]
         DSLLDFLD ' ',POS=(1,2),COLOR=&LITERAL
         DSLLCOND O1=(TEST=DSLMSTAT),EQ=NO,O2='MSG',GOTO=PRNTOP2
PRNTOP1  DSLLDFLD FLD=DSLDATE,LENGTH=7,POS=(NEXT,2)
         DSLLDFLD FLD=DSLTIME,LENGTH=8,POS=(,11)
         DSLLDFLD 'No.',POS=(,67)
         DSLLDFLD FLD=DSLSDYNO,LENGTH=5,POS=(,72)
PRN1MT   DSLLDFLD 'MT',POS=(NEXT,2),COLOR=&LITERAL
         DSLLDFLD FLD=DSLMIDNS,LENGTH=9,POS=(,NEXT+1),COLOR=&LITERAL,  *
               NI=1,PROT=YES
         DSLLDFLD FLD=DSLMIDN,LENGTH=50,COLOR=&TITLE,PROT=YES,NI=1,    *
               DISP=HIGH
         DSLLCOND O1=(TEST=DSLACTS),EQ=YES,O2='L',GOTO=PRN1LINE
PRN1PAGE DSLLDFLD 'Page',POS=(,NEXT+2),COLOR=&LITERAL
         DSLLDFLD FLD=DSLACTP,LENGTH=5,POS=(,NEXT+1),COLOR=&LITERAL,   *
               PROT=YES,NI=1
         DSLLCOND GOTO=PRN1FUNC
PRN1LINE DSLLDFLD 'Line',POS=(,NEXT+2),COLOR=&LITERAL
         DSLLDFLD FLD=DSLLINE,LENGTH=5,POS=(,NEXT+1),COLOR=&LITERAL,   *
               PROT=YES,NI=1
PRN1FUNC DSLLDFLD 'Func',COLOR=&LITERAL,POS=(NEXT,67)
         DSLLDFLD FLD=DSLFUN,LENGTH=8,POS=(,72),COLOR=&LITERAL,        *
               PROT=YES
         DSLLCOND GOTO=PRNBLANK
PRNTOP2  DSLLDFLD FLD=DSLDATE,LENGTH=7,POS=(NEXT,2),COLOR=&LITERAL
         DSLLDFLD FLD=DSLTIME,LENGTH=8,POS=(,11),COLOR=&LITERAL
         DSLLDFLD 'Message No.',POS=(,24),COLOR=&LITERAL
         DSLLDFLD FLD=DSLSDYNO,LENGTH=5,POS=(,NEXT+1),COLOR=&LITERAL
         DSLLCOND O1=(TEST=DSLACTS),EQ=YES,O2='L',GOTO=PRN2LINE
PRN2PAGE DSLLDFLD 'Page',POS=(,45),COLOR=&LITERAL
         DSLLDFLD FLD=DSLACTP,LENGTH=5,POS=(,NEXT+1),COLOR=&LITERAL
         DSLLCOND GOTO=PRN2FUNC
PRN2LINE DSLLDFLD 'Line',POS=(,45),COLOR=&LITERAL
         DSLLDFLD FLD=DSLLINE,LENGTH=5,POS=(,NEXT+1),COLOR=&LITERAL
PRN2FUNC DSLLDFLD 'Function',POS=(,59),COLOR=&LITERAL
         DSLLDFLD FLD=DSLFUN,LENGTH=8,POS=(,NEXT+1),COLOR=&LITERAL
PRNBLANK DSLLDFLD ' ',POS=(NEXT,2),COLOR=&LITERAL
PRINTER0 DSLLDEV  TYPE=SYSP,ID=0
         DSLLCOND GOTO=PRINTER1
         DSLLEXIT IMBED=DUMMY
PRINTER1 DSLLDEV  TYPE=SYSP,ID=1
         DSLLDFLD ' ',POS=(3,2),COLOR=&LITERAL
*
GEN      DSLLGEN
         END
```

Figure 157. Top Frame MCB, System Printer Part

Notes:

[1] PRINTER

The following instructions supplied for a system printer device correspond to those described for the hardcopy device. Some blank lines are added and the current message number is printed instead of the logical terminal identifier.

The Bottom Frame

```
         TITLE    ' BOTTOM-FRAME MCB '
         COPY     DSLCOLOR
DSL0BOT  DSLLMCB
SCREEN   DSLLDEV  TYPE=SCREEN,ID=E
         DSLLDFLD FLD=DSLERR,LENGTH=79,POS=(NEXT,2),COLOR=&BELL,       * [1]
               PROT=YES,DISP=HIGH,EDIT=903
         DSLLDFLD 'Command',COLOR=&LITERAL
         DSLLDFLD '=====>',POS=(,NEXT),COLOR=&TITLE,DISP=HIGH
         DSLLDFLD FLD=DSLCMDL,LENGTH=64,COLOR=&DATA,EDIT=903
         DSLLUNIT DACNT=(1,2)                                            [2]
         DSLLDFLD 'PF',POS=(NEXT,2),COLOR=&LITERAL
         DSLLDFLD FLD=DSLPFKL,LENGTH=76,POS=(,NEXT),COLOR=&LITERAL,    *
               PROT=YES
         DSLLUEND
HARDCOPY DSLLDEV  TYPE=HARDCOPY,ID=E
         DSLLDFLD ' ',POS=(2,2)                                          [3]
HARDCOP0 DSLLDEV  TYPE=HARDCOPY,ID=0
         DSLLCOND GOTO=HARDCOP1
         DSLLEXIT IMBED=DUMMY
HARDCOP1 DSLLDEV  TYPE=HARDCOPY,ID=1
         DSLLDFLD ' ',POS=(3,2)
PRINTER  DSLLDEV  TYPE=SYSP,ID=E
         DSLLDFLD ' ',POS=(2,2)
PRINTER0 DSLLDEV  TYPE=SYSP,ID=0
         DSLLCOND GOTO=PRINTER1
         DSLLEXIT IMBED=DUMMY
PRINTER1 DSLLDEV  TYPE=SYSP,ID=1
         DSLLDFLD ' ',POS=(3,2)
*
         DSLLGEN
         END
```

Figure 158. Bottom Frame MCB

Notes:

[1] EDIT=903

The first Bottom Line is reserved for system messages (contents of the field DSLERR) and is prepared for display by the editing routine DSLME903 (EDIT=903). The edit routine sets the color of this field according to the severity of the error, for example, 'YELLOW' for warning or error messages.

[2] DSLLUNIT DACNT=(1,2) ... DSLLUEND

The following instructions specify the layout of the descriptive lines for the Program Function Keys. Two data areas (DACNT=(1,2)) of the field DSLPFKL are displayed after the literal 'PF' supplied on each line.

Note: The text can be customized in the Program Function Key Table.

[3] DSLLDFLD

For every hardcopy and printer device a blank line is supplied for the bottom part.

Chapter 11. Cover MCBs

Cover MCBs are designed to provide mapping facilities for composed messages. Messages can be composed of MERVA Link, Telex Link, SWIFT and other information. Cover MCBs are also needed if a message consists of only one kind of information, for example, SWIFT information.

MERVA ESA provides the following sample cover MCBs:

DSL0COV is used to:

- Display or print all internal MERVA ESA messages and panels
- Display, print, and format SWIFT messages for the network (exit field DSLEXIT)
- Display or print free formatted telex messages (exit field ENLEXIT)
- Display formatted telex messages (exit field ENLEXIT and DSLEXIT)

DSLKCOV is used to display, print, and format SWIFT, telex, and user-defined messages, including MERVA-MQI Attachment control fields. Refer to "Displaying MQI Control Block Data" on page 326.

ENLTCOV is used to display, print, and format:

- Formatted telex messages (exit fields ENLEXIT and DSLEXIT)
- Free formatted telex messages (exit field ENLEXIT)
- SWIFT messages (exit field DSLEXIT)

EKAMCOV is used to display, print, and format SWIFT and telex messages, including MERVA Link control fields.

Coding Cover MCBs

You can define your own cover MCB for a display or print function, or for mapping purposes. You can include system or user control fields, or omit parts of the message. A cover MCB should be assigned to an *own exitfield* in the TOF (nesting level indicator), using the NLIND parameter in the Message Type Table Entry, for example, NLIND=OWNEXIT.

Only fields on nesting level 0 must be assigned to this exit field. Use the parameter NL0=YES in the MTT-entry to specify this.

The cover MCB must be defined in:

- The MFS program table DSLMPTT
- The message type table DSLMTTT

Display or print functions that refer to the cover MCB must be used with the MSGID parameter in the function table DSLFNTT.

Example for DSL0COV

DSL0COV is the cover MCB provided for internal and SWIFT messages. It is for displaying, printing, and formatting SWIFT messages in their original form.

