MERVA for ESA

System Programming Guide

Version 4 Release 1

SH12-6366-01

MERVA for ESA

System Programming Guide

Version 4 Release 1

SH12-6366-01

Note!
Before using this information and the product it supports, be sure to read the general information under "Appendix E. Notices" on page 165.

Second Edition, May 2001

This edition applies to Version 4 Release 1 of IBM MERVA for ESA (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 describes the system programming interface of the IBM licensed program Message Entry and Routing with Interfaces to Various Applications for ESA Version 4 (abbreviated in this book to MERVA ESA).

This book assumes you have detailed knowledge of assembler language coding and are familiar with the layout of the MERVA ESA macros, which are described in the *MERVA for ESA Macro Reference*.

This book also assumes you are familiar with *MERVA for ESA Concepts and Components*, which describes the functions, services, and utilities supplied, as well as the message concept, queues, routing, message handling, and network links.

If you are using the SWIFT Link, this book assumes that you are familiar with SWIFT messages and SWIFT terminology as defined in the *S.W.I.F.T. User Handbook*, published by the Society for Worldwide Interbank Financial Telecommunication, s.c. in La Hulpe, Belgium.

If you are using the Telex Link, this book assumes that you are familiar with telex terminology as defined in the documentation provided by your local PTT[1].

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®.

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

Chapter 1. Types of MERVA ESA Application Programs

General-use programming interface

MERVA ESA application programs are programs designed to operate with MERVA ESA, using the direct or central services of MERVA ESA and its components. There are two types of MERVA ESA application programs:

1. Programs that link to DSLNUC using one of the following:
   - The MERVA ESA nucleus program table DSLNPTT. For example, the external network programs such as the SWIFT Link Line Server program DWSDGPA for the SWIFT network or the Telex Link program ENLSTPL use DSLNPTT.
   - The task server request table DSLNTRT. This is used by central service programs DSLQMGT.
   - The MERVA ESA operator command table DSLNCMT, these are the command execution routines for user defined operator commands.
2. Programs that are not linked to DSLNUC. These form three groups:
   - Programs working in the same region as DSLNUC (CICS only). These are CICS transaction programs such as DSLEUD, DSLHCP, DSLCXT, and the MERVA Link programs EKAAS10 and EKATR10.
   - Programs working in a different region from DSLNUC. These are the MERVA ESA batch programs DSLSDI, DSLSDO, and DSLSDY. In MERVA ESA operating under IMS, there are also the message processing programs (MPPs) DSLEUD, DSLHCP, DSLCXT, and the MERVA Link programs EKAAS10 and EKATR10.
   - Programs that are run before MERVA ESA is started. These are the MERVA ESA utility programs such as DSLQDSUT, DSLFLUT, DSLEBSPA (IMS only), and the SWIFT Link programs DWSCORUT, DWSCURUT, and DWSAUTLD.

The difference between the various kinds of MERVA ESA application programs is the way they access the services of MERVA ESA.

Programs used by a MERVA ESA application program must follow the same rules for using the services of MERVA ESA as the calling program. This must be considered by MERVA ESA MFS exit routines that can be called in any of the environments described here.

A program not linked to DSLNUC has only the services of DSLMMFS, DSLOMSG, DSLSRVP, DSLTRAP, DSLTOFSV, and DSLFLVP directly available (direct MERVA ESA services). The services of DSLQMGT, DSLNUSR, DSLNCS, DSLNMOP and, for the SWIFT Link, DWSAUTP are only available via the MERVA ESA intertask communication facility DSLNIC (MERVA ESA central services). All functions can be carried out except the initialization and termination of the central service functions handled exclusively by DSLNUC and the programs linked to DSLNUC.

A MERVA ESA application can process the messages of any queue defined in the MERVA ESA function table. However, if queues are processed that are used by the programs DSLHCP, DSLSDI, DSLSDO, DSLSDY, or by programs of the MERVA-MQI Attachment, SWIFT Link, Telex Link, or MERVA Link, they might

interfere with the processing of these programs. The queues used by the latter programs are defined in the customization parameter modules of the components:

DSLKPROC MERVA-MQI Attachment

DWSPRM SWIFT Link

DWSLTT SWIFT Link to the SWIFT network

ENLPRM Telex Link

EKAPT MERVA Link

EKASPRM FMT/ESA with MERVA Link

To see the list layout, write a sample program containing the MF=L, PS, or TS call for the appropriate MERVA ESA macro (for example, DSLMFS).

The following chapters show how the services of MERVA ESA can be used by user-written programs.

End of General-use programming interface

Chapter 2. Buffer Standard of MERVA ESA

General-use programming interface

Whenever a MERVA ESA program uses a buffer for input or output, the MERVA ESA standard buffer must be used.

A program that fills data into the buffer of the caller needs to know how large the buffer is so as not to overlay storage behind the buffer.

Also, after getting back control, the calling program wants to know how much data is returned.

Throughout MERVA/370 V2 all data buffers had the following layout:

```
BUFFER   DS    0H            MERVA/370 V2 buffer
BUFLENG  DC    H'BL'         buffer length BL including header of 8 bytes
         DC    H'0'          reserved
DATALENG DC    H'DL+4'       data length DL contained in DATAAREA
*                            including 4 for the two halfwords
         DC    H'0'          reserved
DATAAREA DS    (BL-8)C       data area in the length BL-8
*                            filled as indicated by DATALENG
```

Because the field BUFLENG is a halfword, the size of a MERVA/370 V2 buffer was limited to 32767 bytes (32KB – 1 byte).

For MERVA ESA, the buffer standard is extended to support a buffer length up to 16777215 bytes (16MB – 1 byte). The maximum message length supported by MERVA ESA is 2MB (2097152 bytes).

Throughout MERVA ESA, data buffers that are larger than 32KB have the following layout:

```
BUFFER   DS    0F            MERVA ESA buffer
BUFLEN   DS    0F            four byte buffer length field
         DC    X'80'         indication
         DC    AL3(BL)       buffer length up to 16MB-1

DATALENG DS    0F            four byte data length field if DL ≥ 32KB - 4
         DC    X'80'         indication
         DC    AL3(DL+4)     data length up to BL - 8

         ORG   DATALENG      two byte data length field if DL < 32KB - 4
         DC    H'DL+4'
         DC    H'0'

DATAAREA DS    (BL-8)C       data area in the length BL-8
*                            filled as indicated by DATALENG
```

For values less than or equal to 32767, the first two bytes are used to store the value, and the remaining two bytes are not used. This implies that the value of the high-order bit (bit 0) is always zero. The format of the length field is compatible to MERVA/370 V2.

For values above 32767, four bytes are used to store the length value. To indicate that four bytes are used to store the length value, bit 0 (high-order bit) is set to one.

MERVA ESA provides service functions to inspect or set the length values according to the MERVA ESA buffer standard. Application programs can use these service functions to process the length fields without having to deal with the indicator bit or formats.

DSLAPBGB Get the buffer length

DSLAPBSB Set the buffer length

DSLAPBGD Get the actual data length

DSLAPBSD Set the actual data length

For detailed information on how to use this service, refer to *MERVA for ESA Application Programming Interface Guide*

End of General-use programming interface

Chapter 3. Using the MERVA ESA Communication Area (DSLCOM)

The MERVA ESA communication area (DSLCOM) is used by all MERVA ESA programs, including user-written programs. Any user-written program must use the DSLCOM in one of the following ways:

- If the user-written program is link-edited to a MERVA ESA program, the user-written program must contain the following statement:

  ```
  DSLCOM DSECT=YES
  ```

  The DSLCOM of the MERVA ESA program is used, and the MERVA ESA program provides for the correct contents of the DSLCOM. If the DSLCOM is used by a program link-edited to DSLNUC, the parameter NUC=YES must be specified with the DSLCOM macro.
- A user-written program that is not link-edited to a MERVA ESA program is called a *main MERVA ESA application*. Such programs must set up the DSLCOM themselves, and must contain the following statement:

  ```
  DSLCOM DSECT=NO
  ```

  A reentrant CICS transaction or another reentrant program maps the DSLCOM within a dummy section (DSECT), while a non-reentrant program maps the fields of DSLCOM in a control section (CSECT). A serially reusable program must clear and initialize the DSLCOM before using it.

The following description shows how the DSLCOM fields are filled, which fields are provided by the various MERVA ESA programs and which fields must be provided by user-written main MERVA ESA applications, depending on the environment.

Filling the Fields of DSLCOM

This section explains how DSLCOM fields are filled by a main MERVA ESA program.

The first 8 bytes of DSLCOM contain the identifier. If the storage for DSLCOM is addressed via a DSECT, the following instruction should be executed after the GETMAIN:

```
         MVC   DSLCOM(8),=C'*DSLCOM*'      LABEL
```

The field COMSRVPA contains the address of the MERVA ESA general service program DSLSRVP. It is filled as follows (this address must be available before DSLSRVP is used the first time):

```
         MVC   COMSRVPA,=V(DSLSRVP)        DSLSRVP ENTRY POINT
```

The field COMPRMA contains the address of the MERVA ESA customizing parameter module DSLPRM. It is filled as follows:

```
         DSLSRV TYPE=LOAD,MODULE='DSLPRM',MF=E  LOAD DSLPRM
 *       CHECK FOR SUCCESSFUL LOAD HERE!
         MVC   COMPRMA,SRVENTRY                 DSLPRM ENTRY POINT
```

The field COMTSVA contains the address of the MERVA ESA TOF supervisor program DSLTOFSV. It is filled as follows:

```
         DSLSRV TYPE=LOAD,MODULE='DSLTOFSV',MF=E   LOAD DSLTOFSV
*        CHECK FOR SUCCESSFUL LOAD HERE!
         MVC   COMTSVA,SRVENTRY                    DSLTOFSV ENTRY POINT
```

The field COMFDTA contains the address of the MERVA ESA Field Definition Table. The name of this table is found in DSLPRM. COMFDTA is filled as follows:

```
         LA    R4,NPFDT                            NAME OF FDT
         DSLSRV TYPE=LOAD,MODULE=(R4),MF=E         LOAD FDT
*        CHECK FOR SUCCESSFUL LOAD HERE!
         MVC   COMFDTA,SRVENTRY                    ENTRY POINT OF FDT
```

The field COMOMSGA contains the address of the MERVA ESA Operator Messages Retrieval program DSLOMSG. It is filled as follows:

```
         MVC   COMOMSGA,=V(DSLOMSG)                DSLOMSG ENTRY POINT
```

The field COMMSGTA contains the address of the MERVA ESA Message table. The name of this table is found in DSLPRM. COMMSGTA is filled as follows:

```
         LA    R4,NPMSG                            NAME OF MSGT
         DSLSRV TYPE=LOAD,MODULE=(R4),MF=E         LOAD MSGT
*        CHECK FOR SUCCESSFUL LOAD HERE!
         MVC   COMMSGTA,SRVENTRY                   ENTRY POINT OF MSGT
```

The field COMMFSA contains the address of the MERVA ESA Message Format Services program DSLMMFS. It is filled as follows:

```
         DSLSRV TYPE=LOAD,MODULE='DSLMMFS',MF=E    LOAD DSLMMFS
*        CHECK FOR SUCCESSFUL LOAD HERE!
         MVC   COMMFSA,SRVENTRY                    ENTRY POINT OF DSLMMFS
```

The field COMMTTA contains the address of the MERVA ESA Message Type table. The name of this table is found in DSLPRM. COMMTTA is filled as follows:

```
         LA    R4,NPMTT                            NAME OF MTT
         DSLSRV TYPE=LOAD,MODULE=(R4),MF=E         LOAD MTT
*        CHECK FOR SUCCESSFUL LOAD HERE!
         MVC   COMMTTA,SRVENTRY                    ENTRY POINT OF MTT
```

The field COMFNTA contains the address of the MERVA ESA Function table. The name of this table is found in DSLPRM. COMFNTA is filled as follows:

```
         LA    R4,NPFNT                            NAME OF FNT
         DSLSRV TYPE=LOAD,MODULE=(R4),MF=E         LOAD FNT
*        CHECK FOR SUCCESSFUL LOAD HERE!
         MVC   COMFNTA,SRVENTRY                    ENTRY POINT OF FNT
```

The field COMFLVPA contains the address of the MERVA ESA File Service program DSLFLVP. It is filled as follows:

```
        DSLSRV TYPE=LOAD,MODULE='DSLFLVP',MF=E    LOAD DSLFLVP
*       CHECK FOR SUCCESSFUL LOAD HERE!
        MVC   COMFLVPA,SRVENTRY                   ENTRY POINT OF DSLFLVP
```

The field COMFLTTA contains the address of the MERVA ESA File table. The name of this table is found in DSLPRM. COMFLTTA is filled as follows:

```
         LA    R4,NPFLT                            NAME OF FLT
         DSLSRV TYPE=LOAD,MODULE=(R4),MF=E         LOAD FLT
*        CHECK FOR SUCCESSFUL LOAD HERE!
         MVC   COMFLTA,SRVENTRY                    ENTRY POINT OF FLT
```

The field COMMFSMA contains the address of the error message buffer of the MERVA ESA Message Format Service permanent storage. COMMFSMA is filled by DSLMMFS.

The field COMTRAPA contains the address of the MERVA ESA trace program. It is filled as follows:

```
         MVC   COMTRAPA,=V(DSLTRAP)              ENTRY POINT OF DSLTRAP
```

The field COMTRATA contains the address of the MERVA ESA Trace table. It is exclusively used by the DSLTRAP program.

The field COMTRAST contains the MERVA ESA trace status. It is exclusively used by the DSLTRAP program.

The field COMTRASF contains the MERVA ESA debugging trace flags for DSLMMFS and DSLTOFSV and are controlled by these. For details, refer to the *MERVA for ESA Diagnosis Guide*.

The MERVA ESA trace parameter list starting at label COMTRAPL is filled by the callers of the MERVA ESA trace service as needed.

The field COMNICPL contains the address of the MERVA ESA Intertask communication parameter list. The intertask communication parameter list must not be moved as long as the intertask communication session is allocated. It is filled as follows:

```
NICPLST  DSLNIC MF=L                     DSLNIC PARAMETER LIST
         .
         .
         LA    R15,NICPLST               ADDRESS OF DSLNIC PARM LIST
         ST    R15,COMNICPL              ..TO DSLCOM
```

The field COMTUCBA contains the address of the MERVA ESA MFS Terminal User Control Block. It is filled as follows:

```
TUCBTUCB DSLMFS MF=TUCB                  TERMINAL USER CONTROL BLOCK
         .
         .
         LA    R15,TUCBTUCB              ADDRESS OF TUCB
         ST    R15,COMTUCBA              ..TO DSLCOM
```

The field COMMTBA contains the address of the MERVA ESA MFS Load table. It is filled only by DSLMMFS.

The field COMERRA contains the address of the MERVA ESA End-User Driver Error program. It is filled only by DSLEUD.

The field COMJRNPA contains the address of the MERVA ESA Journal program. It is filled only by DSLNUC.

The field COMNCSA contains the address of the MERVA ESA Command Server program. It is filled only by DSLNUC.

The field COMNMOPA contains the address of the MERVA ESA Nucleus Operator Interface program. It is filled only by DSLNUC.

The field COMQMGTA contains the address of the MERVA ESA Queue Management program. It is filled only by DSLNUC.

The field COMTIMPA contains the address of the MERVA ESA Timer Service program. It is filled as follows:

```
MVC   COMTIMPA,=V(DSLTIMP)      ENTRY POINT OF DSLTIMP
```

The field COMTIME contains the MERVA ESA startup time. It is filled only by DSLNUC.

The fields COMSTAT0 and COMSTAT1 contain statuses of the complete MERVA ESA. They are filled only by DSLNUC.

The field COMDSNL points to an address list that contains the address of DWSPRM in the second fullword when DWSDGPA for the SWIFT network is started. DWSDGPA fills it only in the DSLCOM of DSLNUC, indicating that DWSDGPA is started.

The field COMRECON is reserved for use by MERVA ESA.

The field COMDTNL is reserved for use by the Telex Link. When the Telex Link is started, the address of the Telex Link storage area ENLSTPST is written into this field.

The field COMDWNN is reserved for use by national network programs.

The three fullwords following COMDWNN are reserved for use by IBM, that is, whenever a correction of a MERVA ESA or other program is made, these fullwords can be used by IBM. These fields must never be used by user-written programs.

The fields COMUSER1, COMUSER2, COMUSER3, and COMUSER4 are for use by user-written programs. IBM will not use these fields.

The field COMPCBLA is used only in MERVA ESA running under IMS and contains the address of the PCB address list that is given to a program called by IMS in register 1. It is filled at entry to the program as follows:

```
ST    R1,COMPCBLA               IMS PCB LIST ADDRESS
```

The fields COMTCAA, COMCSAA, and COMTCTUA are reserved for internal use by MERVA ESA.

The field COMCWAA is used only in MERVA ESA running under CICS by programs running as a CICS task (DSLNUC, DSLEUD, DSLHCP, and DSLCXT). COMCWAA contains the address of the DSLCWA, that is the area used in the CICS common work area for MERVA ESA.