```
         TITLE 'Cover of a Message'
         COPY  DSLCOLOR
DSL0COV  DSLLMCB
SCREEN01 DSLLDEV  TYPE=SCREEN,ID=E                                      [1]
         DSLLCOND O1=(TEST=ENLEXIT,NI=0),EQ=YES,O2='',GOTO=SCRNSW       [2]
         DSLLCOND O1=(TEST=DSLEXIT,NI=0),EQ=NO,O2='',GOTO=SCRNTXSW      [3]
         DSLLEXIT FLD=ENLEXIT,NI=0                                      [4]
         DSLLCOND GOTO=SCRNGEN
SCRNTXSW DSLLEXIT IMBED=TX                                              [5]
         DSLLDFLD 'Message                   :',POS=(NEXT,02),          *
               COMMENT=Y
         DSLLDFLD FLD=DSLEXIT,NI=0,POS=(,NEXT),LENGTH=4,                *
               DISP=HIGH,PROT=YES,COMMENT=Y
         DSLLDFLD 'Initialized',POS=(,NEXT),COMMENT=Y
         DSLLDFLD '-----------------------------------------------------*
               --------------------------',POS=(NEXT,02),COMMENT=Y
SCRNSW   DSLLEXIT FLD=DSLEXIT,NI=0                                      [6]
SCRNGEN  DSLLCOND
HARDCOPY DSLLDEV  TYPE=HARDCOPY,ID=E                                    [7]
         DSLLEXIT FLD=DSLEXIT,NI=0                                      [8]
         PRINTER  DSLLDEV  TYPE=SYSP,ID=E,LIKE=HARDCOPY
LINES    DSLLDEV  TYPE=NET,ID=S,SEP=X'0D25'                             [9]
         DSLLEXIT FLD=DSLEXIT,NI=0                                     [10]
LINEW    DSLLDEV  TYPE=NET,ID=W,SEP=X'0D25',LIKE=LINES
LINEX    DSLLDEV  TYPE=NET,ID=X,SEP=X'0D25',LIKE=LINES
LINEY    DSLLDEV  TYPE=NET,ID=Y,SEP=X'0D25',LIKE=LINES
         DSLLGEN
         END
```

Figure 159. Coding Example for Cover MCB, DSL0COV

Notes:

[1] `SCREEN   DSLLDEV  TYPE=SCREEN,ID=E`

The screen part of the MCB covers the display of internal, SWIFT, and Telex Link messages.

[2] `DSLLCOND O1=(TEST=ENLEXIT,NI=0),EQ=YES,O2='',GOTO=SCRNSW`

Internal field ENLEXIT on NI=0 is checked. If it is empty processing continues with label SCRNSW [6], that is, it is not a telex message. Otherwise processing continues with the next statement.

[3] `DSLLCOND O1=(TEST=DSLEXIT,NI=0),EQ=NO,O2='',GOTO=SCRNTXSW`

Internal field DSLEXIT on NI=0 is checked. If it is empty, processing continues with the next statement, that is, it is not a SWIFT message. Otherwise processing continues with label SCRNTXSW [5].

[4] `DSLLEXIT FLD=ENLEXIT,NI=0`

MCB processing continues with the MCB referred to by field ENLEXIT on NI=0. ENLEXIT contains the message ID for telex messages.

[5] `SCRNTXSW DSLLEXIT IMBED=TX`

MCB processing continues by embedding MT=TX. This is the MCB containing the telex header fields.

[6] `SCRNSW   DSLLEXIT FLD=DSLEXIT,NI=0`

MCB processing continues with the MCB referred to by field DSLEXIT on NI=0. DSLEXIT contains the message ID of the SWIFT messages.

[7] `HARDCOPY DSLLDEV  TYPE=HARDCOPY,ID=E`

The hardcopy section of the MCB covers the printing of SWIFT and internal messages.

[8] `DSLLEXIT FLD=DSLEXIT,NI=0`

MCB processing continues with the MCB referred to by field DSLEXIT on NI=0.

[9] `LINES    DSLLDEV  TYPE=NET,ID=S,SEP=X'0D25'`

The net section of the MCB covers the mapping of SWIFT messages.

[10] `DSLLEXIT FLD=DSLEXIT,NI=0`

MCB processing continues with the MCB referred to by field DSLEXIT on NI=0.

Example for EKAMCOV

EKAMCOV is the MERVA Link cover MCB provided for MERVA Link functions. It is used to display or print a message including MERVA Link control information.

```
         TITLE    'MERVA Link Cover MCB'
         COPY     DSLCOLOR
EKAMCOV  DSLLMCB
SCREEN   DSLLDEV  TYPE=SCREEN
         DSLLCOND SCREEN,GOTO=SCREEN01
         DSLLCOND SYSP,GOTO=SCREEN01
         DSLLCOND O1=(TEST=DSLUSRMN,NI=0,LENGTH=3),EQ=NO,O2='MSG',       *
               GOTO=SCREENO2
SCREEN01 DSLLCOND
         DSLLDFLD 'Start of MERVA Link Control Fields Display',          *
               POS=(NEXT,02),COLOR=&WATCH
         DSLLEXIT IMBED=MCTL                                             [1]
         DSLLCOND O1=(TEST=ENLEXIT,NI=0),EQ=YES,O2='',GOTO=SCREENSW      [2]
         DSLLCOND PAGE=NEW
         DSLLDFLD ' ',POS=(NEXT,02)
         DSLLDFLD  'Message                   :',POS=(NEXT,02),          *
               COMMENT=Y
         DSLLDFLD  FLD=ENLEXIT,NI=0,POS=(,NEXT),LENGTH=5
              DISP=HIGH,PROT=YES,COMMENT=Y
         DSLLDFLD 'Initialized',POS=(,NEXT),COMMENT=Y
         DSLLDFLD ------------------------------------------------------*
              --------------------------',POS=(NEXT,02),COMMENT=Y
         DSLLEXIT FLD=ENLEXIT,NI=0                                       [3]
         DSLLCOND GOTO=SCREEND
SCREENSW DSLLCOND
         DSLLCOND PAGE=NEW
         DSLLEXIT FLD=DSLEXIT,NI=0                                       [4]
SCREEND  DSLLCOND
HARDCOPY DSLLDEV  TYPE=HARDCOPY,ID=E,LIKE=SCREEN
PRINTER  DSLLDEV  TYPE=SYSP,ID=E,LIKE=SCREEN
LINEM    DSLLDEV  TYPE=NET,ID=M,SEP=''                                   [5]
         DSLLEXIT IMBED=MCTL
         DSLLCOND O1=(TEST=ENLEXIT,NI=0),EQ=YES,O2='',GOTO=LINEMSW
         DSLLEXIT FLD=ENLEXIT,NI=0
         DSLLCOND GOTO=LINEX
LINEMSW  DSLLEXIT FLD=DSLEXIT,NI=0
LINEX    DSLLDEV  TYPE=NET,ID=X,SEP=X'0D25'
         DSLLCOND O1=(TEST=ENLEXIT,NI=0),EQ=YES,O2='',GOTO=LINESW
         DSLLEXIT FLD=ENLEXIT,NI=0
         DSLLCOND GOTO=LINET
LINESW   DSLLEXIT FLD=DSLEXIT,NI=0
LINES    DSLLDEV  TYPE=NET,ID=S,LIKE=LINEM
LINET    DSLLDEV  TYPE=NET,ID=S,LIKE=LINEM
LINEW    DSLLDEV  TYPE=NET,ID=S,LIKE=LINEM
LINEY    DSLLDEV  TYPE=NET,ID=S,LIKE=LINEX
         DSLLGEN
         END
```

Figure 160. Coding Example for Cover MCB, EKAMCOV

Notes:

[1] `DSLLEXIT IMBED=MCTL`

Message display starts with the MCTL MCB that contains the MERVA Link control fields.

[2] `DSLLCOND O1=(TEST=ENLEXIT,NI=0),EQ=YES,O2='',GOTO=SCREENSW`

Internal field ENLEXIT on NI=0 is checked. If it is empty, processing continues with label SCREENSW.

[3] `DSLLEXIT FLD=ENLEXIT,NI=0`

MCB processing continues with the MCB referred to by field ENLEXIT on NI=0. ENLEXIT contains the message ID for telex messages.

[4] `DSLLEXIT  FLD=DSLEXIT,NI=0`

MCB processing continues with the MCB referred to by field DSLEXIT on NI=0.

[5] `LINEM    DSLLDEV  TYPE=NET,ID=M,SEP=''`

LINEM is used to map a message including the MERVA Link control fields to or from its net format.

Example for ENLTCOV

ENLTCOV is the cover MCB provided for the TELEX functions. It covers the layout for free formatted and for formatted telex messages, for SWIFT messages, and for MERVA ESA internal messages.

```
ENLTCOV  DSLLMCB
SCREEN   DSLLDEV  TYPE=SCREEN,ID=E
         DSLLCOND SCREEN,GOTO=SCREEN01
         DSLLCOND SYSP,GOTO=SCREEN01
         DSLLCOND O1=(TEST=DSLUSRMN,NI=0),EQ=YES,O2='',                *
               GOTO=SCREEN01
SCREEN00 DSLLCOND
         DSLLCOND O1=(TEST=DSLUSRMN,NI=0,LENGTH=3),EQ=NO,O2='MSG',     *
               GOTO=SCRNSW
SCREEN01 DSLLCOND
         DSLLCOND O1=(TEST=ENLEXIT,NI=0),EQ=YES,O2='',GOTO=SCRNSW       [1]
         DSLLCOND O1=(TEST=DSLEXIT,NI=0),EQ=NO,O2='',GOTO=SCRNTXSW      [2]
         DSLLEXIT FLD=ENLEXIT,NI=0                                      [3]
         DSLLCOND GOTO=SCRNGEN
SCRNTXSW DSLLEXIT IMBED=TX                                              [4]
         DSLLDFLD  'Message'                 :',POS=(NEXT,02),         *
               COMMENT=Y
         DSLLDFLD FLD=DSLEXIT,NI=0,POS=(,NEXT),LENGTH=4,               *
               DISP=HIGH,PROT=YES,COMMENT=Y
         DSLLDFLD 'Initialized',POS=(,NEXT),COMMENT=Y
         DSLLDFLD ----------------------------------------------------*
              ----------------------------',POS=(NEXT,02),COMMENT=Y
SCRNSW   DSLLEXIT FLD=DSLEXIT,NI=0                                      [5]
SCRNGEN  DSLLCOND
HARDCOPY DSLLDEV  TYPE=HARDCOPY,ID=E,LIKE=SCREEN
PRINTER  DSLLDEV  TYPE=SYSP,ID=E,LIKE=SCREEN
LINEX    DSLLDEV  TYPE=NET,ID=X,SEP=X'0D25'
         DSLLCOND O1=(TEST=ENLEXIT,NI=0),EQ=YES,O2='',GOTO=LINEMSW
         DSLLCOND O1=(TEST=ENLEXIT,NI=0,LENGTH=4),EQ=YES,O2='TCOV',     *
               GOTO=LINETXSW
         DSLLEXIT FLD=ENLEXIT,NI=0
         DSLLCOND GOTO=LINEEND
LINETXSW DSLLEXIT IMBED=TX
LINESW   DSLLEXIT FLD=DSLEXIT,NI=0
LINEEND  DSLLCOND
LINEY    DSLLDEV  TYPE=NET,ID=Y,LIKE=LINEX
LINET    DSLLDEV  TYPE=NET,ID=T,LIKE=LINEX
LINES    DSLLDEV  TYPE=NET,ID=S,LIKE=LINEX
LINEW    DSLLDEV  TYPE=NET,ID=W,LIKE=LINEX
* FORM K IS NECESSARY FOR TK-EXTRACT FUNCTION
LINEK    DSLLDEV  TYPE=NET,ID=K,SEP=X'0D25'
         DSLLEXIT FLD=DSLEXIT,NI=0
LINEGEN  DSLLCOND
         DSLLGEN
         END
```

Figure 161. Coding Example for Cover MCB, ENLTCOV

Notes:

[1] `DSLLCOND O1=(TEST=ENLEXIT,NI=0),EQ=YES,O2='',GOTO=SCRNSW`

Internal field ENLEXIT on NI=0 is checked. If it is empty, processing continues with label SCRNSW [5] indicating that it is not a TELEX message. Otherwise, processing continues with the next statement.

[2] `DSLLCOND O1=(TEST=DSLEXIT,NI=0),EQ=NO,O2='',GOTO=SCRNTXSW`

Internal field DSLEXIT on NI=0 is checked. If it is empty, processing continues with the next statement, it is not a SWIFT message. Otherwise, processing continues with label SCRNTXSW.

[3] `DSLLEXIT FLD=ENLEXIT,NI=0`

MCB processing continues with the MCB referred to by field ENLEXIT on NI=0. ENLEXIT contains the message ID for telex messages.

[4] `SCRNTXSW DSLLEXIT IMBED=TX`

MCB processing continues by embedding MT=TX. This is the MCB containing the telex header fields.

[5] `SCREENSW DSLLEXIT FLD=DSLEXIT,NI=0`

MCB processing continues with the MCB referred to by field DSLEXIT on NI=0. DSLEXIT contains the message ID of the SWIFT messages.

Help MCBs

Help MCBs contain mostly fixed information. See the *MERVA for ESA User's Guide* for how to display help panels on a screen terminal.

In a MERVA ESA installation, the help MCBs supplied by MERVA ESA can be changed, or new help MCBs can be created. Help MCBs are processed like message MCBs.

The Help MCB DSLHELP shows a list of the available help panels. It should be updated if user-written help panels are added.

The MERVA ESA help panels can be accessed directly by the command **help 0xxxx** (xxxx = MCB name - prefix DSL). That is, **help 0hpfk** shows the Help panel for the Program Function Keys on the screen. The help panels can also be accessed using the command **help** *mcbname*. It is possible to select a specific page in a help panel directly by using the command **help** *mcbname pagenum*.

To get help information about a user or operator command enter the command **help** *commandname*. For example **help login** shows the help panel for the login command.

There are two ways to get help information for user error messages. If you receive an error message during message processing on the screen you can receive help information about this error by entering the **help** command either on the command line or by pressing PF 1. Alternatively you can get the help panel for any user error message by applying the **show** command with the error message ID as operand. The command **show dws3686** displays the help panel for the error message:

DWS3686 Correspondent's SWIFT address in header must have length 8 or 11.

Note: DWS3686 refers to an MCB which is displayed by this command. If you supply your own error messages, you should provide appropriate MCBs.

MERVA ESA provides you with a Help Menu (master panel, MCB DSLHELP), which uses the following help panels:

DSLHIDX	Help Index MERVA ESA
DSLHBASE	Base Help Menu
DWSHELP	SWIFT Link Help Menu
ENLHELP	Telex Link Help Menu
EHAHELP	MERVA Link Help Menu
DSLHENV	Environment Data
DSLHMTR	Message-Trace Display
DSLHPFK	Program Function Keys
DSLOUMR	Unique Message Reference
DSLHCMD	Operator Commands
DSLHSCC	Screen Commands
DSLHUSR	User File Maintenance Commands
DSLHFLM	General File Maintenance Commands
DSLHRETC	Return and Reason Codes

The SWIFT Link Help Menu uses the following help panels for the SWIFT Link specific functions:

DWSHCMD2	Operator Commands for SWIFT Network
DWSHMTB	Financial Message Header
DWSHMTB	Financial Message Types
DWSHMTS	System Message Types
DWSHCUR	Currency codes
DWSHCUR2	Currency codes in file and table
DWSHAUT	Authenticator Key File Maintenance Commands
DWSHRETC	Return and Reason Codes
DWSHNAK	SWIFT Error (NAK) Codes
DWSHMAC	Message Authentication

The SWIFT Link help panels can be accessed directly by the command **help Sxxxx** (xxxx = MCB name - prefix DWS), that is, **help shcur** shows the Help panel for the currency codes on the screen. The help panels can also be accessed using the command **help** *mcbname*.

The Telex Link Help MCB contains panels for the following Telex Link specific functions:

- Telex Link Operator Commands
- Telex Link status codes for Headoffice Telex on a fault-tolerant system

The MERVA Link Application Control Facility help panel EKAACHP shows:

- The Application Control Facility commands
- The control information for a specific application support process
- Information about the local and partner systems

The MERVA Link general help panel EKAHELP also shows MERVA Link Diagnostic Codes and Abbreviations.

Chapter 12. Message Type Table (DSLMTTT)

The MERVA ESA message type table defines the message types and related information (such as the MCB that describes the message type) for MERVA ESA. MERVA ESA uses a message type table that contains, in the assembler source only, the DSLMTT TYPE=INITIAL and DSLMTT TYPE=FINAL macros and an assembler COPY statement for each network link. The copy members of each component finally contain the message type definitions needed by the component.

These copy members contain all message type definitions required for the operation of MERVA ESA including the SWIFT Link to process all SWIFT messages on all display devices and the connection to the SWIFT network, the Telex Link and the MERVA Link.

The message type table can be altered by changing the appropriate DSLMTT macros or by adding new DSLMTT macros, described in the *MERVA for ESA Macro Reference*. After modification, the message type table must be assembled and link-edited.

The name of the message type table can be customized in DSLPRM.

The DSLMTT macro can be used in two forms:

- Map the MERVA ESA message type table header, the entry section, and index section.
- Generate the MERVA ESA message type table (DSLMTTT).

Mapping the Areas of the Message Type Table

The following macro maps all areas of the message type table:

```
DSLMTT TYPE=MAP             MAP ALL AREAS OF MTT
```

Generating the Message Type Table

The following instructions are used to generate the message type table:

```
MTTT     TITLE 'MERVA MESSAGE TYPE TABLE'
DSLMTTT  DSLMTT TYPE=INITIAL                                          [1]
         COPY  DSLMTTTC            MERVA ESA MESSAGE TYPE TABLE       [2]
         COPY  DWSMTTTC            SWIFT LINK MESSAGE TYPE TABLE      [3]
         COPY  ENLMTTTC            TELEX LINK MESSAGE TYPE TABLE      [4]
         COPY  EKAMTTTC            MERVA LINK MESSAGE TYPE TABLE      [5]
         COPY  EKAMTTSC            FMT MESSAGE TYPE TABLE             [6]
         DSLMTT TYPE=FINAL                                            [7]
         END
```

Figure 162. The MERVA ESA Message Type Table

Notes:

[1] DSLMTTT

The label of the DSLMTT TYPE=INITIAL macro must be coded and is used as the name of the message type table. This must be the first macro.

[2] DSLMTTTC

This copy code describes the MERVA ESA message types and message identifications. A coding example is given in Figure 163.

[3] DWSMTTTC

This copy code describes the SWIFT Link message types and message identifications. A coding example is given in Figure 164.

[4] ENLMTTTC

This copy code describes the Telex Link message types and message identifications. A coding example is given in Figure 165.

[5] EKAMTTTC

This copy code describes the MERVA Link message types and message identifications. A coding example is given in Figure 166.

[6] EKAMTTSC

This copy code describes the FMT/ESA with MERVA Link message types and message identifications.

[7] DSLMTT

TYPE=FINAL is the last macro and completes the message type table definition.

Message Type Table Definitions

This copy member has the name DSLMTTTC.

```
*-------------------------------------------------------------*
*        MERVA MESSAGE IDENTIFICATIONS                        *
*-------------------------------------------------------------*
         DSLMTT MTYPE=0BOT,MCB=DSL0BOT,PANEL=YES                    [1]
         DSLMTT MTYPE=0CMD,MCB=DSL0CMD,                        *    [2]
               DESCR='Operator Command Processing'
         DSLMTT MTYPE=0CORN,MCB=DSL0CORN,MTGEN=NO                   [3]
         DSLMTT MTYPE=0HENV,...,SYNONYM=INFO                        [4]

⋮
```

Figure 163. The MERVA ESA Message Type Table Copy Member

Notes:

[1] DSLMTT MTYPE=0BOT,MCB=DSL0BOT,PANEL=YES

This DSLMTT macro generates an entry for the Bottom Frame MCB DSL0BOT in the Message Type Table and assigns it the message ID 0BOT. PANEL=YES specifies that this message type is used as an internal panel identifier only.

[2] DSLMTT MTYPE=0CMD,MCB=DSL0CMD

This DSLMTT macro generates an entry for the Command Processing Panel MCB DSL0CMD and assigns it the message ID 0CMD. DESCR='Operator Command Processing' defines the title line used by the top frame MCB when displaying this panel.

[3] DSLMTT MTYPE=0CORN,MCB=DSL0CORN,MTGEN=NO

This DSLMTT macro generates an entry for the MERVA ESA Nicknames File MCB DSL0CORN and assigns it the message ID 0CORN. MTGEN=NO specifies that a message with this message type can be generated only by a MERVA ESA application program.

[4] DSLMTT MTYPE=0HENV,...,SYNONYM=INFO

This DSLMTT macro generates an entry for the environment information display. The display shows the current time and date, the current function, the logical terminal name, and other useful information about the user session. The synonym specification means that instead of the command **show 0henv** the alternative command **show info** can be specified. Both commands result in the display of the information panel.

SWIFT Link Message Type Table Definitions

This copy member has the name DWSMTTTC, and it contains the definitions for all the SWIFT message types.

This table must only be changed if SWIFT changes a message type or provides new message types.

Figure 164 shows the beginning of the copy member DWSMTTTC. The complete copy member is in the MERVA ESA macro (MVS) library.

```
* CONSTANTS FOR SWIFT LINK
AUT      EQU   X'80'               AUTHENTICATION REQUIRED            [1]

⋮

*-------------------------------------------------------------*
*  SWIFT Link   Banking Message Types / FIN Application
*-------------------------------------------------------------*
  DSLMTT MTYPE=S100,MCB=DWS100,NETSPEC=(AUT),CHECK=1002,LENGTH=2000, * [2]
              DESCR='Customer Transfer'
  DSLMTT MTYPE=S102,MCB=DWS102,NETSPEC=(AUT),CHECK=1002,LENGTH=10000,* [3]
              DESCR='Mass Payments'

⋮

  DSLMTT MTYPE=S292,MCB=DWSX92,NESTING=YES,CHECK=1002,LENGTH=2000,   * [4]
              NETSPEC=(AUT,'NXTLEV=(¬200,¬201,2**)'),                *
              DESCR='Request for Cancellation'
  DSLMTT MTYPE=S295,MCB=DWSX95,NESTING=YES,CHECK=1002,LENGTH=2000,   * [5]
              NETSPEC=(AUT,'NXTLEV=2'),                              *
              DESCR='Queries'
  DSLMTT MTYPE=S296,MCB=DWSX96,NESTING=YES,CHECK=1002,LENGTH=2000,   *
              NETSPEC=(AUT,'NXTLEV=2'),                              *
              DESCR='Answers'
  DSLMTT MTYPE=S299,MCB=DWSX99,NETSPEC=(AUT),CHECK=1002,LENGTH=2000, *
              DESCR='Free Format'
  DSLMTT MTYPE=SF020,MCB=DWSF020,CHECK=1002,SYNONYM=S020,            * [6]
              DESCR='Retrieval Request(Text && History)'