This address is filled at entry to the program as follows:

```
EXEC CICS ADDRESS CWA(R5)
AL    R5,NPCWAOFF         MERVA CWA OFFSET TO CWA ADDRESS
ST    R5,COMCWAA          SAVE CWA ADDRESS
```

The fields COMEISTG, COMEIB, and COMCOM are also used only in MERVA ESA running under CICS by programs running as a CICS task (DSLNUC, DSLEUD, DSLHCP, and DSLCXT). COMEISTG contains the address of the CICS Exec Interface Storage. COMEIB contains the address of the CICS Exec Interface Block. COMCOM is used only by DSLSRVP. COMEISTG and COMEIB are filled at entry to the program as follows:

```
ANYPGM   DFHEIENT CODEREG=(R10,R11),DATAREG=(R8),EIBREG=(R9)
         ST    R9,COMEIB          EXEC INTERFACE BLOCK ADDR TO DSLCOM
         ST    R8,COMEISTG        EXEC INTERFACE STORAGE ADDR TO DSLCOM
```

The parameter list of DSLSRVP, which follows these fields in DSLCOM, need not be initialized. This parameter list allows all programs to use the services of DSLSRVP.

Most MERVA ESA programs use general register 12 to address the MERVA ESA program communication area (DSLCOM). In all MFS programs and exits, this register is used for this purpose.

The Use of the DSLCOM Fields by the MERVA ESA Programs

Figure 1 on page 10 shows which MERVA ESA program sets which field of DSLCOM. Fields set by MERVA programs can be referred to in user-written programs operating in the same environment. If the user-written program is the main program (for example, a CICS task or a batch program), Figure 1 shows which fields of DSLCOM must be filled by the program to be able to use MERVA ESA services.

No user-written program can act like DSLNUC. There are reserved fields in DSLCOM that must never be filled by a user-written main program.

How to fill the DSLCOM fields is described in "Filling the Fields of DSLCOM" on page 5.

The first column in the table shows the name of the field of DSLCOM in the order in which they appear in the dummy or control section of DSLCOM. The other columns show the use of the DSLCOM fields for those programs that can operate together with user-written programs.

The following keywords are used within the table:

ALL This field is used by MERVA ESA running under both CICS and IMS.

CICS This field is used by MERVA ESA running under CICS only (MVS™ and VSE).

IMS This field is used only by MERVA ESA running under IMS.

NO This field is not used by this program.

Program	DSLNUC	DSLEUD	DSLHCP	DSLCXT	DSLSDI	DSLSDO	DSLSDY
Field							
COMPRM	ALL	ALL	ALL	ALL	ALL	ALL	ALL
COMSRVPA	ALL	ALL	ALL	ALL	ALL	ALL	ALL
COMTSVA	ALL	ALL	ALL	ALL	ALL	ALL	ALL
COMFDTA	ALL	ALL	ALL	ALL	ALL	ALL	ALL
COMOMSGA	ALL	ALL	ALL	ALL	ALL	ALL	ALL
COMMSGTA	ALL	ALL	ALL	ALL	ALL	ALL	ALL
COMMFSA	ALL	ALL	ALL	ALL	ALL	ALL	ALL
COMMTTA	ALL	ALL	ALL	ALL	ALL	ALL	ALL
COMFNTA	ALL	NO	NO	NO	NO	NO	NO
COMFLVPA	NO	ALL	ALL	ALL	NO	NO	ALL
COMFLTTA	NO	ALL	ALL	ALL	NO	NO	ALL
COMMFSMA	NO	ALL	NO	NO	NO	NO	NO
COMTRAPA	ALL	ALL	ALL	ALL	ALL	ALL	ALL
COMTRAPL	ALL	ALL	ALL	ALL	ALL	ALL	ALL
COMNICPL	NO	ALL	ALL	ALL	ALL	ALL	ALL
COMTUCBA	NO	ALL	ALL	ALL	NO	NO	ALL
COMMTBA	NO	ALL	NO	NO	NO	NO	NO
COMERRA	NO	ALL	NO	NO	NO	NO	NO
COMJRNPA	ALL	NO	NO	NO	NO	NO	NO
COMNCSA	ALL	NO	NO	NO	NO	NO	NO
COMNMOPA	ALL	NO	NO	NO	NO	NO	NO
COMQMGTA	ALL	NO	NO	NO	NO	NO	NO
COMTIMPA	ALL	NO	NO	NO	NO	NO	NO
COMTIME	ALL	NO	NO	NO	NO	NO	NO
COMSTAT0	ALL	NO	NO	NO	NO	NO	NO
COMSTAT1	ALL	NO	NO	NO	NO	NO	NO
COMDSNL	ALL	NO	NO	NO	NO	NO	NO
COMRECON	ALL	NO	NO	NO	NO	NO	NO
COMDTNL	ALL	NO	NO	NO	NO	NO	NO
COMPSB	NO	IMS	IMS	IMS	NO	NO	NO
COMPCBLA	IMS	IMS	IMS	IMS	NO	NO	NO
COMCWAA	CICS	CICS	CICS	CICS	NO	NO	NO
COMEISTG	CICS	CICS	CICS	CICS	NO	NO	NO
COMEIB	CICS	CICS	CICS	CICS	NO	NO	NO

Figure 1. The Use of DSLCOM Fields by the MERVA ESA Programs

Notes:

1. The DSLCOM of DSLNUC is available to programs linked to DSLNUC:
   - When the parallel processing is not used, there is only one DSLCOM available. Register 12 contains the address of this DSLCOM.
   - When the parallel processing is used, each separate nucleus server task has its own DSLCOM which is different from the DSLCOM of the nucleus server. Register 12 contains the address of the DSLCOM of the nucleus server.

   The field COMNUCOM in the DSLCOM of each server task contains the address of the DSLCOM of the nucleus server.
2. The DSLCOM of DSLEUD is available to programs linked to DSLEUD, which is not linked to DSLNUC. It is a CICS transaction or an IMS MPP, and uses a screen terminal. DSLHCP is similar but uses a printer terminal, and DSLCXT does not use a terminal.

3. The DSLCOM of DSLSDI, DSLSDO, and DSLSDY is an example of a program not linked to DSLNUC and running in a region or partition other than DSLNUC.

Chapter 4. Using General Services (DSLSRV)

The MERVA ESA service program DSLSRVP provides several frequently used system services to MERVA ESA programs. The services provided are:

DATETIME Get the actual date and time in various formats

DEQ Release exclusive control of a resource

DUMP Get a dump of the tasks storage

ENQ Request exclusive control of a resource

FREEMAIN Free main storage

GETMAIN Get main storage

LOAD Load a module into virtual storage

POST Post an event control block (ECB)

RELEASE/DELETE
Release/Delete a loaded module from virtual storage

SNAP Get a partial dump of a tasks storage

WAIT Wait on one or more events.

The advantage of using this program is that the request is independent of the environment in which the calling program works. DSLSRVP issues the environment-dependent request to fulfill the caller's request.

The program is called using the calling macro DSLSRV, which is described in the *MERVA for ESA Macro Reference*. DSLSRV is also used to map the parameter list. The parameter list of DSLSRVP is contained in DSLCOM.

The calling program provides the parameters required for a specific service request. Except for the list header, the parameter list is not cleared between requests so that data entered in the list remains there unless it is overwritten by the next request.

Setup for DSLSRVP

The setup for using DSLSRVP is shown below:

```
DSLCOM
⋮

L     R12,=A(DSLCOM)
USING DSLCOM,R12
⋮

MVC   COMSRVPA,=V(DSLSRVP)
⋮
```

In this example, DSLCOM is allocated as a data area. With the USING statement, DSLCOM (and, therefore, implicitly the parameter list for DSLSRVP) is addressed. The address of DSLSRVP is moved into the field COMSRVPA.

DATETIME Request

The following is an example of a DATETIME request.

```
        DSLSRV TYPE=DATETIME,DATMASK='DD',TIMMASK='HHMM'
```

After successful completion of DSLSRVP, the requested date is found in the field SRVDATEX, and the requested time is found in the field SRVTIMEX. Both fields are padded with blanks if the date or time mask is shorter than 8 characters.

DEQ Request

The following is an example of a DEQ request. The specification of the QNAME and RNAME parameters must be identical to the specification in the ENQ request.

```
        LA    R3,L'UXRESID        LENGTH OF RESOURCE IDENTIFIER
        DSLSRV TYPE=DEQ,QNAME='DSLQUEUE',RNAME=UXRESID,SIZE=(R3)

UXRESID  DC  CL24'RESOURCE IDENTIFIER'
```

DUMP Request

The following is an example of a DUMP request. The dump identification can either be contained in a halfword or in a register.

```
        DSLSRV TYPE=DUMP,DUMPID=99
  or
        LA    R8,SRVDN13          DUMP ID
        DSLSRV TYPE=DUMP,DUMPID=(R8)
```

ENQ Request

The following is an example of an ENQ request. The name of the resource is specified in the field UXRESID.

```
        LA    R3,L'UXRESID        LENGTH OF RESOURCE IDENTIFIER
        DSLSRV TYPE=ENQ,QNAME='DSLQUEUE',RNAME=UXRESID,SIZE=(R3)

UXRESID  DC  CL24'RESOURCE IDENTIFIER'
```

FREEMAIN Request

The following is an example of a FREEMAIN request.

```
        L     R2,EUDSTORA
        DSLSRV TYPE=FREEMAIN,ADSTOR=(R2)
```

GETMAIN Request

The following is an example of a GETMAIN request.

```
        LH    R4,=LAREA              LENGTH OF THE STORAGE
        DSLSRV TYPE=GETMAIN,SIZE=(R4),INCHAR=3F
        L     R3,SRVSADDR            GET STORAGE ADDRESS
```

After successful completion of DSLSRVP, the address of the obtained storage is found in the field SRVSADDR.

The following is an example of a GETMAIN request to acquire storage above the 16MB line.

```
L     R4,=F'250000'            LENGTH OF STORAGE
DSLSRV TYPE=GETMAINA,SIZE=(R4)
L     R3,SRVSADDR              GET STORAGE ADDRESS
```

The following is an example of a GETMAIN request to acquire CICS shared storage above the 16MB line. For non-CICS environment, this call is identical to a TYPE=GETMAINA call.

```
L     R4,=F'250000'            LENGTH OF STORAGE
DSLSRV TYPE=GETMAINS,SIZE=(R4)
L     R3,SRVSADDR              GET STORAGE ADDRESS
```

LOAD Request

The following is an example of a LOAD request.

```
DSLSRV TYPE=LOAD,MODULE='DSLAPI'
L     R15,SRVENTRY             GET ENTRY POINT ADDRESS
```

After successful completion of DSLSRVP, the load address of the loaded module is found in the field SRVMADDR, and its entry point is found in the field SRVENTRY. This latter field must be used when the module loaded is being given control with a BALR instruction.

POST Request

The following is an example of a POST request. This request can only be used to post an ECB in the same region or partition.

```
DSLSRV TYPE=POST,ECB=(R9)
```

RELEASE/DELETE Request

The following are examples for a DELETE and a RELEASE request. The request types DELETE and RELEASE are synonymous.

```
DSLSRV TYPE=DELETE,MODULE='DWSPRM'
 or
DSLSRV TYPE=RELEASE,MODULE='DSLMMFS'
```

SNAP Request

The following is an example for obtaining a dump of main storage with specified starting point and length.

```
L     R5,STORADDR              BEGIN OF STORAGE FOR SNAP
L     R6,STORLENG              LENGTH OF STORAGE FOR SNAP
DSLSRV TYPE=SNAP,ADSTOR=(R5),SIZE=(R6)
```

WAIT Request

The following is an example of a WAIT request. The WAIT can be issued for an ECB list or for a single ECB.

```
        DSLSRV TYPE=WAIT,ECBLIST=(R3)
or
        DSLSRV TYPE=WAIT,ECB=(R5)
```

Information Returned

Some service requests return data to the calling program. They are found in specific fields of the parameter list. The following table shows which service request provides data in which field.

Request	Fields
ALL	SRVRC SRVRSNCD
DATETIME	SRVDATE SRVTIME SRVDATEX SRVTIMEX
GETMAIN	SRVSADDR SRVSIZE
LOAD	SRVMADDR SRVSIZE SRVENTRY SRVSADDR

'SRV' is the prefix used in the DSLSRV macro mapping the parameter list.

The following table shows which fields contains data for the calling program.

Field	Request	Data Returned	Format of Data
SRVDATE	DATETIME	Date packed decimal	0CYYDDDF
SRVDATEX	DATETIME	Date zoned	depends on mask
SRVENTRY	LOAD	Address of entry point	fullword
SRVMADDR	LOAD	Address of load point	fullword
SRVRC	ALL	Return code	halfword
SRVRSNCD	ALL	Reason code	halfword
SRVSADDR	GETMAIN	Address of storage area	fullword
SRVSIZE	LOAD	Size of load module	fullword
SRVSIZE	GETMAIN	Actual size	fullword
SRVTIME	DATETIME	Time packed decimal	0HHMMSSF
SRVTIMEX	DATETIME	Time zoned	depends on mask

Chapter 5. Using the Operator Message Retrieval Program (DSLOMS)

The DSLOMS macro serves two purposes:

1. It maps the parameter list of DSLOMSG and generates a parameter/substitution list for DSLOMSG.
2. It calls the message program DSLOMSG.

Further details of the DSLOMS macro are given in the *MERVA for ESA Macro Reference*.

Mapping the Parameter List of DSLOMSG

A main purpose of mapping the parameter list of DSLOMSG is defining the substitution parameters for the messages retrieved from the message table, for example, DSLMSGT.

Defining a halfword reason code, an 8-byte name with trailing blank omission, and a 20-byte text without trailing blank omission requires the following DSLOMS macro:

```
OMSPL   DSLOMS (REASON,H),(NAME,8C,S),(TEXT,20C),MF=L
```

Retrieving a Message

When calling DSLOMSG for retrieval of a message, DSLCOM must be addressable and the address of DSLOMSG must be available in the field COMOMSGA. The following example assumes that the field COMMSGTA contains the address of the message table (such as DSLMSGT), and the message USR001I requires the reason code as substitution item. With the DSLOMS parameter list, the message USR001I can look as follows (coded in DSLMSGT):

```
USR001I DSLMSG 'User program failed, reason is @0'
```

@0 is the substitution item which refers to the field REASON in the DSLOMS parameter list here.

Calling DSLOMSG looks as follows:

```
        MVC   REASON,USERREAS         REASON CODE FOR MESSAGE
        DSLOMS MSGID='USR001I',BUF=MSGBUFFR,   GET..                  *
        TABLE=COMMSGTA,MF=(E,OMSPL)   ..THE MESSAGE
```

DSLOMSG replaces the substitution item @0 in the message USR001I with the value found in the DSLOMS parameter list field REASON. This includes the conversion from the binary value to printable.

Chapter 6. Using the TOF Supervisor (DSLTSV)

This chapter describes the services that you can request from the TOF Supervisor, DSLTOFSV, and gives examples of how you can specify your requests using the TOF Supervisor macro, DSLTSV. The information required to use the call interface is supplied with the description of the DSLTSV macro. For complete information on the DSLTSV macro parameters, and the contents of the parameters to be supplied in the TOF parameter list for the call interface, refer to the *MERVA for ESA Macro Reference*.

The tokenized form (TOF) contains the data and all the fields required for processing MERVA ESA messages. It can be accessed only via the TOF Supervisor. You can use the TOF for your own programs or user exits.

The following services of DSLTOFSV are available:

- Creating a new TOF
- Writing data to the TOF
- Adding data areas to the TOF
- Reading data from the TOF
- Deleting data from the TOF
- Accessing fields in the TOF
- Checking fields in the TOF
- Expanding fields in the TOF
- Initializing fields in the TOF
- Adding a nesting identifier to the TOF
- Compressing the TOF into a buffer, or merging the TOF with another TOF, or both
- Joining a split TOF into a single TOF
- Freeing the data part of a split TOF.

You request these services by calling the TOF Supervisor DSLTOFSV. If you use the macro DSLTSV, it handles the layout of the DSLTSV parameter list and puts each parameter in the correct position when setting up the call.

To use the call interface, you must supply TSVPARMS with enough information before executing the call. Most parameters of TSVPARMS remain unchanged unless you change them explicitly, for example, addresses of valid buffers required by DSLTOFSV: TOF (TSVPADDR), TOF working buffer (TSVPWORK), MFS permanent storage (TSVPENVR), and the MERVA ESA communication area (TSVPDSLC). These addresses should be set once before the first call to DSLTOFSV. All buffers supplied must have the standard MERVA ESA buffer layout.

In general, if data is read, written, or changed in the TOF then this data is assigned to a field, the reference of which must be supplied. You can process data either for a main field or a subfield when not mentioned otherwise. If you want to process a subfield you must supply the name of the subfield in the field reference. DSLTOFSV finds the name of the main field from the FDT and invokes the separation routine assigned.

Using the Call Interface or the TOF Supervisor Macro DSLTSV

To use the call interface, use the instruction:

```
[label]  CALL  DSLTOFSV,(ATSVPARM)
```

ATSVPARM is the address of the parameter list TSVPARMS, shown here:

```
TSVPARMS DSECT                    TOF PARAMETER LIST
TSVPRC   DC    AL2(0)             RETURNCODE
TSVPRSC  DC    AL2(0)             REASONCODE
TSVPFTYP DC    CL4' '             FUNCTION FOR TOF REQUEST
                                  ADDA = ADD DATA AREA
                                  ADNI = ADD NESTING ID
                                  CHK  = CHECK
                                  COMP = COMPRESS
                                  DELE = DELETE
                                  EXPA = EXPAND
                                  FREE = Free TOF's Data Part
                                  INIT = INIT
                                  JOIN = JOIN Index and Data TOF
                                  READ = READ
                                  WRT  = WRITE
                                  TNEW = MAKE A NEW TOF
                                  MERG = MERGE TWO TOFS
                                  SHOT = DIAGNOSTICS
TSVPCURR DS    0CL15              CURRENT TOF POSITION
TSVPCUNI DC    AL1(0)             NESTING IDENTIFIER
TSVPCUFG DC    AL1(0)             FIELD GROUP INDEX
TSVPCURS DC    AL2(0)             REPEATABLE SEQUENCE INDEX
TSVPCUFN DC    CL8' '             FIELD NAME
TSVPCUDA DC    AL2(0)             DATA AREA INDEX
TSVPCUOM DC    CL1' '             OPTION FIELD MODIFIER
         DC    CL1' '             RESERVED
TSVPNEXT DS    0CL15              NEXT TOF POSITION
TSVPNENI DC    AL1(0)             NESTING IDENTIFIER
TSVPNEFG DC    AL1(0)             FIELD GROUP INDEX
TSVPNERS DC    AL2(0)             REPEATABLE SEQUENCE INDEX
TSVPNEFN DC    CL8' '             FIELD NAME
TSVPNEDA DC    AL2(0)             DATA AREA INDEX
TSVPNEOM DC    CL1' '             OPTION FIELD MODIFIER
*                                 BLANK = AS IS
*                                  D    = DATA AREA
*                                  O    = OPTION
*                                  A    = AFTER
*                                  B    = BEFORE
TSVPMODS DS    0CL5               MODIFIER GROUP
TSVPMONI DC    CL1' '             NI MODIFIER
*                                 BLANK = AS IS
*                                  N    = NEXT NI
*                                  F    = FIRST NI
*                                  L    = LAST NI
*                                  P    = PRECEDING NI
TSVPMOFG DC    CL1' '             FIELD GROUP MODIFIER
*                                 BLANK = AS IS
TSVPMORS DC    CL1' '             REPEATABLE SEQUENCE MODIF.
*                                 BLANK = AS IS
*                                  N    = NEXT RS
*                                  F    = FIRST RS
*                                  L    = LAST RS
*                                  X    = RS EXTENSION SUPPLIED
TSVPMOFN DC    CL1' '             FIELD MODIFIER
*                                 BLANK = AS IS
*                                  N    = NEXT FN
*                                  F    = FIRST FN
*                                  L    = LAST FN
*                                  E    = FIRST FN EQUAL
```

```
*                                        A    = NEXT AFTER FN
*                                        V    = VERY FIRST FN
*                                        S    = NEXT FD WITHIN RS
TSVPMODA DC    CL1' '                   DA MODIFIER
*                                       BLANK = AS IS
*                                        N    = NEXT DA
*                                        F    = FIRST DA
*                                        L    = LAST DA
TSVPNIRZ DC    AL1(0)                   0=NI AS IS / 1=NI MODIFIED T 0
TSVPFCMO DC    CL4' '                   FUNCTION MODIFIER
*                                       *** READ ****
*                                       BLANK = NO EDIT, NO CHECK
*                                        EDIT = EDIT, NO CHECK
*                                        CHK  = NO EDIT, BUT CHECK
*                                        EDCH = EDIT AND CHECK
*                                        OPTL = READ OPT. LIST
*                                        INFO = READ TOF INFORMATION
*                                       *** WRITE ****
*                                        DEED = DEEDIT, NO CHECK
*                                        CHK  = NO EDIT, BUT CHECK
*                                        DECH = DEEDIT AND CHECK
*                                        IGN  = IGNORE DATA FOR WRITE
*                                        SVAL = Set TOF increase values
*                                       *** DEL  ****
*                                        DLNI = DELETE NI
*                                        DLRS = DELETE RS OCC.
*                                        DLFN = DELETE FIELD
*                                        DLDA = DELETE DA OR OF
*                                        DLAF = DELETE ALL FIELDS
*                                        DLAX = DELETE ALL NI ¬= 0
*                                        DLAD = DELETE ALL DA OF FN
*                                        DLGR = DELETE DA>DA OF FLDREF
*                                       *** CHECK ****
*                                        DATA = CHECK DATA AREA
TSVPNIEX DC    AL1(0)                   NI OF EXIT FIELD
TSVPMFSR DC    AL2(0)                   MFS REASON CODE OR CHECK INFO
TSVPADDR DC    AL4(0)                   POINTER TO TOF
TSVPWORK DC    AL4(0)                   PTR TO WORK. BUFFER FOR TOF
TSVPENVR DC    AL4(0)                   ENVIRONM. FOR CHECK./SEP.
TSVPBUFF DC    AL4(0)                   POINTER TO INTERFACE BUFFER
TSVPDSLC DC    AL4(0)                   POINTER TO DSLCOM
TSVPDATL DC    AL2(0)                   LENGTH OF DA IN TSVPBUFF
TSVPFS   DC    XL1'00'                  STATUS OF A FIELD
TSVPFSMF EQU   X'80'                    MFS CHECKING ERROR OCCURRED
TSVPFSEM EQU   X'40'                    DATA AREA EMPTY
TSVPFSRX EQU   X'20'                    TOO MANY OCCURRENCES
TSVPFSDX EQU   X'10'                    TOO MANY COMPONENTS
TSVPFSLS EQU   X'08'                    FIELD LENGTH TOO SMALL
TSVPFSLG EQU   X'04'                    FIELD LENGTH TOO GREAT
TSVPFSLF EQU   X'02'                    FIELD LENGTH NOT FIXED
TSVPFSMN EQU   X'01'                    FIELD MANDATORY
         DC    XL1'00'                  RESERVED
TSVPRSXA DC    AL4(0)                   POINTER TO RS EXTENSION
```

If you request services for a field in a nested repeatable sequence, you must supply the repeatable sequence occurrence parameters using an RS Extension Parameter List:

```
TSVPRSXT DS    0F                       RS EXTENSION / BUFFER FORMAT
         DC    AL2(TSVPRSXL,0,0,0)
TSVPRSXK DC    AL2(0)                   NUMBER OF FOLLOWING ENTRIES
*        RS EXTENSION LIST ENTRY (6 BYTE/ENTRY)
TSVPRSXC DC    AL2(0)                   CURRENT RS INDEX ON LEVEL
TSVPRSXN DC    AL2(0)                   NEXT RS INDEX ON LEVEL
TSVPRSXM DC    CL1' '                   MODIFIER FOR RS INDEX ON LEVEL
         DC    CL1' '                   RESERVED
         ORG   TSVPRSXC
```

```
TSVPRSXX DC    &RSEXTC.XL6'00'         DECLARE EXTENSION AREA
*                                      DEFAULT IS RSEXT=3
TSVPRSXL EQU   *-TSVPRSXT              LENGTH OF RS EXTENSION AREA
```

The address of the RS Extension Parameter List is supplied in TSVPRSXA, and TSVPMORS must be set to 'X' to indicate that an RS Extension Parameter List is supplied.

Setup for DSLTOFSV

You communicate with DSLTOFSV via the parameter list (TSVPARMS) and the repeatable sequence extension parameter list (RSEXT) if a field in a nested repeatable sequence is to be accessed. TSVPBUFF is a TOF input/output buffer referenced by TSVPARMS.

To make use of the dynamic space allocation of the TOF you must either define the increase value of the TOF in the DSLPARM, or set the increase value using the following TOF request:

```
TYPE=WRITE,FMODIF=SVAL,FDNAM='*DSLTOF$'.
```

If the dynamic TOF is enabled, a TOF full condition results in the allocation of internally requested storage. From now on the TOF is split into an index part (the TOF space supplied by the caller) and a data part (internally requested). These two areas are linked by a pointer in the TOF header of the caller's supplied TOF.

Notes:

1. The TOF supervisor only requests more storage if the reorganization did not return enough free space to fulfill the request.
2. When you use the dynamic TOF, the following rules apply:
   - Do not relocate the TOF buffer in storage.
   - Issue a TOF free request before releasing the storage of the TOF.

The dynamic TOF is fully transparent if you use the call interface.

If you require TOF services as an MFS exit program (user exit), you must supply OPT=EXTTS with the DSLMFS MF=START call when you start your MFS exit (see "Coding MFS Exit Programs" on page 75). Then MFS provides your program with a prefilled TOF parameter list. The generated parameter list starts with the label 'MFSTSVL' and uses the prefix 'TS$'. For example, a DSLTSV TYPE=READ macro can look like the following example:

```
DSLTSV TYPE=READ,BUFFER=INFBUF,FDNAM='DSLMSG',                *
       NESTID=0,FDGPIND=1,RSINDEX=1,DAINDEX=1,                *
       PREFIX=TS$,MF=(E,MFSTSVL)
```

If you use DSLTOFSV directly, you must supply the TOF parameter list and temporary storage used by DSLTOFSV modules.

To generate TSVPARMS, either supply directly a DSECT according to the DSLTOFSV calling parameter list and enter the addresses for the buffers required by DSLTOFSV, or use the following macro:

```
[label]  DSLTSV MF=L[,DSECT=YES|NO,PREFIX=...]
```

You must also provide the temporary storage of DSLTOFSV (TS). The temporary storage TS is used as working buffer by DSLTOFSV. The length of this buffer should be 2KB. Either create a buffer using general services DSLSRVP and supply

the address with the WORK parameter in a DSLTSV macro, or in the parameter TSVPWORK of TSVPARMS, or use the following macro:

```
[label]  DSLTSV MF=TS[,PREFIX=...]
```

This call creates a buffer addressable with the name TSVTSBEG for "begin of temporary storage."

Note: If the parameter PREFIX is omitted in a DSLTSV macro then the default prefix 'TSV' is taken. Throughout this description the default prefix is assumed. If you supply a different prefix, you must replace 'TSV' in all parameters with your prefix.

Information Returned

You get return information from DSLTOFSV in register 15, and in the TSVPARMS fields TSVPRC, TSVPRSC, TSVPCURR, TSVPNIEX, TSVPMFSR, TSVPDATL, and TSVPFS.

Register 15 and TSVPRC contain the return code; TSVPRSC contains the reason code. For a description of the return and reason codes, refer to *MERVA for ESA Messages and Codes*.

The return code can be 0, 4, or 8.

- Return code 0 means that your request has been completed successfully.
- For return code 4, the reason code in TSVPRSC gives additional information, TSVPRSC=TOFRFDNF after a 'READ' request, which means that the field to be read was not found in the TOF.

  The following general reason codes referring to errors in the TOF parameter list can be returned:

  ```
  TOFRRTYP = incorrect contents of modifiers (TSVPMODS)
  TOFRFTYP = incorrect function type (TSVPFTYP)
  TOFRFMOD = incorrect function modifier (TSVPFCMO)
  TOFRINIT = TOF not correctly initialized
  TOFRNOBU = no valid buffer for data transfer (TSVPBUFF)
  TOFREXBU = no valid RS Extension parameter list supplied (TSVPRSXA)
  TOFRXBUF = RS Extension Parameter List too small
  TOFRFNRV = field name supplied is reserved (TSVPNEFN).
  ```

  The following field names are reserved:

  – TOFDUMMY
  – DSLRSBEG
  – DSLRSEND
  – *DSLTOF$.

  The following general reason codes indicate errors depending on the definition of modifiers supplied in TSVPMODS:

  ```
  TOFRFDNF = field required for change not found
  TOFRNEST = nesting identifier not available
  TOFROCCU = occurrence not found
  TOFRRSEQ = field is not in repeatable sequence (RS-modifiers)
  TOFRDAIN = data area index required for change not found
  TOFRCMOD = 'illogical' combination in list of modifiers
  TOFRNIFG = nesting identifier changed by modification.
  ```

  If a routine was invoked by DSLTOFSV via MFS and the routine was not processed successfully, the following reason codes can be returned:

```
TOFRCHEC = checking routine failed
TOFREDIT = editing routine failed
TOFRDFLT = default setting routine failed
TOFREXPA = expansion routine failed.
```

- Return code 8 shows a severe error. The following general reason codes can be returned:

```
TOFRSMAL = TOF buffer supplied is smaller than minimum size
TOFRDAMD = TOF supplied is damaged
TOFRWSMA = TOF working buffer supplied too small
TOFRWBMI = TOF working buffer supplied is smaller than
           minimum size.
```

TSVPCURR contains the field reference of the field accessed if the DSLTOFSV request could be executed successfully, otherwise the field reference of the last successful call is retained.

TSVPNIEX is filled when an exit field to a nesting identifier is accessed or read with the value of the nesting identifier this exit field points to, otherwise it is set to 0.

TSVPMFSR is filled with the MFS reason code returned if MFS internally called by DSLTOFSV fails, for example, when DSLTOFSV called a checking routine via MFS required for checking of data and MFS returned a checking error. Then, DSLTOFSV returns a DSLTOFSV reason code and supplies the MFS reason code with TSVPMFSR in addition.

TSVPDATL is set by DSLTOFSV when data is transferred from or to the TOF by request types 'READ', 'WRITE', or 'ADDDA' and contains the net length of the data in the buffer referenced by TSVPBUFF.

TSVPFS contains the status of a field. The following status bits are set according to status conditions recognized while executing checking or read requests:

- TSVPFSMF is set when a checking routine called via MFS returned a nonzero return code.
- TSVPFSEM is set when a data area to be read is empty.
- TSVPFSRX is set when DSLTOFSV basic checking found too many occurrences of a field.
- TSVPFSDX is set when DSLTOFSV basic checking found too many data areas of a field.
- TSVPFSLS is set when DSLTOFSV basic checking found that the length of data of a field was smaller than specified in the FDT or MCB.
- TSVPFSLG is set when DSLTOFSV basic checking found that the length of data of a field was greater than specified in the FDT or MCB.
- TSVPFSLF is set when DSLTOFSV basic checking found that the length of data of a field does not match the fixed length specification in the FDT or MCB.
- TSVPFSMN is set when the data to be read by a 'READ' request is mandatory.

TSVPBUFF (shown below) is the info buffer for a READ (info) for WRITE (sval) request for the reserved field *DSLTOF$. The data is prefixed with the standard MERVA ESA buffer.

```
TOFINFO  DS    0F              TOF info structure for *DSLTOF$
TINFXTEN DS    F               TOF Extension value
TINFMSZ  DS    F               TOF Maximum Size
```

```
TINFJBSZ DS    F               TOF Join Buff Size
TINFFMT  DS    F               TOF Format 1 (single) or 2 (split)
TINFDAA  DS    F               TOF Data Address
```

Creating a New TOF

Before you can start to use TOF services successfully you must supply a valid TOF buffer. If MFS does not supply a valid TOF, you must provide the address of a buffer with standard MERVA ESA layout and code the DSLTSV TYPE=TOFNEW macro (TSVPFTYP='TNEW' for calling interface).

This instruction causes DSLTOFSV to initialize the buffer according to the TOF requirements and provides the nesting identifier with NESTID=0 ready for use. The new TOF data area is initially cleared to X'58'.

Note: The dynamic TOF definitions (TOFSIZE,MAXBUF) are read from the DSLPARM and saved as the dynamic TOF settings. You can change these settings by a WRITE dynamic TOF settings request.

If the return code from TOF initializing was zero, you can start to use the TOF. Below is an example of initializing the TOF:

```
         DSLTSV TYPE=TOFNEW,         REQUEST TYPE = TOFNEW           *
               TOF=TOFBUF,         TOFBUF CONTAINS BUFFER ADDRESS    *
               MF=(E,TSVPARMS)
         LTR   R15,R15             OK?
         BNZ   ERROR
         ...                       PERFORM DSLTOFSV REQUESTS AS
         ...                       REQUIRED BY YOUR PROGRAM
         ...
ERROR    ...
```

Note: The length of the buffer supplied must be at least 200 bytes and cannot exceed 2097144 bytes. If a buffer is supplied containing a TOF, the TOF is cleared.

Writing Dynamic TOF Settings

The dynamic TOF settings can be changed at any time. The DSLPARM settings are system-wide settings. An application might want to override them, for example, the application in the installation that works with large messages. You need to change the settings if you do not want the defaults taken from the DSLPARM at the initialization of the TOF (following the TOFNEW request).

To change the settings of a dynamic TOF, send the TOFINFO data in a buffer referenced by TSVPBUFF and communicated to DSLTOFSV by the parameter BUFFER either in this or in a previous DSLTSV macro.

The following WRITE SVAL request disables the dynamic TOF functions. This means that you get a ″TOF full″ return code if all the space you have initialized with the TOFNEW request is filled.

Below is an example of writing the dynamic TOF settings:

```
         ...                       PROVIDE THE DATA TO BE
         ...                       WRITTEN IN THE BUFFER 'SVALBUF'
         DSLTSV TYPE=WRITE,        REQUEST TYPE = WRITE          *
               BUFFER=SVALBUF,     SVALBUF REFERS TO THE SETTINGS *
               ,                   CONTAINING THE DATA           *
               NESTID=0,           NESTING IDENTIFIER = 0        *
```

```
                FDGPIND=0,            FIELD GROUP = 0               *
                FDNAM='*DSLTOF$',     FIELD NAME = '*DSLTOF$'       *
                DAINDEX=0,            DATA AREA INDEX = 0           *
                OPTION=NO,            DATA AREA TO BE WRITTEN       *
                FMODIF=SVAL,          OVERWRITE SETTING VALUES      *
                MF=(E,TSVPARMS)
          LTR   R15,R15               OK?
          BNZ   ERROR
          ...                         CONTINUE NORMAL
          ...                         PROGRAM PROCESSING
          ...
ERROR     ...
          ...
          DS    0F
SVALBUF   DC    H'28',H'0'    Buffer Length
          DC    H'24',H'0'    Data Area Length
TOFINFO   DS    0F            TOF info structure for DSLTOF$        *
TINFXTEN  DC    F'0'          TOF Extension value                   *
TINFMSZ   DC    F'0'          TOF Maximum Size                      *
TINFJBSZ  DC    F'0'          TOF Join Buff Size                    *
TINFFMT   DC    F'0'          TOF Format 1 or 2                     *
TINFDAA   DC    F'0'          TOF Data Address                      *
```

The key parameters are:

TOFINFO The following fullwords describe the structure used for the dynamic TOF setting.