⋮
```

Figure 164. The SWIFT Link Message Type Table Copy Member

Notes:

[1] AUT EQU X'80'

This instruction defines an assembler symbol to be referenced in the NETSPEC operand. “AUT” specifies that this message type requires the SWIFT authentication.

[2] DSLMTT MTYPE=S100

With this instruction the message identifier S100 for SWIFT message type 100 is assigned to the MCB DWS100. NETSPEC=(AUT) specifies that authentication of this message type is required. CHECK=1002 specifies the number of the message-type-specific checking exit called by the MFS program DSLMCHE for message checking. LENGTH=2000 specifies that the SWIFT defined message length limit is 2000 bytes. If a message is larger than this value, the message is not sent to SWIFT. DESCR= specifies the descriptive title of the message type used by the TOP frame MCB when displaying this message.

[3] DSLMTT MTYPE=S102

With this instruction the message identifier S102 for SWIFT message type 102 is assigned to the MCB DWS102. NETSPEC=(AUT) specifies that authentication of this message type is required. CHECK=1002 specifies the number of the message-type-specific checking exit called by the MFS program DSLMCHE for message checking. LENGTH=10000 specifies that the SWIFT defined message length limit is 10000 bytes for this message type. If a message is larger than this value, the message is not sent to SWIFT. DESCR= specifies the descriptive title of the message type used by the TOP frame MCB when displaying this message.

[4] DSLMTT MTYPE=S292

With this instruction the message identifier S292 for SWIFT MT 292 is assigned to the MCB DWSX92. NESTING=YES specifies that nesting of messages is allowed for this message type, and further information on the message types to be nested is passed with the NETSPEC operand. NXTLEV=(¬200,¬201,2**) specifies that MTs 202 to 299 can be nested.

Note: The parameter NXTLEV is defined in the message type table for the SWIFT Link message processing programs DWSMX002 and DWSMU154.

[5] DSLMTT MTYPE=S295

With this instruction the message identifier S295 for SWIFT MT 295 is assigned to the MCB DWSX95. NESTING=YES specifies that nesting of messages is allowed for this message type, and further information on the message types to be nested is passed with the NETSPEC operand. NXTLEV=2 specifies that MTs 200 to 299 can be nested.

[6] DSLMTT MTYPE=SF020

With this instruction the message identifier SF020 for the SWIFT financial application message type 020 is assigned to MCB DWSF020. The parameter SYNONYM=S020 specifies that S020 can be used as synonym for this message type. The short form 020 can be entered on a message selection panel to create a message of this type. In MERVA/370 V2 the message identification S020 was used for the SWIFT I system message type 20.

Telex Link Message Type Table Definitions

All message types that are used by the Telex Link must be specified with an DSLMTT macro and must be incorporated into the copy book ENLMTTTC.

The sample entries used by the Telex Link are shown in Figure 165.

```
         DSLMTT MTYPE=TINV,MCB=ENLMTINV,NLO=YES,NLIND=ENLEXIT,       * [1]
               DESCR='Invalid Telex'
         DSLMTT MTYPE=TRCV,MCB=ENLMTRCV,NLO=YES,NLIND=ENLEXIT,       *
               DESCR='Received Telex'
         DSLMTT MTYPE=TELEX,MCB=ENLMTSND,NLO=YES,NLIND=ENLEXIT,      *
               DESCR='Telex'
         DSLMTT MTYPE=TCOR,MCB=ENLTCOR,MTGEN=NO
*/*-------------------------------------------------------------*
         DSLMTT MTYPE=TX,MCB=ENLTXHD,NLIND=ENLEXIT,NLO=YES,          *
               DESCR='Telex Header'
         DSLMTT MTYPE=TCOV,MCB=ENLTCOV,NLIND=ENLEXIT,NLO=YES,        *
               DESCR='Telex Cover'
*/*-------------------------------------------------------------*
         DSLMTT MTYPE=LHELP,MCB=ENLHELP,PANEL=YES
         DSLMTT MTYPE=ENLLTXIP,MCB=ENLLTXIP,MTGEN=NO
```

Figure 165. DSLMTT Entries for the Telex Link in the Copy Book ENLMTTTC

Notes:

[1] DSLMTT MTYPE=TINV

With this instruction, MT=TINV is assigned to MCB ENLMTINV. It is also assigned to ENLEXIT and contains only fields on nesting level 0.

MERVA Link Message Type Table Definitions

All message types that are used by MERVA Link must be specified in the MERVA ESA Message Type Table (DSLMTTT).

The sample DSLMTTT entries used by MERVA Link are shown in Figure 166 on page 382. The copy book EKAMTTTC of the MERVA ESA macro library contains the macro instructions to generate these DSLMTTT entries.

```
*--------------------------------------------------------------------
*  MERVA MESSAGE TYPE TABLE ENTRIES FOR MERVA LINK
*--------------------------------------------------------------------
         DSLMTT MTYPE=MCTL,MCB=EKAMCTL,                               *    [1]
               DESCR='Edit and View MERVA Link Control Fields'
         DSLMTT MTYPE=MCOV,MCB=EKAMCOV,NLIND=EKAEXIT,NL0=YES,MTGEN=NO, *    [2]
               DESCR='MERVA Link Cover MCB'
         DSLMTT MTYPE=ACMM,MCB=EKAACMM,                               *    [3]
               DESCR='MSC Main Menu'
         DSLMTT MTYPE=AC00,MCB=EKAAC00,                               *
               DESCR='MSC MERVA Operator Command Processing       '
         DSLMTT MTYPE=AC01,MCB=EKAAC01,                               *
               DESCR='MERVA Link List of Message Transfer Applications'
         DSLMTT MTYPE=AC02,MCB=EKAAC02,                               *
               DESCR='MERVA Link Display Specific ASP/MTP'
         DSLMTT MTYPE=AC03,MCB=EKAAC03,                               *
               DESCR='MERVA Link List of Partner MERVA Systems'
         DSLMTT MTYPE=AC04,MCB=EKAAC04,                               *
               DESCR='MERVA Link Display PT Header Information'
         DSLMTT MTYPE=ACHP,MCB=EKAACHP,                               *
               DESCR='MSC Environment Dependent Explanation'
         DSLMTT MTYPE=MHELP,MCB=EKAHELP,                              *
               DESCR='MERVA Link Help Information'
*--------------------------------------------------------------------
*  MERVA MESSAGE TYPE TABLE ENTRIES FOR THE MERVA LINK SAMPLE
*--------------------------------------------------------------------
         DSLMTT MTYPE=DEMO,MCB=EKADEMO,                               *    [4]
               DESCR='MERVA Link Demo Message for the Sample Scenario'
```

Figure 166. DSLMTTT Entries for MERVA Link

Notes:

[1] DSLMTT MTYPE=MCTL defines the MERVA Link Control Message.

The MERVA Link Control Message is assigned the message type MCTL. It is a mandatory resource owned by the MERVA Link. Message type MCTL is used by the MERVA Link for internal purposes.

The definition of the message type MCTL in the MERVA ESA MCB named EKAMCTL contains a screen and a net format section. The net format section defines the MERVA Link internal layout of an LC Control Message and the layout of the set of MERVA Link control fields.

Note: The net format section of EKAMCTL must not be altered.

The MERVA Link assumes that the message type MCTL is not yet used in your MERVA ESA system. However, if you have a need to change this message type, you must change it in the EKAPT macro, which is provided in the MERVA Link macro library, and regenerate your partner table.

There is no parameter in the partner table generation to modify the message type of the MERVA Link Control Message.

[2] DSLMTT MTYPE=MCOV defines the MERVA Link cover MCB.

The MERVA Link cover MCB EKAMCOV is used to display a message with control information and data of all applicable MERVA ESA components. If you define parameter MSGID=MCOV in the DSLFNTT entry of a MERVA ESA function, you will see the MERVA Link control fields first when you display a message in the queue corresponding to that function.

[3] DSLMTT MTYPE=ACxx defines one of the MERVA System Control Facility message types.

The user interface of the MERVA System Control Facility is implemented via the message types, ACMM, ACHP, AC00, AC01, AC02, AC03, and AC04. The corresponding MCBs are named EKAACMM, EKAACHP, EKAAC00, EKAAC01, EKAAC02, EKAAC03, and EKAAC04, respectively.

Note: The net format sections of these MCBs must not be altered.

[4] DSLMTT MTYPE=DEMO defines the MERVA Link Sample Message DEMO.

The MERVA Link message type DEMO is a part of the MERVA Link sample scenario. A message of this type is not used by MERVA Link for internal purposes.

Chapter 13. Field Definition Table (DSLFDTT)

The Field Definition Table describes all TOF fields used when processing MCBs of MERVA ESA.

Field Definition Macroinstructions

The following macros are used to define fields and subfields to create the Field Definition Table:

DSLLFDT Is used to generate the Field Definition Table header and must be the first macro specified for the Field Definition Table.

DSLLFLD Defines a field with its field characteristics. The definition should reflect those attributes that are most often used for the field in all messages in which the field occurs. For example, length of the field, or editing routine used for display devices. The attributes defined in the Field Definition Table are used if they are not overwritten in the MCB for a specific message type. A total of 32000 fields and subfields can be defined in the Field Definition Table.

DSLLSUBF Defines a subfield with its field characteristics. The subfield definitions of a field must immediately follow the DSLLFLD macro to which they belong. A field can have different layouts defined by the DSLLSUBF macros belonging to it. A subfield cannot be further subdivided into subfields but the main field can have the additional subfields. A total of 32000 fields and subfields can be defined in the Field Definition Table.

DSLLGEN Is the last macro of the Field Definition Table and completes the Field Definition Table generation. The Assembler END statement must immediately follow the DSLLGEN macro.

For details see the *MERVA for ESA Macro Reference*.

Coding the Field Definition Table (FDT)

MERVA ESA uses a Field Definition Table, which contains in the assembler source only the DSLLFDT and DSLLGEN macros and an assembler COPY statement for each network link. The copy members of each component finally contain the field definitions needed by the component.

Note: If you need to define your own fields, you should include these as a copy into the Field Definition Table.

When MERVA ESA and the SWIFT Link are installed, the Field Definition Table DSLFDTT looks as shown in Figure 167.