TINFXTEN This defines the increase value if the TOF must be extended. A zero value indicates NO dynamic TOF space allocation.

TINFMSZ The dynamic allocation stops if this value is reached as a sum of the Index plus Data Part space.

TINFJBSZ Not used on SVAL request. On READ INFO it returns the smallest buffer size needed to JOIN the Index and Data TOF part.

TINFFMT Not used on SVAL request. On READ INFO it indicates the TOF format:

- **1** The TOF uses only the original user-supplied area (single TOF)
- **2** An additional area has been allocated by the TOF Supervisor to hold the Data part of the TOF (split TOF).

TINFDAA Not used on SVAL request. On READ INFO it passes the address of internally requested space for the data part of the TOF.

Freeing the Space Allocated by the TOF Supervisor

If you used the dynamic TOF service the TOF supervisor might have allocated additional space for the data part of the TOF. You must issue a FREE request before you free or reuse the space you have initialized by the TOFNEW request.

The TOF supervisor checks if additional space was allocated and returns the area to the operating system.

Below is an example of freeing the dynamic TOF space:

```
          ...
          ...
          DSLTSV TYPE=FREE,           REQUEST TYPE = FREE           *
                MF=(E,TSVPARMS)
          LTR   R15,R15               OK?
```

```
         BNZ   ERROR
         ...                          CONTINUE NORMAL
         ...                          PROGRAM PROCESSING
         ...
ERROR    ...
         ...
```

Writing Data to the TOF

To write data to a TOF, send it in the buffer referenced by TSVPBUFF and communicated to DSLTOFSV by the parameter BUFFER either in this or a previous DSLTSV macro.

If the field reference of the data to be written is not in the “current” parameters of TSVPARMS (retained from a previous successful TOF service) then you must provide the field reference with the “next” or “modifier” parameters or both. The field reference is evaluated starting from the “current” position (TSVPCURR). This is overwritten by the “next” position parameters (TSVPNEXT) specified. This “actual” position is further changed by the modifiers specified (TSVPMODS) yielding the final field reference.

Note: If any of the parameters: field group (FG), repeatable sequence occurrence (RS), or data area index (DA) of the field reference contains a value of 0, this is replaced by a value of 1.

If data is written to a field that is not in the TOF, the field is implicitly initialized. For example, a field descriptor is created in the TOF using only the options specified in the FDT for this field. Options specified in an MCB are not included.

If the field that is implicitly initialized has a default setting routine assigned in the FDT, this routine is called by DSLTOFSV. This routine can imply additional DSLTOFSV requests. Data returned by this routine is ignored in the actual “WRITE” request, and the original data is written.

If the field name FN supplied refers to a subfield, the corresponding main field is initialized, but no default setting routine is called.

Note: A field can be written only on a nesting identifier introduced in the TOF. If the nesting identifier specified in the field reference is missing, it must be introduced first using DSLTSV TYPE=ADDNI instructions to add the nesting identifiers required. If the data area index or option to be written is found in the TOF, the data found is replaced, otherwise the data is added to the field.

If the repeatable sequence was initialized using a repeatable sequence extension parameter list (RSEXT) and the actual “WRITE” request supplies an RSEXT, a field that is part of a repeatable sequence cannot be implicitly initialized.

If the return code from the 'WRITE' request is not 0, checking of the reason code can be necessary. The following more specific reason codes are to be expected:

```
TOFRFULL = TOF is full, not enough space available
TOFRNEST = nesting identifier not available.
```

Below is an example of writing data to the TOF:

```
         ...                          PROVIDE THE DATA TO BE
         ...                          WRITTEN IN THE BUFFER 'MYBUF'
         DSLTSV TYPE=WRITE,           REQUEST TYPE = WRITE          *
               BUFFER=MYBUF,          MYBUF REFERS TO THE BUFFER    *
```

```
                ,                       CONTAINING THE DATA           *
                NESTID=0,               NESTING IDENTIFIER = 0        * [1]
                FDGPIND=1,              FIELD GROUP = 1               *
                FDNAM='MYNAME',         FIELD NAME = 'MYNAME'         *
                DAINDEX=3,              DATA AREA INDEX = 3           * [2]
                OPTION=NO,              DATA AREA TO BE WRITTEN       *
                FMODIF=DECHECK,         DATA ARE TO BE CHECKED AND    * [3]
                ,                       DE-EDITED                     *
                MF=(E,TSVPARMS)
          LTR   R15,R15                 OK?
          BNZ   ERROR
          ...                           CONTINUE NORMAL
          ...                           PROGRAM PROCESSING
          ...
ERROR     ...
```

Notes:

1. NESTID=0

With this parameter the DSLTSV parameter list field TSVPNENI is set to 0.

2. DAINDEX=3

With this parameter the DSLTSV parameter list field TSVPNEDA is set to 3. If there is no data area in the TOF for the field referenced, then the data is written as data area index DA=1.

3. FMODIF=DECHECK

With this parameter the DSLTSV parameter list field TSVPFCMO is set to 'DECH'. TOF Supervisor tries first to de-edit and then to check the data to be written to the TOF. Editing and checking are independent functions. If de-editing was successful then the changed data is taken, otherwise the original data is processed further. If the field referenced has no editing routine assigned, the de-edit request is ignored. If the field has no checking routine assigned, the DSLTOFSV basic checking is carried out. The following checks are performed against the definition in the field descriptor:

- Number of data areas
- Repeatable sequence occurrence number
- Minimum and maximum field length.

If de-editing or checking fails, return and reason codes are supplied, but the data is written.

Note: In this example the parameter RSINDEX is not explicitly set. If it was set in the same TOF parameter list with a DSLTSV macro before, that value is still in effect. If the values of both TSVPCURS and TSVPNERS are 0, then RS=1 is used as default by DSLTOFSV.

Below is an example of writing data to the TOF using a nested RS Extension Parameter List:

```
          ...                           PROVIDE THE DATA TO BE
          ...                           WRITTEN IN THE BUFFER 'MYBUF'
          DSLTSV TYPE=WRITE,            REQUEST TYPE = WRITE          *
                BUFFER=MYBUF,           MYBUF REFERS TO THE BUFFER    *
                ,                       CONTAINING THE DATA           *
                NESTID=1,               NESTING IDENTIFIER = 1        * [1]
                FDGPIND=1,              FIELD GROUP = 1               *
                FDNAM='MYNAME',         FIELD NAME = 'MYNAME'         *
                DAINDEX=3,              DATA AREA INDEX = 3           *
                OPTION=NO,              DATA AREA TO BE WRITTEN       *
```

```
                MODIF=RSEXT,          RS EXTENSION TO BE USED       * [2]
                RSINDEX=(3,LASTRS,2),  SUPPLY OCC PARAMETERS        * [3]
                MF=(E,TSVPARMS)
          LTR   R15,R15               OK?
          BNZ   ERROR
          ...                         CONTINUE NORMAL
          ...                         PROGRAM PROCESSING
          ...
 ERROR    ...
```

Notes:

1. NESTID=1

With this parameter the DSLTSV parameter list field TSVPNENI is set to 1.

2. MODIF=RSEXT

With this parameter the DSLTSV parameter list field TSVPMORS is set to 'X'. This modifier indicates to the TOF Supervisor that the parameters, specifying the repeatable sequence occurrences (occurrence number or modifier) are supplied with an RS Extension Parameter List.

3. RSINDEX=(3,LASTRS,2)

With this parameter a list of parameters is provided, which sets the occurrence number or occurrence modifier of each repeatable sequence level in the RS Extension Parameter List.

In this example the field is part of a repeatable sequence that is nested in two other repeatable sequences. The occurrence number of the first, outer repeatable sequence is set to 3, the modifier of the next nested repeatable sequence is set to 'L', and the occurrence number of the innermost repeatable sequence is set to 2. The second occurrence of the innermost repeatable sequence is addressed. This is nested in the last occurrence of the middle repeatable sequence, which is nested in the third occurrence of the outermost repeatable sequence.

Adding Data Areas to the TOF

To add a data area, you can use the following two instructions: DSLTSV TYPE=WRITE or DSLTSV TYPE=ADDDA. If you want to append the data area to a sequence of data areas, then you use the 'WRITE' request and specify DAINDEX=32767 to make sure that no data area is overwritten in the TOF.

If you want to insert a data area before or after an existing data area, you must use the 'ADDDA' request.

Note: The 'ADDDA' request can be used only for adding data to a field already initialized in the TOF.

If the return code from the 'ADDDA' request has not been zero, checking of the reason code can be required. The following more specific reason codes are to be expected:

```
TOFRFULL = TOF is full, not enough space available
TOFRNEST = Nesting identifier not available
TOFROCCU = Occurrence not found
TOFRFDNF = Field not found.
```

Below is an example of adding a data area to the TOF:

```
          ...                       PROVIDE THE DATA TO BE
          ...                       ADDED IN THE BUFFER 'MYBUF'
          DSLTSV TYPE=ADDDA,        REQUEST TYPE = ADDDA          *
                BUFFER=MYBUF,       MYBUF REFERS TO THE BUFFER    *
                ,                   CONTAINING THE DATA           *
                FDGPIND=1,          FIELD GROUP = 1               *
                FDNAM='MYNAME',     FIELD NAME = 'MYNAME'         *
                DAINDEX=1,          DATA AREA INDEX = 1           * [1]
                MODIF=(NEXTNI,NEXTDA), POSITION MODIFIED          * [2]
                OPTION=BEFORE,      DA INSERTED BEFORE            * [3]
                FMODIF=CHECK,       DATA ARE TO BE CHECKED        * [4]
                MF=(E,TSVPARMS)
          LTR   R15,R15             OK?
          BNZ   ERROR
          ...                       CONTINUE NORMAL
          ...                       PROGRAM PROCESSING
          ...
ERROR     ...
```

Notes:

1. DAINDEX=1

With this parameter the DSLTSV parameter list field TSVPNEDA is set to 1. The data area index in the “actual” position is set to DA=1.

2. MODIF=(NEXTNI,NEXTDA)

With this parameter the “actual” position evaluated from the current (TSVPCURR) and next (TSVPNEXT) parameters is verified in the TOF. If the field referenced by the “actual” position is not found in the TOF, the change cannot be carried out according to the specified MODIF parameter, and a return and reason code is supplied. Otherwise the “actual” position is changed first, with the field name of the first field on the next logical nesting identifier (NEXTNI). FG, RS, and DA are set to 1, irrespective of the values set in the 'next' parameter.

If either the next nesting identifier is not yet introduced to the TOF or no field has been initialized on this nesting identifier, DSLTOFSV stops processing and returns the reason code TOFRNEST (nesting identifier not available). The second modifier (NEXTDA) changes DA to DA+1, so that the data area index in the final field reference is DA=2.

If further processing does not detect an error, the reason code TOFRNIFG is returned, indicating that a modifier changed the “actual” nesting identifier. If you expect this to happen, ignore this reason code in the processing of your program.

Note: In this example the macro parameter DAINDEX=1 does not affect positioning because the modifier NEXTNI sets the data area index.

3. OPTION=BEFORE

If the reference of the data area evaluated after changing the “actual” position is in the TOF, the data supplied is inserted as data area DA=2 before the previous DA=2 for the field referenced.

4. FMODIF=CHECK

With this parameter, the DSLTSV parameter list field TSVPFCMO is set to 'CHEK'. The TOF Supervisor tries to check the data before writing it to the TOF. If the field has no checking routine assigned, the DSLTOFSV basic checking is performed. If checking fails, return and reason codes are supplied, but the data is inserted.

Note: In this example the parameter NESTID is not explicitly set. TSVPNIRZ is defaulted to 0. If TSVPNENI is still 0, then the nesting identifier NI specified in TSVPCUNI is not changed.

Reading Data from the TOF

To read data from the TOF, use the DSLTSV TYPE=READ macro. You can read one of the following, depending on the option modifier OPTION and the function modifier FMODIF:

- A data area
- An option
- The option list
- The field descriptor
- The maximum nesting level of repeatable sequences initialized in the TOF.

If you read a field descriptor (DSLTSV TYPE=READ,FMODIF=FDSCRPT) the result obtained in the buffer referenced by TSVPBUFF depends on whether a main field or a subfield is referenced:

- If a main field, the descriptor, as stored in the TOF, is received according to the TOFFDE DSECT, described in "Initializing Fields in the TOF" on page 37.
- If a subfield, a field descriptor of a subfield is received.

This is indicated by the reason code TOFRSUBF in the field TSVPMFSR of the TOF parameter list (TOF return and reason code = 0). The DSECT of the field descriptor of a subfield is obtained by the DSLDSFDT macro (DSECT SUBFDS). For more information refer to the *MERVA for ESA Macro Reference*.

If you read the maximum nesting level of repeatable sequences (MAXLEV) (DSLTSV TYPE=READ,FMODIF=INFO,FDNAM=DSLRSLEV), the result is returned in the first fullword of the buffer referenced by TSVPBUFF. To scan through the TOF, for example by using the modifier NEXTFD, you must provide a repeatable sequence extension parameter list large enough to save all parameters necessary to access a field of the innermost nested repeatable sequence. The buffer size of the *rs* extension buffer (BL) to be allocated can be calculated as follows: BL = MAXLEV * 6 + 2 + 8 (Buffer Header).

If the field referenced was initialized in the TOF, but no data area is available for this field, then a default setting routine is called by DSLTOFSV (if assigned). The data returned by the default setting routine is supplied in the buffer referenced by TSVPBUFF and written to the TOF as first data area of the field referenced. If the field name supplied refers to a subfield, the default setting routine assigned to the subfield is called. The data returned is written as first data area to the main field; an assigned separation routine is not called.

Note: A default setting routine is called only once for subfields of the same main field. The default setting routine is called only when the first subfield is read. Additional default setting routines specified for other subfields of the same main field are not called.

If the return code from the 'READ' request is not zero, check the reason code. The following more specific reason codes can occur:

```
TOFRFULL = TOF is full, not enough space available, if a default
           setting routine was invoked and supplied data to be
           written to the TOF
TOFRFDNF = Field not found
```

```
TOFROCCU = Occurrence not found
TOFRDAIN = Data area index too high, not in TOF
TOFRNOPT = Field has no option defined
TOFROPTN = Option not found
TOFRBUFU = Buffer too small to return all data.
```

Below is an example of reading the option list of a field:

```
         DSLTSV TYPE=READ,          REQUEST TYPE = READ           *
                BUFFER=MYBUF,       MYBUF REFERS TO THE BUFFER    *
                ,                   AIMED TO RECEIVE THE DATA     *
                NESTID=1,           NESTING IDENTIFIER = 1        * [1]
                FDGPIND=5,          FIELD GROUP = 5               * [2]
                RSINDEX=1,
                FDNAM='MYNAME',     FIELD NAME = 'MYNAME'         *
                FMODIF=OPTLIST,     OPTION LIST TO BE READ        * [3]
                MF=(E,TSVPARMS)
         LTR    R15,R15             OK?
         BNZ    ERROR
         ...                        CONTINUE NORMAL
         ...                        PROGRAM PROCESSING
         ...
ERROR    ...
```

Notes:

1. NESTID=1

This parameter sets the DSLTSV parameter list field TSVPNENI to 1. The nesting identifier in the “actual” position is set to NI=1.

2. FDGPIND=5

This parameter sets the DSLTSV parameter list field TSVPNEFG to 5. The field group in the ″actual″ position is set to FG=5.

3. FMODIF=OPTLIST

This parameter sets the DSLTSV parameter list field TSVPFCMO to 'OPTL'. Assuming the field referenced is in the TOF, DSLTOFSV returns the reason code TOFRNOPT if the field has no option defined and TOFRMISS when the option list is missing. Otherwise the option list is returned in the buffer referenced by 'MYBUF'. The first byte of the option list contains the length of the option literals followed by the list of option literals.

Note: If the option modifier OPTION was not specified explicitly, any specified value is ignored when reading the option list.

Reading Dynamic TOF Settings

The dynamic TOF settings can be read at any time. The TOFINFO data structure is returned in a buffer referenced by TSVPBUFF.

The following READ INFO request returns the dynamic TOF information.

Below is an example of reading the dynamic TOF settings:

```
         ...
         ...
         DSLTSV TYPE=READ,          REQUEST TYPE = WRITE          *
                BUFFER=INFOBUF,     INFOBUF REFERS TO THE SETTINGS *
                ,                   CONTAINING THE DATA           *
                NESTID=0,           NESTING IDENTIFIER = 0        *
                FDGPIND=0,          FIELD GROUP = 0               *
                FDNAM='*DSLTOF$',   FIELD NAME = '*DSLTOF$'       *
```

```
                DAINDEX=0,            DATA AREA INDEX = 0           *
                OPTION=NO,            DATA AREA TO BE WRITTEN       *
                FMODIF=INFO,          OVERWRITE SETTING VALUES      *
                MF=(E,TSVPARMS)
           LTR  R15,R15               OK?
           BNZ  ERROR
           ...                        CONTINUE NORMAL
           ...                        PROGRAM PROCESSING
           ...
ERROR      ...
           ...
           DS   0F
INFOBUF    DC   H'28',H'0'    Buffer Length
           DC   H'24',H'0'    Data Area Length
TOFINFO    DS   0F            TOF info structure for DSLTOF$          [1]
TINFXTEN   DC   F'0'          TOF Extension value                     [2]
TINFMSZ    DC   F'0'          TOF Maximum Size                        [3]
TINFJBSZ   DC   F'0'          TOF Join Buff Size                      [4]
TINFFMT    DC   F'0'          TOF Format 1 or 2                       [5]
TINFDAA    DC   F'0'          TOF Data Address                        [6]
```

The key parameters are:

TOFINFO The following fullwords describe the structure used for the dynamic TOF settings.

TINFXTEN This defines the increase value if the TOF must be extended. A zero value indicates NO dynamic TOF space allocation.

TINFMSZ The dynamic allocation stops if this value is reached as a sum of the Index plus Data Part space.

TINFJBSZ This value is the smallest buffer size needed to JOIN the Index and Data TOF part.

TINFFMT Indicates the TOF format:

1 The TOF uses only the original user-supplied area (single TOF)

2 An additional area has been allocated by the TOF Supervisor to hold the Data part of the TOF (split TOF).

TINFDAA Passes the address of internally requested space for the TOF's data part.

Deleting Data from the TOF

To remove data from the TOF, use the DSLTSV TYPE=DELETE macro. You can delete all fields in the TOF, a nested message, an occurrence, a field, data areas or the option of a field depending on the option modifier OPTION and function modifier FMODIF.

Note: You must specify a function modifier; no default is assumed for the part of the TOF to be deleted.

If the return code from the 'DELETE' request is not zero, check the reason code. The following more specific reason codes can occur:

```
TOFRFDNF = Field not found
TOFROCCU = Occurrence not found
TOFRDAIN = Data area index too high, not in TOF
TOFRRSEQ = Field is not in repeatable sequence (FMODIF=DELRS).
```

Below is an example of deleting a nesting identifier:

```
          DSLTSV TYPE=DELETE,         REQUEST TYPE = DELETE    *
                NESTID=1,             NESTING IDENTIFIER = 1   *  [1]
                FDGPIND=0,            FIELD GROUP = 0          *  [2]
                FDNAM='MYNAME',       FIELD NAME = 'MYNAME'    *
                FMODIF=DELNI,         DELETE NI                *  [3]
                MF=(E,TSVPARMS)
          LTR   R15,R15               OK?
          BNZ   ERROR
          ...                         CONTINUE NORMAL
          ...                         PROGRAM PROCESSING
          ...
ERROR     ...
```

Notes:

1. NESTID=1

This parameter sets the DSLTSV parameter list field TSVPNENI to 1. The nesting identifier in the “actual” position is set to NI=1.

2. FDGPIND=0

This parameter sets the DSLTSV parameter list field TSVPNEFG to 0. The field group as specified in TSVPCUFG is kept in the “actual” position.

3. FMODIF=DELNI

This parameter sets the DSLTSV parameter list field TSVPFCMO to 'DLNI'. When executing the request, the field reference is evaluated as usual, but only the nesting identifier is used. If the nesting identifier required (NI=1 in this example) is not in the TOF, the reason code TOFRNEST is returned. Otherwise, the exit to this nesting identifier is removed in the exit field on NI=0, and all fields on the nesting identifier NI=1 and on nesting identifiers logically added to this nesting identifier are deleted.

Note: If this request executes successfully, the complete field reference is saved in TSVPCURR, although only the nesting identifier is checked. This position is not valid because the nesting identifier specified is no longer available.

Accessing Fields in the TOF

To access a field in the TOF, use the DSLTSV TYPE=ACCESS macro.

This request type is useful when you want to make sure that a field you would like to read, change, or check is really in the TOF.

Note: With this request the presence of a field in the TOF verified using the parameters nesting identifier NI, field group FG, repeatable sequence occurrence RS, and field name FN. The data area index DA supplied need not be in the TOF.

If the return code from the 'ACCESS' request is not 0, check the reason code. The following more specific reason codes can occur:

```
TOFRFDNF = Field not found
TOFROCCU = Occurrence not found.
```

Below is an example of accessing fields in the TOF sequentially:

```
          DSLTSV TYPE=ACCESS,         REQUEST TYPE = ACCESS         *
                NESTID=0,             NESTING IDENTIFIER = 0        * [1]
                FDGPIND=0,            FIELD GROUP = 0               * [2]
```

```
               RSINDEX=0,            REP. SEQ. OCC. = 0            * [3]
               FDNAM='       ',      FIELD NAME = '      '         * [4]
               MODIF=FIRSTNI,        POSITION MODIFIED             * [5]
               MF=(E,TSVPARMS)
          LTR  R15,R15               OK?
          BNZ  ERROR
LOOP      DSLTSV TYPE=ACCESS,        REQUEST TYPE = ACCESS         *
               MODIF=NEXTFD,         POSITION MODIFIED             * [6]
               MF=(E,TSVPARMS)
          CLC  TSVPRSC,TOFRFDNF      FIELD NOT FOUND                 [7]
          BE   CONT
          ...                        PROCESS FIELD REFERENCE
          ...                        RETURNED IN TSVPCURR
          B    LOOP
CONT      ...                        CONTINUE NORMAL
          ...                        PROGRAM PROCESSING
          ...
ERROR     ...
```

'READ', 'WRITE' and 'CHECK' requests can be processed similarly.

Notes:

1. NESTID=0

This parameter sets the DSLTSV parameter list field TSVPNENI to 0 and TSVPNIRZ to 1. The nesting identifier in the “actual” position is set to 0.

2. FDGPIND=0

This parameter sets the DSLTSV parameter list field TSVPNEFG to 0. The field group as specified in TSVPCUFG is kept in the “actual” position.

3. RSINDEX=0

This parameter sets the DSLTSV parameter list field TSVPNERS to 0. The repeatable sequence occurrence as specified in TSVPCURS is kept in the “actual” position.

4. FDNAM=' '

This parameter sets the DSLTSV parameter list field TSVPNEFN to blanks. The field name as specified in TSVPCUFN is kept in the “actual” position.

5. MODIF=FIRSTNI

With this parameter the “actual” position evaluated from the current (TSVPCURR) and next (TSVPNEXT) parameters is changed with the first nesting identifier (NI=0), the first field group index and the name of the first field of this nesting identifier (NI=0). RS and DA are set to 1 in the field reference.

Note: With this modifier you access the first field in the TOF.

6. MODIF=NEXTFD

With this parameter the “actual” position evaluated from the current (TSVPCURR) and next (TSVPNEXT) parameters is changed with the field reference of the field following the field at the “actual” position.

Note: In the preceding DSLTSV macro the 'next' parameters (NI, FG, RS, FN) were set to values that do not overwrite the current position parameters returned. This is a prerequisite to use the NEXTFD parameter effectively in a loop.

In contrast to MODIF=NEXTFN, the fields in repeatable sequences are accessed sequentially for all occurrences of this repeatable sequence initialized in the TOF.

7. CLC TSVPRSC,TOFRFDNF

The reason code TOFRFDNF is expected when the last field in the TOF was read and the next field is not found. All other reason codes to be expected can either be ignored or referred to general errors that would have occurred with the first DSLTSV macro, and should be dealt with after this instruction.

Checking Fields in the TOF

To check a field in the TOF, use the DSLTSV TYPE=CHECK macro. You can check the complete field or a single data area when supplying the function modifier FMODIF=DATA.

With this request, the referenced field is accessed by DSLTOFSV. Assuming the field is found in the TOF, the checking routine supplied for this field in the FDT or MCB is called. If no checking routine was assigned to this field, the DSLTOFSV basic checking is performed.

The result of checking is returned in TOF return and reason code. If a checking routine called via MFS returns an error, the bit TSVPFSMF is set, and the reason code supplied in TSVPMFSR. DSLTOFSV basic checking sets the status bits in TSVPFS and supplies an error message in MFS permanent storage.

If the return code from the 'CHECK' request is not 0, check the reason code. The following more specific reason codes can occur:

```
TOFRFDNF = Field not found
TOFROCCU = Occurrence not found
TOFRCHEC = The checking routine called failed
TOFRLMIN = Field length error detected by DSLTOFSV basic checking
TOFRLMAX = Field length error detected by DSLTOFSV basic checking
TOFRLFIX = Field length error detected by DSLTOFSV basic checking
TOFRAREA = Too many data areas detected by DSLTOFSV basic checking
TOFROCCR = Too many occurrences detected by DSLTOFSV basic checking
TOFRCONT = Contents error communicated by the SWIFT Link checking routine.
```

Below is an example of checking a field in the TOF:

```
         DSLTSV TYPE=CHECK,          REQUEST TYPE = CHECK          *
               NESTID=1,             NESTING IDENTIFIER = 1        *
               FDGPIND=1,            FIELD GROUP = 1               *
               RSINDEX=1,            REP. SEQ. OCC. = 1            *
               FDNAM='MYNAME',       FIELD NAME = 'MYNAME'         *
               MODIF=NEXTFN,         POSITION MODIFIED             * [1]
               MF=(E,TSVPARMS)
         LTR   R15,R15               OK?
         BNZ   ERROR
         ...                         CONTINUE NORMAL
         ...                         PROGRAM PROCESSING
         ...
ERROR    ...
```

Notes:

1. MODIF=NEXTFN

With this parameter, the “actual” position evaluated from the current (TSVPCURR) and next (TSVPNEXT) parameters is changed with the next field name of the nesting identifier of the “actual” position (NI=1). RS and DA are set to 1 in the field reference.

Note: To carry out the change, the “actual” position achieved after evaluating the current and next parameters must be a valid position in the TOF.

Expanding Fields in the TOF

To expand a field in the TOF, use the DSLTSV TYPE=EXPAND macro.

With this request, first the field referenced is accessed by DSLTOFSV. Assuming the field is found in the TOF, then the expansion routine supplied for this field in the FDT or MCB is called. If no expansion routine was assigned to this field, then DSLTOFSV returns with return and reason code zero and saves the field reference in TSVPCURR.

If the return code from the 'EXPAND' request is not zero, check the reason code. The following more specific reason codes can occur:

```
TOFRFDNF = Field not found
TOFROCCU = Occurrence not found
TOFREXPA = The expansion routine called failed.
```

Below is an example of expanding a field in the TOF:

```
         DSLTSV TYPE=EXPAND,         REQUEST TYPE = EXPAND           *
               NESTID=1,             NESTING IDENTIFIER = 1          *
               FDGPIND=1,            FIELD GROUP = 1                 *
               RSINDEX=1,            REP. SEQ. OCC. = 1              *
               FDNAM='MYNAME',       FIELD NAME = 'MYNAME'           *
               MF=(E,TSVPARMS)                                       *
         LTR   R15,R15               OK?
         BNZ   ERROR
         ...                         CONTINUE NORMAL
         ...                         PROGRAM PROCESSING
         ...
ERROR    ...
```

Initializing Fields in the TOF

To initialize a field in the TOF, use the DSLTSV TYPE=INIT macro.

This request type is useful when you want to change the information from the field descriptor entry in the FDT with information supplied by an MCB entry. The field is initialized with a field descriptor containing the information from the FDT merged with the information from the MCB entry supplied. If the field has a default setting routine assigned, this default setting routine is called by DSLTOFSV.

If the return code from the 'INIT' request is not 0, check the reason code. The following more specific reason codes can occur:

```
TOFRFINI = Field already initialized
TOFRSUBF = Subfield cannot be initialized
TOFRXFNI = Field with rs extension does not fit to rs extension
           structure initialized in TOF
TOFRFULL = Not enough space in TOF
TOFRINNI = Nesting identifier not allowed for initialization
TOFRIMCB = MCB supplied is incorrect
TOFRDFLT = Default setting routine called failed.
```

To initialize a field, an MCB entry must be supplied in the buffer referenced by TSVPBUFF. You must supply the data required according to the DSECT shown below. You can map this structure in your program by using a DSLDSMCB macro and use the MFLDDS DSECT to supply the data.

```
MFLDDS   DSECT
MFDLNGTH DS    AL2                MFLD ENTRY LENGTH                    [ 1]
MFDIDENT DS    X                  IDENTIFIER
MFDID    EQU   X'20'               MFLD ID
         DS    X                  RESERVED
MFDNAME  DS    CL8                FIELD NAME                           [ 2]
MFDCHKN  DS    AL2                CHECK ROUTINE NUMBER                 [ 3]
MFDEDTN  DS    AL2                EDIT ROUTINE NUMBER                  [ 3]
MFDDEFN  DS    AL2                DEFAULT ROUTINE NUMBER               [ 3]
MFDEXPN  DS    AL2                EXPANSION ROUTINE NUMBER             [ 3]
MFDMASK  DS    XL1                MASK (NEGATIVE OF FIELD OPTIONS)
MFDOPT   DS    XL1                FIELD OPTIONS                        [ 4]
MFDQUEUY EQU   X'40'               QUEUE=YES
MFDOPTYS EQU   X'20'               OPTION=YES
MFDMANDY EQU   X'10'               MAND=YES
MFDRSEXT EQU   X'01'               MFD RS EXTENSION AVAILABLE
MFDLTHFL DS    XL1                TYPE OF LENGTH SPECIFICATION         [ 5]
MFDLTHFX EQU   X'08'               FIXED LENGTH SPECIFIED
MFDLTHVR EQU   X'04'               VARIABLE LENGTH SPECIFIED
MFDLTHUN EQU   X'02'               UNLIMITED LENGTH SPECIFIED
MFDSPEC  DS    XL1                OCCURRENCE INDICATORS                [ 6]
MFDSCHK  EQU   X'80'               CHECKING ROUTINE SPECIFIED
MFDSEDT  EQU   X'40'               EDIT ROUTINE SPECIFIED
MFDSDEF  EQU   X'20'               DEFAULT ROUTINE SPECIFIED
MFDSEXP  EQU   X'10'               EXPANSION ROUTINE SPECIFIED
MFDSLTH  EQU   X'08'               LENGTH SPECIFIED
MFDSQ    EQU   X'04'               QUEUE SPECIFIED
MFDSOPT  EQU   X'02'               OPTION SPECIFIED
MFDSMAND EQU   X'01'               MAND SPECIFIED
MFDLTH1  DS    AL2                MINIMUM LENGTH                       [ 7]
MFDLTH2  DS    AL2                MAXIMUM LENGTH                       [ 8]
MFDDAMAX DS    AL2                MAX NUM OF DATA AREAS IN FIELD       [ 9]
MFDRSMAX DS    AL2                MAX NUM OF OCCURRENCES IN REP. SEQ.  [10]
MFDRSMIN DS    AL2                MIN NUM OF OCCURRENCES IN REP. SEQ.  [10]
MFDGRPNO DS    XL1                GROUP NUMBER TO WHICH FIELD BELONGS
MFDOPTSN DS    AL1                NUMBER OF OPTIONS SPECIFIED          [11]
MFDLEN   EQU   *-MFLDDS           BASIC MFLD LENGTH
MFDLITL  DS    AL1                LENGTH OF OPTION LITERAL             [12]
MFDLIT   DS    CLX                LIST OF OPTION LITERALS              [13]
MFDRSXLL DS    AL2(MFDRSXEN-MFDRSXLL)  LENGTH OF MFD RS EXTENSION      [14]
MFDRSXNN DS    AL2                NUMBER OF MAX/MIN PAIRS
MFDRSXGN DS    AL2                STATIC RS GROUP NUMBER
MFDRSXA1 DS    AL2                MAX OCC INDEX IN FIRST REP SEQ
MFDRSXI1 DS    AL2                MIN OCC INDEX IN FIRST REP SEQ
* VARIABLE NUMBER OF RS INDEX FIELDS - GENERATED AS SPECIFIED
MFDRSXA9 DS    AL2                MAX OCC INDEX IN NESTED REP SEQS
MFDRSXI9 DS    AL2                MIN OCC INDEX IN NESTED REP SEQS
MFDRSXEN DS    0C
         ORG   MFDRSXLL
```

Notes:

1. MFDLNGTH

The parameter contains the total length of the MCB entry.

2. MFDNAME

The parameter contains the name of the field to be initialized. The contents of the parameter is compared with the name of the field in the field reference.

3. MFDCHKN, MFDEDTN, MFDDEFN, MFDEXPN

   These parameters contain the numbers of the routines to be assigned to the field. If you specify these parameters, the corresponding occurrence indicator must be set.

4. MFDOPT

   This parameter assigns which options of the field are set. If specified, the corresponding occurrence indicator must be set.

   The option MFDRSEXT indicates that an extension buffer is supplied. This passes the information to initialize a field in a nested repeatable sequence. The sequence of fields initialized in a nested repeatable sequence is critical and is done preferably using the DSLLDEV TYPE=MESSAGE part of an MCB.

5. MFDLTHFL

   This parameter specifies the type of the field length.

6. MFDSPEC

   If you want to define or change a routine number, a field option, or a length parameter, then the corresponding indicator must be set. Only those parameters with set indicator are considered when the field descriptor is created in the TOF.

7. MFDLTH1

   This parameter specifies the minimum length of the data areas of the field to be initialized.

8. MFDLTH2

   This parameter specifies the maximum length of the data areas of the field to be initialized.

9. MFDDAMAX

   This parameter specifies the maximum number of the data areas of the field to be initialized.

10. MFDRSMAX, MFDRSMIN

    These parameters specify the maximum and minimum number of the occurrences of a field belonging to a repeatable sequence. They must be the same for all fields belonging to this repeatable sequence.