```
FDTT     TITLE 'MERVA FIELD DEFINITION TABLE'
DSLFDTT  DSLLFDT                                                    [1]
         COPY  DSLFDTTC           MERVA FIELD DEFINITIONS           [2]
         COPY  DWSFDTTC           SWIFT LINK FIELD DEFINITIONS      [3]
         COPY  ENLFDTTC           TELEX LINK FIELD DEFINITIONS      [4]
         COPY  EKAFDTTC           MERVA LINK FIELD DEFINITIONS      [5]
         COPY  EKAFDTSC           FMT FIELD DEFINITIONS             [6]
         DSLLGEN
         END
```

Figure 167. Field Definition Table for MERVA ESA and the SWIFT Link

Notes:

[1] DSLFDTT

The label of the DSLLFDT macro must be coded. It is used as the name of the Field Definition Table. If a name other than DSLFDTT is used, this name must also be defined in the MERVA ESA customizing parameters DSLPRM, such as that in the FDT parameter of the DSLPARM macro. MERVA ESA programs load the field definition table with the name found in DSLPRM.

[2] DSLFDTTC

This copy code describes all MERVA ESA fields used by:

- The MERVA ESA End-User Driver programs (DSLEUD)
- The MERVA ESA hardcopy printer program (DSLHCP)
- The MERVA ESA address expansion program (DSLCXT)
- The MERVA-MQI Attachment programs (DSLKQR, DSLKQS)

These fields are:

- MERVA ESA system fields. For example:
  - User identification
  - Processing function
  - Origin identification
- Data field for free format message
- Condition check field
- Exit check field
- Message identification field

The complete list of all fields used by MERVA ESA can be seen in the copy code DSLFDTTC.

[3] DWSFDTTC

This copy code describes all SWIFT Link fields. For example:

- The message header fields of SWIFT messages
- The trailer field of SWIFT messages
- All other fields of SWIFT messages
- The fields required for address expansion of the address fields of SWIFT messages
- The fields required for the on-line maintenance of the SWIFT Correspondents file
- The fields required for the on-line maintenance of the SWIFT Link Authenticator-Key file

The complete list of all fields used by the SWIFT Link can be seen in the copy code DWSFDTTC.

[4] ENLFDTTC

This copy code describes all Telex Link fields. For example:

- The telex header fields
- The Telex Link control fields
- The telex data fields
- The test-key calculation fields

[5] EKAFDTTC

This copy code describes all MERVA Link fields.

[6] EKAFDTSC

This copy code describes all fields for FMT/ESA with MERVA Link.

FDT Coding Examples

Coding Example of the SWIFT Field 39

The name of this field is SW39. This name is referenced in the MCB for the TYPE=MESSAGE and TYPE=NET parts. The names of the subfields are referenced in the TYPEs SCREEN, HARDCOPY, and SYSP.

```
*************************************************************************
*         SWIFT - FIELD 39 : AMOUNT SPECIFICATION                   *   [1]
*                            CONTAINS FOUR DATA AREAS (TEXT_LINES)  *
*                            USED IN  APPL  APDU-ID  MT   OPTIONS   *
*                                      F      01     700  A,B,C     *
*                                      F      01     705  A,B,C     *
*                                      F      01     707  A,B,C     *
*                                      F      01     710  A,B,C     *
*                                      F      01     720  A,B,C     *
*                                      F      01     740  A,B,C     *
*                                      F      01     747  A,B,C     *
*                                      F      01     769  C         *
*************************************************************************
SW39     DSLLFLD  OPTLIST=(A,B,C),OPTION=YES,CHECK=SEPR,SEPR=1001,      * [2]
               STRIP=YES,FSEP=YES
SW39S1B  DSLLSUBF LENGTH=(13,13,F),OFFSET=0,SEPR=STANDARD,MAND=YES        [3]
**
* POSITIVE TOLERANCE
SW39PT   DSLLSUBF FIELD=SW39,CHECK=SEPR,SEPR=1001,STRIP=YES               [4]
* NEGATIVE TOLERANCE
SW39NT   DSLLSUBF FIELD=SW39,CHECK=SEPR,SEPR=1001,STRIP=YES
```

Figure 168. Coding of the SWIFT Field SW39 in the Field Definition Table

Notes:

[1] These are the comments for field 39, its meaning and in which SWIFT messages it is used, for example, in MT 700 with either Option A, B, or C.

[2] DSLLFLD

This statement defines the main field SW39. The length and the data areas of field SW39 are checked by checking routine 1001, options are specified (OPTION=YES) and the option list contains the options allowed: A, B, and C (OPTLIST=(A,B,C)).

CHECK=SEPR shows that the separation routine specified (1001) is also used for checking.

STRIP=YES specifies that trailing blanks in the input are stripped out.

FSEP=YES specifies that the separation routine (1001) for the field is always called when field data is read or written.

[3] DSLLSUBF

The subfield SW39S1B extracts the first 13 characters from a data area of field SW39. The extraction is done by the MERVA ESA standard separation routine.

[4] DSLLSUBF

The subfields SW39PT and SW39NT are defined for main field SW39, separated by the SWIFT Link routine 1001.

Processing New or Changed FDTs

When existing fields are changed in the Field Definition Table, or new fields are added to the Field Definition Table, or a new Field Definition Table is created, the following should be considered:

- Each field requires a unique name.
- Define subfields of the field if necessary. If yes, decide whether the MERVA ESA standard separation is to be used or if a special separation module must be provided.
- Define the attributes of the fields:
  - Length
  - List of Field options
  - Number of data areas
  - Attributes MAND=, PERM=, and QUEUE=
- Define features of the data to be checked. Decide if the standard checking according to the characteristics as defined in the FDT or MCB are to be used or if a special checking module must be provided.
- Decide if the editing of the data is required. For amount fields you can use the editing routine DSLME901 or DSLME902. For other fields a special editing module must be provided.
- Decide if a default setting for the field is required, and if a special default setting module is required.
- Decide if expansion of the field is required, and if is a special expansion module is required.

A new or changed Field Definition Table must be assembled, and link-edited. If the name of the Field Definition Table is to be changed (default is DSLFDTT), the MERVA ESA customizing module DSLPRM must be changed too (FDT parameter of the DSLPARM macro), assembled and link-edited.

Note: The changed field characteristics are only available for fields in messages mapped after installation of the changed Field Definition Table. For further information, refer to “Processing New or Changed MCBs” on page 361.

MCBs referring to the new or changed fields must be installed.

All new or changed separation, edit, checking, default setting and expansion routines must be installed via the Message Format Service (MFS) program table (DSLMPTT). Usually the program DSLMMFS must be link-edited then.

You need not change DSLMPTT for the editing, checking, or default setting routine, which adheres to a special naming convention. The module is loaded dynamically, if the name of the exit routine is DSLMEnnn, DSLMCnnn, or DSLMDnnn, where nnn is the number of an exit. In this case you need not link-edit DSLMMFS.

MERVA Link Modifications in the Field Definition Table

All MERVA Link control fields must be specified in the DSLFDTT. The DSLFDTT entries used by MERVA Link are shown in Figure 169 on page 390. The copy book EKAFDTTC of the MERVA ESA macro library contains the macro instructions to generate these DSLFDTT entries.

The parameters of the MERVA Link control field definitions must not be altered. The names of these control fields may be modified as described below.