11. MFDOPTSN

    This parameter specifies the number of option literals to be assigned to the field to be initialized. If MFDOPTSN=0, any option list supplied is ignored.

12. MFDLITL

    This parameter specifies the length of one option literal. This length is required to interpret the option list correctly, and is part of the option list.

13. MFDLIT

    This parameter specifies the list of the option literals.

14. MFDRSXLL

    This parameter marks the start of the buffer that supplies the information needed to initialize fields in nested repeatable sequences.

The contents of the field descriptor created in the TOF can be received in the buffer referenced by TSVPBUFF using the DSLTSV TYPE=READ,FMODIF=FDSCRPT macro. The information is supplied according to the DSECT shown below.

Note: If you apply the DSLTSV TYPE=READ,FMODIF=FDSCRPT macro on a subfield, the descriptor for the subfield from the FDT is received.

```
TOFFDE   DSECT
TOFFTLEN DS    AL2                ENTRY LENGTH EXCLUDING LENGTH FIELD   [ 1]
TOFFCHKN DS    AL2                CHECK ROUTINE NUMBER                  [ 2]
TOFFEDTN DS    AL2                EDIT ROUTINE NUMBER                   [ 2]
TOFFDEFN DS    AL2                DEFAULT ROUTINE NUMBER                [ 2]
TOFFEXPN DS    AL2                EXPANSION ROUTINE NUMBER              [ 2]
TOFFSEPN DS    AL2                SEPARATION ROUTINE NUMBER             [ 2]
TOFFMASK DS    AL1                MASK (NEGATIVE OF FIELD OPTIONS)
TOFFOPT  DS    AL1                FIELD OPTIONS                         [ 3]
TOFPERMY EQU   X'80'                PERM=YES
TOFQUEUY EQU   X'40'                QUEUE=YES
TOFOPTYS EQU   X'20'                OPTION=YES
TOFMANDY EQU   X'10'                MAND=YES
TOFPADYS EQU   X'08'                PAD SPECIFIED
TOFIN1ST EQU   X'04'                INIT=FIRST
TOFCHSPR EQU   X'02'                CHECKING DONE IN SEPARATION ROUTINE
TOFFRSXT EQU   X'01'                RS EXTENSION AVAILABLE
TOFLTHFL DS    AL1                TYPE OF LENGTH SPECIFICATION          [ 4]
TOFLTHFX EQU   X'08'                FIXED LENGTH SPECIFIED
TOFLTHVR EQU   X'04'                VARIABLE LENGTH SPECIFIED
TOFLTHUN EQU   X'02'                UNLIMITED LENGTH SPECIFIED
         DS    X                  RESERVED
TOFLTH1  DS    AL2                MINIMUM LENGTH                        [ 5]
TOFLTH2  DS    AL2                MAXIMUM LENGTH                        [ 6]
TOFDAMAX DS    AL2                MAX NUM OF DATA AREAS IN FIELD        [ 7]
TOFRSMAX DS    AL2                MAX NUM OF OCCURRENCES IN FIELD       [ 8]
TOFPAD   DS    CL1                PADDING CHARACTER
TOFOPTSN DS    AL1                NUMBER OF OPTIONS SPECIFIED           [ 9]
TOFRSMIN DS    AL2                MIN NUM OF OCCURRENCES IN FIELD       [ 8]
TOFLITL  DS    AL1                LENGTH OF OPTION LITERAL              [10]
TOFLIT   DS    CLX                LIST OF OPTION LITERALS               [11]
TOFRSXLL DS    AL2(TOFRSXEN-TOFRSXLL)  LENGTH OF TOF RS EXTENSION       [12]
TOFRSXNN DS    AL2                NUMBER OF MAX/MIN PAIRS
TOFRSXGN DS    AL2                STATIC RS GROUP NUMBER
TOFRSXA1 DS    AL2                MAX OCC INDEX IN FIRST REP SEQ
TOFRSXI1 DS    AL2                MIN OCC INDEX IN FIRST REP SEQ
* VARIABLE NUMBER OF RS INDEX FIELDS - GENERATED AS SPECIFIED
TOFRSXA9 DS    AL2                MAX OCC INDEX IN NESTED REP SEQS
TOFRSXI9 DS    AL2                MIN OCC INDEX IN NESTED REP SEQS
TOFRSXEN DS    0C
```

Notes:

1. TOFFTLEN

This parameter contains the length of the field descriptor entry except for the length of TOFFTLEN.

2. TOFFCHKN, TOFFEDTN, TOFFDEFN, TOFFEXPN

These parameters contain the numbers of the routines assigned to the field.

3. TOFFOPT

This parameter contains the options assigned to the field.

4. TOFLTHFL

This parameter contains the type of the field length specified.

5. TOFLTH1

This parameter contains the minimum length of the data areas specified. It is checked by TOF basic checking for fixed length and variable length fields.

6. TOFLTH2

This parameter contains the maximum length of the data areas specified. It is checked by TOF basic checking for fixed length and variable length fields.

7. TOFDAMAX

This parameter contains the maximum number of data areas specified for this field. It is checked by TOF basic checking.

8. TOFRSMAX, TOFRSMIN

These parameters contain the maximum and minimum number of the occurrences specified for this field. They are checked by TOF basic checking.

9. TOFOPTSN

This parameter contains the number of option literals assigned to the field.

10. TOFLITL

This parameter specifies the length of one option literal. It is part of the option list and only present if the field has an option list assigned.

11. TOFLIT

This parameter contains the list of the option literals. It is part of the option list and only present if the field has an option list assigned.

12. TOFRSXLL

The start of the buffer that supplies the information needed to initialize fields in nested repeatable sequences.

Below is an example of initializing a field in the TOF:

```
         ...                      PROVIDE MCB ENTRY IN BUFFER
         ...                      REFERENCED BY TSVPBUFF
         DSLTSV TYPE=INIT,        REQUEST TYPE = INIT           *
               NESTID=1,          NESTING IDENTIFIER = 1        * [1]
               FDGPIND=1,         FIELD GROUP = 1               * [2]
               RSINDEX=3,         REP. SEQ. OCC. = 3            * [3]
               FDNAM='MYNAME',    FIELD NAME = 'MYNAME'         * [4]
               MF=(E,TSVPARMS)
         LTR   R15,R15            OK?
         BNZ   ERROR
         ...                      CONTINUE NORMAL
         ...                      PROGRAM PROCESSING
         ...
ERROR    ...
```

Notes:

1. NESTID=1

This parameter sets the DSLTSV parameter list field TSVPNENI to 1. The nesting identifier in the “actual” position is set to NI=1.

2. FDGPIND=1

This parameter sets the DSLTSV parameter list field TSVPNEFG to 1. The field group in the “actual” position is set to FG=1.

3. RSINDEX=3

This parameter sets the DSLTSV parameter list field TSVPNERS to 3. The repeatable sequence occurrence in the “actual” position is set to RS=3.

Note: Because of this specification the field is initialized at least for 3 empty occurrences. If other fields of the same repeatable sequence were already initialized for more occurrences, this field is initialized for as many occurrences as are present.

4. FDNAM='MYNAME'

This parameter sets the DSLTSV parameter list field TSVPNEFN to 'MYNAME ‘. The field name in the “actual” position TSVPCUFN is set to 'MYNAME ‘.

Note: If 'MYNAME ‘ is the name of a subfield according to the FDT, then the request is rejected.

Adding a Nesting Identifier to the TOF

To add a nesting identifier to the TOF, specify an exit field on the preceding identifier that is not already an exit field for a nesting identifier.

You can either use a field already used in the TOF or specify a field only used as an exit field. The exit field must at least be initialized before the 'ADDNI' request can be successfully executed.

Note: The reference of the exit field accessed in a DSLTSV TYPE=ADDNI instruction is evaluated differently from normal processing. Positioning is done using only the “current” position (TSVPCURR) and the modifiers (TSVPMODS). The name of the exit field, however, must be supplied in the next parameter TSVPNEFN.

If the return code from the 'ADDNI' request is not 0, check the reason code. The following more specific reason codes can occur:

```
TOFRFULL   TOF is full, not enough space available
TOFRNEST   No further nesting identifier available, maximum number=255
           for NI already in TOF
TOFRFINI   Field already initialized, not available as exit field
TOFROCCU   Occurrence not found
TOFRFDNF   Field not found.
```

Below is an example of adding a nesting identifier to the TOF:

```
...                     PROVIDE IDENTIFIER FOR            [1]
...                     NESTING IDENTIFIER
...                     IN THE BUFFER 'MYBUF'
DSLTSV TYPE=WRITE,      REQUEST TYPE = WRITE            * [2]
      BUFFER=MYBUF,     MYBUF REFERS TO THE BUFFER      *
      ,                 CONTAINING THE DATA             *
      NESTID=0,         NESTING IDENTIFIER = 0          * [3]
      FDGPIND=1,        FIELD GROUP = 1                 * [3]
      FDNAM='MYEXIT',   FIELD NAME = 'MYEXIT'           *
      OPTION=NO,        DATA AREA TO BE WRITTEN         *
      MF=(E,TSVPARMS)
LTR   R15,R15           OK?
BNZ   ERROR1
DSLTSV TYPE=ADDNI,      REQUEST TYPE = ADDNI            * [4]
      FDGPIND=1,        FIELD GROUP = 1                 * [5]
      FDNAM='MYEXIT',   FIELD NAME = 'MYEXIT'           * [6]
      DAINDEX=1,        DATA AREA INDEX = 1             *
```

```
            MF=(E,TSVPARMS)
          LTR   R15,R15              OK?
          BNZ   ERROR2
          ...                        CONTINUE NORMAL
          ...                        PROGRAM PROCESSING
          ...
ERROR1    ...
          ...
ERROR2    ...
```

Notes:

1. Provides an indication of the data for the nesting identifier.

It is useful to supply an indicator of the type of information to be stored on the new nesting identifier. In the SWIFT Link the message type is supplied.

2. TYPE=WRITE

With this request the nesting identifier indicator is written to the exit field 'MYEXIT' and this field is initialized if it was not in the TOF before.

3. NESTID=0 and FDGPIND=1

The exit field is written on nesting identifier 0 and field group 1.

4. TYPE=ADDNI

This parameter specifies a new nesting identifier of the TOF.

Note: The nesting identifier is not NI=1 if NI=1 was introduced before. It is set to the highest nesting identifier at that time in the TOF + 1. This value is returned in the parameter TSVPNIEX of the TOF parameter list when the exit field is read or accessed.

5. FDGPIND=1

This parameter sets the DSLTSV parameter list field TSVPNEFG to 1. The parameter is not used for evaluating the field reference.

6. FDNAM='MYEXIT'

With the preceding DSLTSV TYPE=WRITE instruction, this field was written to the TOF. Because no modifiers are specified in this DSLTSV macro and the field reference was saved from the preceding request, the exit field will be found.

Note: To make sure that the exit field required is in the TOF, either use a sequence of DSLTSV TYPE=WRITE or DSLTSV TYPE=ACCESS before using the DSLTSV TYPE=ADDNI macros.

Compressing the TOF into a Buffer and Merging the TOF from a Buffer

If you want to remove the free space in the TOF, you can compress the TOF into the buffer referenced by TSVPBUFF using the DSLTSV TYPE=COMPRESS macro.

Note: You lose the fields with the OPTION QUEUE=NO in the FDT or MCB.

If you want to get free space in a TOF supplied in the buffer referenced by TSVPBUFF, you can merge the TOF supplied with an empty TOF using the DSLTSV TYPE=MERGE macro.

Note: The MERGE request adds the fields from the TOF referenced by TSVPBUFF to the TOF referenced by TSVPADDR. If a field already exists, it is overwritten.

If the return code from the 'COMPRESS' or 'MERGE' request is not zero, checking of the reason code is required. The following more specific reason codes can occur:

```
TOFRBUFU   Buffer too small for compressed TOF
TOFRCOEM   TOF to be compressed is empty
TOFRFULL   TOF too small to receive the TOF supplied.
```

Below is an example of compressing and merging a TOF:

```
          DSLTSV TYPE=COMPRESS        REQUEST TYPE = COMPRESS        *
                BUFFER=MYBUF,         MYBUF REFERS TO THE BUFFER     * [1]
                ,                     TO RECEIVE THE COMP. TOF       *
                MF=(E,TSVPARMS)
          LTR   R15,R15               OK?
          BNZ   ERROR1
          DSLTSV TYPE=TOFNEW,         REQUEST TYPE = TOFNEW          * [2]
                MF=(E,TSVPARMS)
          LTR   R15,R15               OK?
          BNZ   ERROR2
          DSLTSV TYPE=MERGE,          REQUEST TYPE = MERGE           * [3]
                MF=(E,TSVPARMS)
          LTR   R15,R15               OK?
          BNZ   ERROR3
          ...                         CONTINUE NORMAL
          ...                         PROGRAM PROCESSING
          ...
ERROR1    ...
ERROR2    ...
ERROR3    ...
```

Notes:

1. BUFFER=MYBUF

With this parameter the buffer referenced by MYBUF is used to receive the compressed TOF.

Note: Supply a buffer large enough to contain a TOF.

2. TYPE=TOFNEW

With this DSLTSV macro the original TOF referenced by TSVPADDR is emptied.

3. TYPE=MERGE

With this DSLTSV macro the empty TOF is merged with the compressed TOF referenced by MYBUF.

Joining the TOF into a Buffer

If you want to join a split TOF, you can join the TOF into the buffer referenced by TSVPBUFF using the DSLTSV TYPE=JOIN macro.

Note: In contrast with the 'COMPRESS' request, the 'JOIN' request does not cause the fields with OPTION QUEUE=NO to be lost.

If the return code from the 'JOIN' request is not zero, checking of the reason code is required. The following more specific reason codes can occur:

```
TOFRBUFU   buffer too small for compressed TOF
TOFRCOEM   TOF to be joined is empty.
```

Below is an example of joining a split TOF:

```
          DSLTSV TYPE=JOIN            REQUEST TYPE = JOIN           * [1]
                BUFFER=MYBUF,         MYBUF REFERS TO THE BUFFER    * [2]
                ,                     TO RECEIVE THE COMP. TOF      *
                MF=(E,TSVPARMS)
          LTR   R15,R15               OK?
          BNZ   ERROR1
          ...                         CONTINUE NORMAL
          ...                         PROGRAM PROCESSING
          ...
 ERROR1   ...
```

Notes:

1. TYPE=JOIN

With this DSLTSV macro the empty TOF is merged with the compressed TOF referenced by MYBUF.

2. BUFFER=MYBUF

With this parameter the buffer referenced by MYBUF is used to receive the compressed TOF.

Note: Supply a buffer large enough to contain a TOF. The required length can be obtained by reading the dynamic TOF settings field TINFJBSZ, as described in "Reading Dynamic TOF Settings" on page 32.

Chapter 7. Using General File Services (DSLFLV)

MERVA ESA provides general file services that enable you to access files from programs running in a MERVA ESA environment. You can use these services to access the SWIFT Correspondents File, the SWIFT Currency Code File, the MERVA ESA Nicknames File, or the Telex Correspondents File.

The following general file services are available:

- Open a file
- Close a file
- Add a record
- Delete a record
- Replace a record
- Get a record by direct access
- Get records by sequential access.

You can request these services by calling the File Service Program DSLFLVP. You use the File Service Macro DSLFLV to call this program.

Requesting general file services requires that the files you refer to are defined in the MERVA ESA file table (DSLFLTT). The files must also be defined to VSAM, and to CICS file control (CICS installations) or DL/I (IMS installations). DSLFLVP translates requests to access a file into VSAM macros and CICS file control commands (CICS installations), or DL/I calls (IMS installations). For the caller of DSLFLVP, there is no difference between CICS and IMS. This is convenient when writing programs for both systems. Also, in CICS installations, batch processing of files is often simplified if you use DSLFLVP instead of calling VSAM directly.

Note: DSLFLVP is DC-system dependent, that is, CICS and IMS environments use different DSLFLVP modules. The module must be loaded from the appropriate load library for your system.

How to define a file in the MERVA ESA file table is described in *MERVA for ESA Customization Guide*.

Using the File Service Macro DSLFLV

The macro DSLFLV is used to call the program DSLFLVP. In the following, the services that you can request from DSLFLVP are described, and examples of how you can specify your requests by DSLFLV macros. For complete information on the DSLFLV macro parameters, refer to *MERVA for ESA Macro Reference*.

Setup for DSLFLVP

If you set up your own MERVA ESA environment and you want to use the services of DSLFLVP, then the DSLCOM area must be addressable and the following fields of DSLCOM must be filled (see “Filling the Fields of DSLCOM” on page 5 for more information):

- COMSRVPA
- COMOMSGA

- COMMSGTA
- COMFLVPA
- COMFLTTA
- COMEIB (CICS only)
- COMEISTG (CICS only)
- COMPCBLA (IMS only).

You communicate with DSLFLVP via the parameter list (PL) and the request control block (RCB), which are both generated by means of a DSLFLV MF=L macro. Also a temporary storage of at least 3072 bytes length is required.

The example below shows how to define the PL, the RCB and the temporary storage for DSLFLVP:

```
          DSLFLV MF=L              PL AND RCB OF DSLFLVP              [1]
VTS       DS     CL3072            TEMPORARY STORAGE OF DSLFLVP
```

Note:

1. The label is omitted, so a default label is generated depending on the PREFIX parameter. As the PREFIX parameter is also omitted, the label default is FLVL, FLV being the default for the prefix parameter. All fields of the PL and the RCB have also the default prefix FLV.

Information Returned

You receive return information in register 15 and in the RCB.

Register 15 always contains the return code. Unless the return code is 16, you get additional return information in the following RCB fields:

FLVRC Return code

FLVRSN Reason code

FLVOMBUF Unless the return code is zero: a diagnostic message that can be printed or displayed.

Return code 16 means that register 1 or at least one of the fields FLVRCBA, FLVTSA, FLVCOMA, or COMOMSGA is hexadecimal zero. You can avoid this error by a correct programming setup.

For a detailed description of return codes, reason codes, and diagnostic messages, refer to *MERVA for ESA Messages and Codes*.

Layouts of Buffers and Records

When you add a record to a file or replace a record in a file, you must provide the new record in a buffer. Figure 2 on page 49 shows the required buffer layout. The same layout is supplied by DSLFLVP when you get a record from a file. The offset of the record within the buffer is always 4. In the first halfword of the buffer, you store the buffer length. DSLFLVP checks this value against the record length defined in the file table when you request a record.

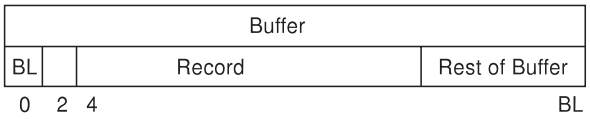


Figure 2. Buffer Layout When Using DSLFLVP

DSLFLVP does not restrict the record layout. However, if you use MFS services for mapping record data to screen/printer devices, your records must have the layout shown in Figure 3. Remember that you get this layout automatically if you provide new record data in the TOF area and use MFS line-formatter services to map them into the buffer.

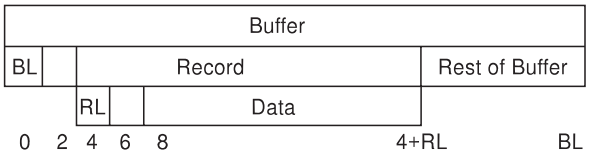


Figure 3. Buffer/Record Layout When Using DSLFLVP and MFS Services

Opening and Closing a File

In IMS batch or message processing programs (BMP and MPP) and in CICS tasks, requests for opening or closing files are not required. If you code such requests, they are ignored and the return code is zero.

In batch programs of CICS installations, you must open and close each file you access. Begin your sequence of DSLFLV macros with DSLFLV TYPE=OPEN, and end it with DSLFLV TYPE=CLOSE. Specify the file name by the parameter DAT.

When you open the file, you must show which request types will follow by specifying or omitting the option GET in the option list (parameter OPT). If the option GET is specified, only GET and GETNEXT request types are subsequently allowed. If GET is omitted, any request type is allowed. If you want to have several regions (MVS) or partitions (VSE) access the same file concurrently, only one of them is allowed to open the file without GET, but all can open it with GET. A batch program that opens a file without the option GET cannot run while the region or partition of MERVA ESA running under CICS has the file opened.

Below is an example of opening and closing a file. The file is opened for all request types.

```
...                     PROVIDE FILE NAME IN VDAT
...
DSLFLV TYPE=OPEN,       REQUEST TYPE: OPEN                *
      DAT=VDAT,         FILE NAME: VDAT                   *
      TS=VTS,           TEMPORARY STORAGE: VTS            *
      MF=E
LTR   R15,R15           OK?
BNZ   ERROR1
...                     ACCESS THE FILE USING
...                     ...DSLFLV MACRO INSTRUCTIONS TYPE=
...                     ...ADD/DELETE/REPLACE/GET/GETNEXT
...
DSLFLV TYPE=CLOSE,      REQUEST TYPE: CLOSE               *
      DAT=VDAT,         FILE NAME: VDAT                   *
      TS=VTS,           TEMPORARY STORAGE: VTS            *
      MF=E
LTR   15,15             OK?
```

```
         BNZ   ERROR2
         ...
ERROR1   ...
ERROR2   ...
         ...
VDAT     DS    CL8               FILE NAME
```

Adding a Record

To add a record to a file, code a DSLFLV TYPE=ADD macro. Specify the file name by the parameter DAT, and the address of the record buffer by the parameter BUF. Below is an example of adding a record to a file. The record provided in VBUF is added.

```
         ...                     PROVIDE FILE NAME IN VDAT
         ...                     PROVIDE NEW RECORD IN VBUF
         ...
         DSLFLV TYPE=ADD,        REQUEST TYPE: ADD                  *
               DAT=VDAT,         FILE NAME: VDAT                    *
               BUF=VBUF,         RECORD BUFFER: VBUF                *
               TS=VTS,           TEMPORARY STORAGE: VTS             *
               MF=E
         LTR   R15,R15           OK?
         BZ    OK
         CLI   1+FLVRSN,FLVRDUP  RECORD DUPLICATED?
         BE    DUP
         B     ERROR
OK       ...
DUP      ...
ERROR    ...
         ...
VDAT     DS    CL8               FILE NAME
VBUF     DS    ...               RECORD BUFFER
```

Deleting a Record

To delete a record from a file, code a DSLFLV TYPE=DELETE macro. Specify the file name by the parameter DAT, and the address of your search argument by the parameter ARG. The record where the search field content is equal to your search argument is deleted.

Below is an example of deleting a record from a file. The record whose search field is equal to the content of VARG is deleted.

```
         ...                     PROVIDE FILE NAME IN VDAT
         ...                     PROVIDE SEARCH ARGUMENT IN VARG
         ...
         DSLFLV TYPE=DELETE,     REQUEST TYPE: DELETE               *
               DAT=VDAT,         FILE NAME: VDAT                    *
               ARG=VARG,         SEARCH ARGUMENT: VARG              *
               TS=VTS,           TEMPORARY STORAGE: VTS             *
               MF=E
         LTR   R15,R15           OK?
         BZ    OK
         CLI   1+FLVRSN,FLVRNOF  RECORD NOT FOUND?
         BE    NOF
         B     ERROR
OK       ...
NOF      ...
ERROR    ...
         ...
VDAT     DS    CL8               FILE NAME
VARG     DS    ...               SEARCH ARGUMENT
```

Replacing a Record

To replace a record in a file, code a DSLFLV TYPE=REPLACE macro. Specify the file name by the parameter DAT, and the address of the record buffer by the parameter BUF. The new record provided in the record buffer will replace the record in the file where the search field content is equal to the search field content of the new record.

Below is an example of replacing a record in a file. The new record provided in VBUF replaces the record in the file where the search field content is equal to the search field content of the new record.

```
         ...                    PROVIDE FILE NAME IN VDAT
         ...                    PROVIDE NEW RECORD IN VBUF
         ...
         DSLFLV TYPE=REPLACE,   REQUEST TYPE: REPLACE            *
               DAT=VDAT,        FILE NAME: VDAT                  *
               BUF=VBUF,        RECORD BUFFER: VBUF              *
               TS=VTS,          TEMPORARY STORAGE: VTS           *
               MF=E
         LTR   R15,R15          OK?
         BZ    OK
         CLI   1+FLVRSN,FLVRNOF RECORD NOT FOUND?
         BE    NOF
         B     ERROR
OK       ...
NOF      ...
ERROR    ...
         ...
VDAT     DS    CL8              FILE NAME
VBUF     DS    ...              RECORD BUFFER
```

Getting a Record by Direct Access

To get a record from a file by direct access, code a DSLFLV TYPE=GET macro. Specify the file name by the parameter DAT, the address of your search argument by the parameter ARG, and the address of the record buffer by the parameter BUF.

Show your search condition by specifying one of the following options in the option list (parameter OPT):

EQ Get the record where the search field content is equal to the search argument. EQ is the default if none of the options EQ, GT, or GTEQ is specified.

GT Get the first record where the search field content is greater than the search argument.

GTEQ Get the record where the search field content is equal to the search argument. If this record does not exist, get the first one where the search field content is greater than the search argument.

Note: You never get the initialization record that was written to the file by DSLFLUT. With search conditions GT/GTEQ you always get the first record that meets your search condition.

Below is an example of getting a record directly from a file. You get the record where the search field content is equal to the content of VARG.

```
         ...                    PROVIDE FILE NAME IN VDAT
         ...                    PROVIDE SEARCH ARGUMENT IN VARG
         ...
```

```
         DSLFLV TYPE=GET,         REQUEST TYPE: GET                  *
               DAT=VDAT,          FILE NAME: VDAT                    *
               BUF=VBUF,          RECORD BUFFER: VBUF                *
               ARG=VARG,          SEARCH ARGUMENT: VARG              *
               TS=VTS,            TEMPORARY STORAGE: VTS             *
               MF=E
         LTR   R15,R15            OK?
         BZ    OK
         CLI   1+FLVRSN,FLVRNOF   RECORD NOT FOUND?
         BE    NOF
         B     ERROR
OK       ...
NOF      ...
ERROR    ...
         ...
VDAT     DS    CL8                FILE NAME
VBUF     DS    ...                RECORD BUFFER
VARG     DS    ...                SEARCH ARGUMENT
```

Getting Records by Sequential Access

To get records by sequential access, code DSLFLV TYPE=GET to get the first record, and code a loop with DSLFLV TYPE=GETNEXT to get the next records. For TYPE=GET, specify GETNEXT in the option list (parameter OPT). This shows that requests of type GETNEXT will follow. For both macros, specify the file name by the parameter DAT, and the address of the record buffer by the parameter BUF.

Below is an example of how to get all records from a file sequentially.

```
         ...                      PROVIDE FILE NAME IN VDAT
         XC    VARG,VARG          ZERO THE SEARCH ARGUMENT
         ...
         DSLFLV TYPE=GET,         REQUEST TYPE: GET                * [1]
               OPT=(GT,GETNEXT),  OPTION LIST: GT, GETNEXT FOLLOWS *
               DAT=VDAT,          FILE NAME: VDAT                  *
               BUF=VBUF,          RECORD BUFFER: VBUF              *
               ARG=VARG,          SEARCH ARGUMENT: VARG            *
               TS=VTS,            TEMPORARY STORAGE: VTS           *
               MF=E
         LTR   R15,R15            OK?
         BZ    LOOP
         CLI   1+FLVRSN,FLVRNOF   RECORD NOT FOUND?                  [2]
         BE    NOF
         B     ERROR
LOOP     DS    0H                 DO UNTIL RETURN CODE IS NOT ZERO
         ...
         ...                        PROCESS THE RECORD DATA
         ...
         DSLFLV TYPE=GETNEXT,       REQUEST TYPE: GETNEXT          * [3]
               DAT=VDAT,            FILE NAME: VDAT                *
               BUF=VBUF,            RECORD BUFFER: VBUF            *
               TS=VTS,              TEMPORARY STORAGE: VTS         *
               MF=E
         LTR   R15,R15
         BZ    LOOP               ENDDO
         CLI   1+FLVRSN,FLVREOF   END OF FILE?                       [4]
         BE    EOF
         B     ERROR
EOF      ...
NOF      ...
ERROR    ...
         ...
VDAT     DS    CL8                FILE NAME
VBUF     DS    ...                RECORD BUFFER
VARG     DS    ...                SEARCH ARGUMENT
```

Notes:

1. This DSLFLV macro gets the record with the lowest search field content if a record is in the file at all.
2. The reason code FLVRNOF ("not found") is returned if the file is empty.

   Note: There is always the initialization record with a search field of hexadecimal zero that is loaded by the File Batch Utility Program DSLFLUT when the file is initialized. You cannot get this record. When only this record is in the file, you still get the reason code FLVRNOF.
3. This DSLFLV macro gets the records after the first one sequentially until the last record in a loop.
4. The reason code FLVREOF ("end of file") is returned when there are no more records in the file.

Chapter 8. Using the Message Format Service (DSLMFS)

The MERVA ESA Message Format Service (DSLMMFS) is called by the various MERVA ESA modules to transform a message according to the requirements of the internal or external medium with which the MERVA ESA module is working. A "medium" in this context is the tokenized form (TOF), a MERVA ESA message queue, a terminal screen, a line printer, a hard-copy printer, or a line buffer. Whenever the message passes from one medium to another, the physical characteristics of each medium must be considered in the transformation of the message.

The medium-oriented tasks of the Message Format Service are controlled by the MCB defined for the message currently being processed.

In the following descriptions no distinction is made about which MERVA ESA component really invokes Message Format Service for a given function; when referring to the module or modules invoking MFS, only the terms "caller" or "calling module" are used.

Whenever you call Message Format Service by the macro DSLMFS, the MFS interface program DSLMMFS is also called.

See the *MERVA for ESA Macro Reference* for more information about the MERVA ESA Message Format Service and the macro DSLMFS.

The DSLMFS Macro

This macro is used in three forms to:

- Map the Message Format Service (MFS) storage areas and control blocks
- Invoke Message Format Service mapping functions, exit programs or general functions
- Build general entry and exit coding for Message Format Service programs and user exits.

The different forms with their operands and parameters are described in the following.

Invoking MFS Service Functions

There are three classes of MFS service calls:

- Message Format Service general service calls
- Message Format Service mapping functions
- Message Format Service exit program calls.

The linkage calls for the different Message Format Service mapping functions and services are described. The functions are divided into several classes specified by the TYPE and MEDIUM parameter. The parameters used for these calls are described later. With some of these parameters certain restrictions apply when using special type/medium combinations. These restrictions are explained in the description of the corresponding parameters. The following gives a definition of all

combinations of TYPE/MEDIUM supported by MERVA ESA Message Format Service. The first component in the definition item is the TYPE code; the second is the MEDIUM code.

MFS General Functions:

INIT MFS	Initialize Message Format Service
TERM MFS	Terminate Message Format Service
ERRMSG MFS	Build and issue MFS error messages
GETDEV MFS	Get an MCB device description
GETMTT MFS	Get a message type table entry
GETPFK MFS	Get a program function key table
COMMAND MFS	Execute a screen command.

MFS Message-Processing Functions:

INIT MESSAGE	Initialize a message or nesting identifier in the TOF
GET QUEUE	Move a message from queue buffer into the TOF
PUT QUEUE	Move a message from the TOF into a queue buffer
CHECK MESSAGE	Check the contents of a message
EXPAND MESSAGE	Expand fields in a message.

MFS Mapping Functions:

INIT LDS	Initialize the Logical Data Stream (LDS)
GET SCREEN	Format screen input data for an LDS
PUT SCREEN	Format LDS for a screen device
PUT SYSOUT	Format LDS for the system printer
PUT HARDCOPY	Format LDS for a hardcopy printer
GET LDS	Map LDS into TOF
PUT LDS	Build LDS from TOF data
GET NET	Map a network buffer into the TOF
PUT NET	Build a network buffer from TOF data
GET ELFORM	Map an external line format to the tokenized format
PUT ELFORM	Map a tokenized format to the external line format.

MFS Screen Supporting Functions:

GET NOPR	Map NOPROMPT network buffer into TOF
PUT NOPR	Build a NOPROMPT network buffer.

MFS Exit Programs:

CHECK FIELD	Check a message field
DEFAULT FIELD	Default setting for a field
EDIT FIELD	Edit field data

EXPAND FIELD	Expand a message (address) field
SEPR FIELD	Separate and process subfield data
USER MFS	Call a MFS user exit.

General MFS Linkage Description

The MERVA ESA Message Format Service (MFS) is provided as a distributed service for message generation, display, mapping, and data manipulation. The Message Format Service functions and services are defined in the Message Format Service program table DSLMPTT. From this table the interface and selection module DSLMMFS retrieves the necessary linkage information according to the request type and medium. For high-level language user exits the required language environment is also defined in the DSLMPTT.

All Message Format Service components are reentrant. It is possible for different applications to use only one copy of the module DSLMMFS by loading it into virtual storage prior to the first request for an MFS function.

Calling Message Format Service Components

Depending on the type of MFS request, the MFS main routine DSLMMFS selects the component servicing the specific request. A component, in this context, is either a separate program or an entry to a program. The latter allows a routine used for different purposes to have common processing paths. (Refer to "Coding MFS Exit Programs" on page 75.)

After the component has been selected, the linkage is resolved and the module is, if necessary, loaded into virtual storage. Whether a program is to be loaded at execution time or is link-edited to the module DSLMMFS is specified by the component's entry in the Message Format Service program table DSLMPTT.

When a high-level language user exit is to be executed, DSLMMFS transfers control to the MERVA ESA exit manager DSLXMGR which in turn prepares the language environment and calls the user exit program.

The size of the temporary working storage is retrieved from the program header, that has the same structure in all Message Format Service components. This storage is acquired and is cleared to X'00'. For the definition of this storage area refer to Message Format Service Exits and User Exits, "Coding MFS Exit Programs" on page 75.

Before returning to the calling program, this storage area is released.

The address of the temporary working storage is passed in the Message Format Service permanent storage (MFSPTSA) to the MFS component. On return from the MFS component, the calling program should inspect the return code and the reason code. Depending on the reason code, an appropriate error message is selected and written to the TOF field DSLERR.

Since the temporary working storage is released, it is not possible to use it for passing parameters, results, or data to the calling program. For this purpose buffers and data areas must be provided by the calling program and their addresses passed in the MFS parameter list to the MFS component.

Calling Message Format Service Components from MFS Components or Exits

In any MFS component, another MFS component can be called as described above. To achieve this, a copy of the Message Format Service parameter list with the initial parameter values is provided in the temporary working storage. This parameter list can be changed by an “internal” Message Format Service request. All storage areas or buffers provided by the caller of a Message Format Service component are, therefore, also available to “internally” called Message Format Service components.

The nesting depth is limited only by the storage available for running application programs. When writing Message Format Service components, you must make sure that endless recursion does not occur.