```
*-------------------------------------------------------------------
*        MERVA LINK CONTROL FIELDS
*-------------------------------------------------------------------
EKAAMCID DSLLFLD LENGTH=(16,16,F),PAD=' ' ASL MESSAGE ID (CONTENT ID)
EKAMSGID DSLLFLD LENGTH=(16,16,F),PAD=' ' MTL MESSAGE ID
EKAAMSID DSLLFLD LENGTH=(16,16,F),PAD=' ' IAM MESSAGE IDENTIFIER
EKAAMSEQ DSLLFLD LENGTH=(0,4,F)      MIP OUTBOUND MSG SEQUENCE NUMBER
EKAIMSEQ DSLLFLD LENGTH=(0,4,F)      MIP INBOUND MSG SEQUENCE NUMBER
EKAMIPID DSLLFLD LENGTH=(0,8,F)        MIP MESSAGE IDENTIFIER
*
EKAOAFFN DSLLFLD LENGTH=(0,60,V)     ORIGINATING APPL FREE FORM NAME
EKAONODE DSLLFLD LENGTH=(0,8,V)      ORIGINATING MT NODE NAME
EKAOAPPL DSLLFLD LENGTH=(0,8,V)      ORIGINATING APPLICATION NAME
EKAMTPNM DSLLFLD LENGTH=(0,8,V)      INTERNAL MSG TRANSFER PROCESS NAME
*
EKARAFFN DSLLFLD LENGTH=(0,60,V)     RECEIVING APPL FREE FORM NAME
EKARNODE DSLLFLD LENGTH=(0,8,V)      RECEIVING MT NODE NAME
EKARAPPL DSLLFLD LENGTH=(0,8,V)      RECEIVING APPLICATION NAME
*
EKAAMBSL DSLLFLD LENGTH=(0,256,V)    BUCKSLIP
EKAAMBS1 DSLLSUBF LENGTH=(0,22,V),OFFSET=0,SEPR=STANDARD
EKAAMBS2 DSLLSUBF LENGTH=(0,78,V),OFFSET=22,SEPR=STANDARD
EKAAMBS3 DSLLSUBF LENGTH=(0,78,V),OFFSET=100,SEPR=STANDARD
EKAAMBS4 DSLLSUBF LENGTH=(0,78,V),OFFSET=178,SEPR=STANDARD
*
EKAAMSUB DSLLFLD LENGTH=(0,60,V)     MESSAGE SUBJECT
*
EKATARQD DSLLFLD LENGTH=(0,1024,V)   APPLICATION REQUEST DATA
EKATARSD DSLLFLD LENGTH=(0,256,V)    APPLICATION RESPONSE DATA
EKATAACK DSLLFLD LENGTH=(0,256,V)    APPLICATION ACK DATA
EKATAMAC DSLLFLD LENGTH=(0,256,V)    APPLICATION MAC
EKATAPAC DSLLFLD LENGTH=(0,256,V)    APPLICATION PAC
*
EKASUBDT DSLLFLD LENGTH=(0,12,F)     SUBMIT DATE-TIME STAMP
EKASUBDA DSLLSUBF LENGTH=(0,6,F),OFFSET=0,SEPR=STANDARD
EKASUBTM DSLLSUBF LENGTH=(0,6,F),OFFSET=6,SEPR=STANDARD
EKASUBRC DSLLFLD LENGTH=(0,2,F)      SUBMIT RETURN CODE
EKASUBDC DSLLFLD LENGTH=(6,6,F),PAD=' ' SUBMIT DIAGNOSTIC CODE
*
EKADELDT DSLLFLD LENGTH=(0,12,F)     DELIVER DATE-TIME STAMP
EKADELDA DSLLSUBF LENGTH=(0,6,F),OFFSET=0,SEPR=STANDARD
EKADELTM DSLLSUBF LENGTH=(0,6,F),OFFSET=6,SEPR=STANDARD
EKADELRC DSLLFLD LENGTH=(0,2,F)      DELIVER RETURN CODE
EKADELDC DSLLFLD LENGTH=(6,6,F),PAD=' ' DELIVER DIAGNOSTIC CODE
*
```

Figure 169. MERVA Link DSLFDTT Entries (Part 1 of 2)

```
EKARECDT DSLLFLD LENGTH=(0,12,F)     RECEIPT DATE-TIME STAMP
EKARECDA DSLLSUBF LENGTH=(0,6,F),OFFSET=0,SEPR=STANDARD
EKARECTM DSLLSUBF LENGTH=(0,6,F),OFFSET=6,SEPR=STANDARD
EKARECRC DSLLFLD LENGTH=(0,2,F)      RECEIPT RETURN CODE
EKARECDC DSLLFLD LENGTH=(6,6,F),PAD=' ' RECEIPT DIAGNOSTIC CODE
*
EKAACKRQ DSLLFLD LENGTH=(0,1,F)      REQUEST FOR ACK (0, 1, 2)
EKAPRIOR DSLLFLD LENGTH=(0,1,F)      MESSAGE PRIORITY (H, N, L)
EKAPDUPM DSLLFLD LENGTH=(0,3,F)      PDM INDICATOR (PDM)
EKACLASS DSLLFLD LENGTH=(0,2,F)      MESSAGE CLASS (LC, LR, IP, CF, ..)
EKAWSIZE DSLLFLD LENGTH=(0,3,F),CHECK=NUMERIC MIP WINDOW SIZE
EKAAWQSN DSLLFLD LENGTH=(0,4,F)      MERVA QSN OF MSG IN ACK WAIT QUEUE
EKANETID DSLLFLD LENGTH=(0,1,F)      NET FORMAT IDENTIFIER
EKAACQNM DSLLFLD LENGTH=(0,8,F)      APPL CONTROL QUEUE NAME
*
EKARDATA DSLLFLD LENGTH=(0,256,V),DAMAX=32   REPORT DATA
EKARDAT1 DSLLSUBF LENGTH=(0,22,V),OFFSET=0,SEPR=STANDARD
EKARDAT2 DSLLSUBF LENGTH=(0,78,V),OFFSET=22,SEPR=STANDARD
EKARDAT3 DSLLSUBF LENGTH=(0,78,V),OFFSET=100,SEPR=STANDARD
EKARDAT4 DSLLSUBF LENGTH=(0,78,V),OFFSET=178,SEPR=STANDARD
*
EKADA    DSLLFLD LENGTH=(0,78,V)     AC01/2 DATA FIELD (ASP LIST LINE)
EKAAN    DSLLSUBF LENGTH=(0,8,F),OFFSET=0,SEPR=STANDARD ASP NAME
```

Figure 169. MERVA Link DSLFDTT Entries (Part 2 of 2)

The MERVA Link assumes that the names of the MERVA Link control fields are not yet used in your MERVA ESA system. However, if you must change one of these field names, you must also modify the EKAPT macro provided in the MERVA Link macro library, and regenerate your partner table. There is no parameter in the partner table generation to modify the MERVA Link Control Field names.

The EKAPT macro contains the short names (field name without the EKA prefix) of all MERVA Link control fields. You can change any of these short field names.

Note: The field name prefix **EKA** cannot be modified.

Part 3. Appendixes

Appendix. 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.

Programming Interface Information

This book is intended to help the customer to understand MERVA. This book primarily documents Product-Sensitive Programming Interface and Associated Guidance Information provided by MERVA.

General-Use Programming Interface allow the customer to write programs that obtain the services of MERVA.

However, this book also documents Product-Sensitive Programming Interface and Associated Guidance Information.

Product-Sensitive programming interfaces allow the customer installation to perform tasks such as diagnosing, modifying, monitoring, repairing, tailoring, or tuning of this IBM software product. Use of such interfaces creates dependencies on the detailed design or implementation of the IBM software product. Product-Sensitive programming interfaces should be used only for these specialized purposes. Because of their dependencies on detailed design and implementation, it is to be expected that programs written to such interfaces may need to be changed in order to run with new product releases or versions, or as a result of service.

Product-Sensitive Programming Interface and Associated Guidance Information is identified where it occurs, either by an introductory statement to a chapter or section by the following marking:

Product-Sensitive Programming Interface and Associated Guidance Information...

Trademarks

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

- Advanced Peer-to-Peer Networking
- AIX
- APPN
- C/370
- CICS
- CICS/ESA
- CICS/MVS
- CICS/VSE
- DB2
- Distributed Relational Database Architecture
- DRDA
- eNetwork
- IBM
- IMS/ESA
- Language Environment
- MQSeries
- MVS
- MVS/ESA
- MVS/XA
- OS/2
- OS/390
- RACF
- 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 are trademarks of Sun Microsystems, Inc. in the United States, other countries, or both.

UNIX is a registered trademark of The Open Group in the United States and other countries.

Microsoft, Windows, Windows NT, and the Windows logo are trademarks or registered 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 *SWIFT address*.

address expansion. The process by which the full name of a financial institution is obtained using the SWIFT 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 partener 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.

ASP. (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 SWIFT 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 SWIFT 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 SWIFT network. Also called a SWIFT 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 SWIFT user institution, or the letters "BIC" for institutions that are not SWIFT 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. SWIFT 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 SWIFT 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 SWIFT 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 SWIFT field 30, the field tag is **:30:**.

FIN. Financial application.

FIN-Copy. The MERVA component used for SWIFT 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 SWIFT 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 SWIFT 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 SWIFT 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 2MB 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
- 32KB

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 SWIFT 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 SWIFT 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 *SWIFT financial message.*

message body. The part of the message that contains the message text.

message category. A group of messages that are logically related within an application.

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

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

message sequence number (MSN). A sequence number for messages transferred by MERVA Link.

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

MFS. Message Format Service.

MIP. Message Integrity Protocol.

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

MPP. In IMS, message-processing program.

MQA. MQ Attachment.

MQ Attachment (MQA). A MERVA feature that provides message transfer between MERVA and a user-written MQI application.

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.

MRN. Message reference number.

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, SWIFT MT 195 could be used to request information about a SWIFT MT 100 (customer transfer). The SWIFT MT 100 (or at least its mandatory fields) is then nested in SWIFT 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 SWIFT

network service access point (NSAP). The endpoint of a network connection used by the SWIFT 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 SWIFT messages. With NOPROMPT mode, only the SWIFT 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 SWIFT field number to distinguish among different layouts for and meanings of the same field. For example, SWIFT 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 SWIFT network.

output message. A message that has been received from the SWIFT 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 SWIFT 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 SWIFT messages. With PROMPT mode, all the fields and tags are displayed for the SWIFT 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 queueing 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 SWIFT messages that are ready for sending to the SWIFT 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 SWIFT 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 signalling 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
- 32KB

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 SWIFT 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.

SWIFT address. Synonym for *bank identifier code*.

SWIFT 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.

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

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

SWIFT input message. A SWIFT message with an input header to be sent to the SWIFT network.

SWIFT link. The MERVA ESA component used to link to the SWIFT network.

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

SWIFT output message. A SWIFT message with an output header coming from the SWIFT network.

SWIFT system message. A SWIFT general purpose application (GPA) message or a financial application (FIN) message in SWIFT 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 systems 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: you read the message and confirm that you have done so
- Retype verification: 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).

W

Windows NT service. A type of Windows NT application that can run in the background of the Windows NT operating system even when no user is logged on. Typically, such a service has no user interaction and writes its output messages to the Windows NT event log.

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 sets. 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

MERVA ESA Components Publications

- *MERVA Automatic Message Import/Export Facility: User's Guide*, SH12-6389
- *MERVA Connection/NT*, SH12-6339
- *MERVA Connection/400*, SH12-6340
- *MERVA Directory Services*, SH12-6367
- *MERVA Extended Connectivity: Installation and User's Guide*, SH12-6157
- *MERVA Message Processing Client for Windows NT: User's Guide*, SH12-6341
- *MERVA-MQI Attachment User's Guide*, SH12-6714
- *MERVA Traffic Reconciliation*, SH12-6392
- *MERVA USE: Administration Guide*, SH12-6338
- *MERVA USE & Branch for Windows NT: User's Guide*, SH12-6334
- *MERVA USE & Branch for Windows NT: Installation and Customization Guide*, SH12-6335
- *MERVA USE & Branch for Windows NT: Application Programming Guide*, SH12-6336
- *MERVA USE & Branch for Windows NT: Diagnosis Guide*, SH12-6337
- *MERVA USE & Branch for Windows NT: Migration Guide*, SH12-6393
- *MERVA USE & Branch for Windows NT: Installation and Customization Guide*, SH12-6335
- *MERVA Workstation Based Functions*, SH12-6383

Other IBM Publications

- *CICS/ESA Version 4.1 Intercommunication Guide*, SC33-1181
- *CICS/ESA Version 4.1 Resource Definition Guide*, SC33-1166
- *High Level Assembler Language Reference*, SC26-4940
- *IMS/ESA Version 5 Administration Guide: Transaction Manager*, SC26-8014
- *MQSeries Application Programming Guide*, SC33-0807
- *MQSeries Application Programming Reference*, SC33-1673
- *MQSeries Command Reference*, SC33-1369
- *MQSeries Intercommunication*, SC33-1872
- *MQSeries System Management Guide*, GC34-5364
- *OS/390 MVS Planning: APPC/MVS Management*, GC28-1807

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

Numerics

A

B

C

D

E

F

G

H

I

J

K

L

M

N

O

P

Q

R

S

T

U

V

W

X

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 for ESA
Customization Guide
Version 4 Release 1

Publication No. SH12-6380-01

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

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

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

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

Please tell us how we can improve this book:

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

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

Name _____ Address _____

Company or Organization _____

Phone No. _____

Readers' Comments — We'd Like to Hear from You
SH12-6380-01

Cut or Fold
Along Line

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

PLACE
POSTAGE
STAMP
HERE

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

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

SH12-6380-01

Cut or Fold
Along Line

Program Number: 5648-B29

SH12-6380-01

Spine information:

MERVA for ESA

Customization Guide

Version 4
Release 1