Message Format Service Error Messages

This component evaluates the reason code and possibly generates an error message. The error message is stored in the MFS permanent storage and in the TOF field DSLERR, when the option ERRMSG is requested in the parameter list.

Network-Dependent Error Messages

The first three letters of the message identifications are taken from the internal environment information. This means that DSLMMFS prepares network-dependent error messages; that is, DSL3... for MERVA ESA programs, DWS3... for the SWIFT Link, and ENL3 ... for the Telex Link.

Support for Languages other than US English for Error Messages

The language identification for the current session is used for preparing the error message. This language identification can be switched by the **form** command.

Establishing the MFS Environment in an Application Program

An application program using the Message Format Service must provide storage areas and addresses of the MERVA ESA service modules. The required addresses must be available before the first Message Format Service request.

Storage Areas Used by the MFS Functions

For correct execution of all Message Format Service functions, an application program must define storage areas and insert the addresses of control blocks and service modules. These addresses are passed to Message Format Service using the appropriate parameters in the DSLMFS macros for the Message Format Service request. A detailed description of the macro DSLMFS and its parameters can be found in the *MERVA for ESA Macro Reference*. A description of the parameters for each type of storage area or control block follows.

Addresses Used in the MERVA ESA Communication Area (DSLCOM)

Calling DSLMMFS requires some fields of DSLCOM being filled to make the MERVA ESA services available. The same addresses can then be used in Message Format Service programs and exits.

The DSLCOM fields must be initialized again when, for example, a conversational transaction running in an IMS Message Processing Region is scheduled again because these fields are empty.

See "Filling the Fields of DSLCOM" on page 5 for more information on how to fill these fields.

The following fields of DSLCOM must be filled before calling DSLMMFS:

- COMPRMA
- COMSRVPA
- COMTSVA
- COMFDTA
- COMOMSGA
- COMMSGTA
- COMMFSA
- COMMTTA
- COMTRAPA
- COMTUCBA (only when screen or printer mapping is requested).

Programs using Message Format Service in a CICS installation must also provide the fields:

- COMEIB
- COMEISTG.

Programs using the DSLMFS TYPE=EXPAND function for MERVA ESA general file accesses must also provide the fields required for DSLFLVP:

- COMFLVPA
- COMFLTTA
- COMPCBLA for IMS installations only.

DSLMMFS supplies the following fields in DSLCOM after the DSLMFS TYPE=INIT call:

- COMMFSMA
- COMMTBA.

The MERVA ESA MFS Parameter List

The application program must allocate storage for the Message Format Service parameter list, which is used for all Message Format Service requests.

This is done by using the following macro:

```
DSLMFS TYPE=MAP,MF=L
```

In this DSLMFS macro, both the label and the PREFIX parameter are omitted. Therefore a label MFSL is generated, and all field names of the parameter list start with the prefix default MFS.

If several Message Format Service parameter lists are required, they are allocated by additional macros, all specifying a different PREFIX parameter (for example, PREFIX=MF2).

The address of the Message Format Service parameter list is passed to MFS by:

```
DSLMFS TYPE=...,MF=(E,MFSL)
```

The name MFSL for the parameter list in the second subparameter of the MF parameter must be changed when another MFS parameter list is used, that is, to

the PREFIX parameter value of the definition of this parameter list definition concatenated with L. A PREFIX=MF2 parameter results in a parameter list name MF2L.

Message Format Service Permanent Storage

Each application program must allocate MFS permanent storage which is used for all MFS requests.

This storage contains address fields, buffers, and values used for:

- Linking to MFS functional modules and exits
- Storage maintenance
- Loading MCBs and passing descriptor addresses to the requesting program
- Building informational and error messages for all Message Format Service programs and user exits
- Holding status information for Message Format Service screen formatting programs that are called several times when a message is processed
- Internal save area.

Message Format Service permanent storage is allocated by:

```
DSLMFS TYPE=MAP,MF=PS
```

Programs using screen and printer services must allocate the amount of Message Format Service (MFS) permanent storage specified by the MFSSTOR parameter in the MERVA ESA customizing parameter module DSLPRM. The length of the required area is in field NPMFSPS.

```
L     R2,NPMFSPS                GET REQUIRED LENGTH
DSLSRV TYPE=GETMAINA,SIZE=(R2)
L     R2,SRVSADDR               ADDRESS OF STORAGE
USING MFSPS,R2
```

The first halfword of the MFS permanent storage must be set to the length of this storage before the DSLMFS TYPE=INIT call.

MFS Temporary Storage

Each application program must also allocate Message Format Service temporary storage, which is used by the MFS interface module DSLMMFS during the linkage to Message Format Service programs. The size of the MFS temporary storage pool is specified in the MFSSTOR parameter in the MERVA ESA customizing parameter module DSLPRM. The length of the required area is in field NPMFSTS.

For nested MFS calls the temporary storage pool is reused; the temporary storage areas are chained and used as a stack. If the temporary storage pool is too small or exhausted, the MFS interface program allocates additional storage dynamically.

The temporary storage contains a save area for the status of the interface register at linkage time to the Message Format Service program, TOF supervisor and internal Message Format Service parameter lists, and the TOF supervisor work buffer.

For debugging purposes a trace area is provided that contains status information.

Message Format Service temporary storage is allocated by:

```
L     R3,NPMFSTS                GET REQUIRED LENGTH
DSLSRV TYPE=GETMAINA,SIZE=(R3)
L     R3,SRVSADDR               ADDRESS OF STORAGE
USING MFSTS,R3
```

Note: The first halfword of the MFS temporary storage must be set to the length of this storage in front of the DSLMFS TYPE=INIT call. When DSLSRV service was used to allocate the storage area the length field is already set.

The addresses of the permanent and temporary storage areas are passed to MFS by:

```
DSLMFS TYPE=...,PS=MFSPS,TS=MFSTS
```

The Terminal User Control Block (DSLTUCB)

The Terminal User Control Block (TUCB) contains terminal-dependent and user-dependent information mostly used by MERVA ESA transactions invoking Message Format Service. For example, MERVA ESA End-User Driver, and MERVA ESA Hard Copy Printer.

```
DSLMFS TYPE=MAP,MF=TUCB
```

Return Information from Message Format Service

On return from a Message Format Service request, a return code in general register 15, together with a reason code in the parameter list field MFSLREAS, shows the completion status. The return and reason codes are explained in the *MERVA for ESA Messages and Codes*.

RC	Reason	Meaning and Action
0	—	Successfully completed; continue.
4	ANY	Warning, error message in DSLERR and MFSPEMSB.
8	ANY	Incomplete: error message in TOF field DSLERR. The program can continue or end with this error message.
12	ANY	TOF error : error message in MFSPS (MSFPEMSB). The program should take a dump and end with this error message.
16	ANY	MFS storage areas or the content of general registers are not usable. The program should end with a dump.

Calling Message Format Service Programs

DSLMINIT—Initialize MFS

This MFS program is called by the macro DSLMFS where the following parameters are specified:

```
[name]   DSLMFS TYPE=INIT,MEDIUM=MFS,                    *
               MF=....,                                  *
               PS=....,                                  *
```

```
TS=....,                                                    *
TOF=....,                                                   *
...
...
```

This request must precede any other Message Format Service request.

All parameters shown here must be specified.

This request initializes the Message Format Service parameter list with the addresses of storage areas and buffers required when calling Message Format Service programs. These addresses are used by each following Message Format Service request unless they are redefined in an appropriate DSLMFS macro.

In addition, the MFS permanent storage is cleared to X'00' and the address fields are initialized.

DSLMTERM—Terminate MFS

This MFS program is called using the macro DSLMFS where the following parameters are specified:

```
[name]   DSLMFS TYPE=TERM,MEDIUM=MFS,                       *
               MF=....
```

This call must be the last MFS request in a processing cycle, for example, in a batch application or a session with a conversational transaction.

It releases all MCBs loaded during the session and, therefore, listed in the MCB load table and the MCB load table itself, when the load table has been acquired dynamically by DSLMMFS. When the high-level language interface was initialized before it is terminated by this call.

DSLMTIN—Message Initialization in the TOF

To initialize DSLMTIN:

- Initialize a new message in the TOF
- Move message from MERVA ESA queue buffer into the TOF
- Move message from TOF to MERVA ESA queue buffer.

Message Initialization

The module initializes the TOF for a new message. The message description of the MCB for the message for which the TOF is to be initialized is used for the initialization of the field descriptors. The TOF can be initialized for only one message type at a time, that is, if a message contains nested message types or different nesting level indicators, DSLMTIN must be called separately for each message type or nesting level indicator.

When a message is initialized, the old message fields in the TOF are deleted, unless OPT=CONT is specified. With OPT=CLEARPERM, the permanent fields can also be deleted from the TOF.

The MSGID provided by the caller is written to the appropriate nesting level indicator (exit field), as defined in the MTT entry.

On completion of DSLMTIN, the TOF is ready to accept data for this message type.

This Message Format Service routine is called using the macro DSLMFS where the following parameters are specified:

```
[name]   DSLMFS TYPE=INIT,MEDIUM=MSG,                          *
               MF=....,                                        *
               TOF=....,                                       *
               MSGID=...,                                      *
               FLD=...,                                        *
               OPT=...
```

This Message Format Service program initializes the TOF for a message type to be created or for a message type on a new nesting identifier.

The message type is specified in the 8-byte MSGID field setup by the calling program.

OPT=(NXTID) specifies that a nesting identifier is logically added to a message in the TOF. The FLD= parameter specifies the exit field name to be used for the new nesting identifier.

OPT=(CONT) specifies either that a nesting identifier is added to a new exit field, or that additional information is put on an existing nesting level in the TOF (old data remains in the TOF).

From MERVA ESA Queue to TOF

This component transfers a message from the queue buffer to the TOF. If the queue buffer contains UMR data, it is written into the TOF with field name DSLUMR.

On completion, all QUEUE=YES fields of the message are back in the TOF in exactly the format they had before they were moved into the queue buffer.

All QUEUE=NO fields on TOF level 0 that were in the TOF before the transfer are unchanged.

In MERVA ESA, the queue buffer may be larger than 32KB.

From TOF to MERVA ESA Queue

This component maps a message, without its QUEUE=NO fields (if any), from the TOF into the queue format. If the TOF contains a UMR field DSLUMR, this UMR information is transferred into the queue buffer.

On completion, the message is in the queue data buffer, and it can be presented to the MERVA ESA queue management program for storage in the queue data set. The message in the TOF remains unchanged.

In MERVA ESA, the queue buffer may be larger than 32KB. The dynamic buffer option may be used to map large messages from the TOF into the MERVA ESA queue format. When a message is too large to fit into the provided output buffer, the service program acquires a buffer large enough to hold the message and passes this buffer to the caller. This service is called using the following technique:

```
         L     R4,BUFADDR          Get current buffer address
         DSLMFS TYPE=PUT,MEDIUM=QUEUE,MF=(E,MFSL),                *
               OUTBUF=(R4),OPT=DYNBUF
         C     R4,MFSLOBUF         New buffer returned?
         BE    LABEL               ..No, use current buffer
         DSLSRV TYPE=FREEMAIN,ADSTOR=(R4),SIZE=0  Free current buffer
         L     R4,MFSLOBUF         Get new dynamic buffer address
         ST    R4,BUFADDR          Save new buffer address for future
LABEL    DS    0H
```

Line Formatter Program

The Message Format Service line formatter program (DSLMLFP) is used to map a message from a line or network buffer into the TOF or vice versa. The functions GET and PUT are supported by the line formatter program.

The operation of DSLMLFP is controlled by the line format description in the MCB for the message being processed. The line formatter program is invoked when an MFS service for MEDIUM=NET is requested.

An auxiliary function for the line formatting process is the message type determination routine. There is a specific message type determination routine for each network. These routines are invoked by the MERVA ESA message type determination routine.

Mapping from TOF to Line Buffer

PUT is specified when data is to be mapped from the TOF into a network buffer according to message and line format identifications. One or more message identifications (MCBs) can be used to process a message. This depends on the exit statements in the MCB definition and the initialization of the message. The starting message identification used by the formatting program is either the message identification parameter or the actual message identification of the first nesting identifier initialized in the TOF. The information is used to determine an MCB and a line (net) device description within this MCB.

The mapping is done by processing an MCB net device description item by item. An item to be mapped can be a tag, a separator, or a data component. Conditions are evaluated accordingly.

The data contained in the various TOF fields is read and transferred to the line buffer. DSLMLFP adds control characters as defined in the MCB. The message does not contain any frame control characters required for its transmission over a network. These characters are added by the appropriate network link program.

This MFS program is called using the macro DSLMFS where the following parameters are specified:

```
[name]   DSLMFS TYPE=PUT,MEDIUM=NET,                          *
               MF=....,                                       *
               TOF=....,                                      *
               MSGID=...,                                     *
               OUTBUF=...
```

This MFS program transfers a message contained in the TOF to an output buffer and converts it into the network format as defined in the DSLLDEV TYPE=NET section of the MCB describing the message.

The output buffer receives only the nonempty fields of the message together with the field tags and separators defined in the corresponding DSLLNFLD macros of the DSLLDEV TYPE=NET descriptor in the MCB.

In MERVA ESA, the network buffer may be larger than 32KB. The dynamic buffer option may be used to map large messages from the TOF into the MERVA ESA line format. When a message is too large to fit into the provided output buffer, the service program acquires a buffer large enough to hold the message and passes this buffer to the caller. This service is called using the following technique:

```
         L     R4,BUFADDR          Get current buffer address
         DSLMFS TYPE=PUT,MEDIUM=NET,MF=(E,MFSL),                       *
               OUTBUF=(R4),OPT=DYNBUF
         C     R4,MFSLOBUF         New buffer returned?
         BE    LABEL               ..No, use current buffer
         DSLSRV TYPE=FREEMAIN,ADSTOR=(R4),SIZE=0  Free current buffer
         L     R4,MFSLOBUF         Get new dynamic buffer address
         ST    R4,BUFADDR          Save new buffer address for future
LABEL    DS    0H
```

Mapping from Line Buffer to TOF

The input data is mapped into TOF according to an MCB device type net description. The assignment of data in the buffer to TOF fields is made according to field tags. Tags can also be connected to exit points in MCBs; these exit points define inclusion of other MCBs. The line formatter program scans the buffer contents and tries to find a matching tag in the MCB definition. If a tag is not found, the component is mapped as an additional data area of the previous field. In MERVA ESA, the network buffer may be larger than 32KB.

Note: This Message Format Service program is also called internally by the Message Format Service screen formatting services when mapping a message in NOPROMPT mode. Tags included by editing on the screen in NOPROMPT mode that do not match the message structure as defined in the MCB can cause wrong assignments of data to data areas or fields.

The determination of message IDs at the beginning or at exit points is done by specific exit programs.

This Message Format Service program is called using the macro DSLMFS where the following parameters are specified:

```
[name]   DSLMFS TYPE=GET,MEDIUM=NET,                                   *
               MF=....,                                                *
               TOF=....,                                               *
               MSGID=...,                                              *
               INBUF=...,                                              *
               OPT= ( CHECK | CLRPERM | CONT )
```

It is used to transfer a message contained in a line buffer to the TOF as defined in the DSLLDEV TYPE=NET section of the MCB describing the message.

The input buffer remains unchanged after the mapping process.

Normally, if an old message was contained in the TOF, it is deleted before the message in the line buffer is mapped from the line buffer to the TOF. If OPT=CONT is specified in the call, the old message is kept in the TOF and the new message added to the existing one (on a separate nesting level indicator, for example). Internal TOF fields are only deleted from the TOF if OPT=CLRPERM is specified in the call.

To determine the message type to be processed, either message determination is done by a message determination program in the MFS user exit 054 or, if provided, the 8-byte MSGID field is used for formatting.

If the message type cannot be determined or if the message type set up by the determination algorithm cannot be found in the message type table, or if no appropriate MCB or DSLLDEV TYPE=NET,ID=... can be loaded, a default message

type 0DSL is initialized and the complete buffer is transferred to the TOF field DSLLFBUF. This message is accepted by other Message Format Service and MERVA ESA message-processing stages.

With OPT=CHECK, the complete message in the TOF is checked for syntactic and semantic errors. A possible error indication (reason code) is returned to the calling program together with a return code (RC=4).

Update Functions

The line formatting program can be called with OPT=CONT to add input data to an existing TOF (defined, for example, in a separate MCB and for a different nesting level indicator).

External Line Format for Messages

The external line format for messages is supported through a simplified tokenized format by mapping the line format into one or more data areas of a single field. The field name is DSLELF.

As the message in external format is stored as one or more data areas in the TOF, the standard TOF processing can be applied to the message in this format. Screen display and printing in noprompt format is standardly supported. There is no limitation in adding system information to fields in the TOF.

The system field DSLTYPE contains the message identifier and the line format identification when the message in external line format has been created using the MFS mapping services.

The external line format can be stored exclusively or together with the fully tokenized format of the message. In the latter case the message is effectively stored twice. All fields or message parts are available to an application in the appropriate format without mapping. Synchronization of both formats in the TOF is the responsibility of the application using the TOF.

Support of messages in external line format allows a much faster processing of messages because the mapping step can be skipped. But this means that messages must be syntactically correct when delivered by an application program in this form.

The Message Format Service formatting program (DSLMLEF) is used to map a message from or to the external line format in the TOF. The message is mapped within the TOF; an optional line format may be generated in an intermediate buffer. The functions GET and PUT are supported. The line formatter program is invoked when an MFS service for MEDIUM=ELFORM is requested.

Mapping from Tokenized Format to External Line Format

PUT is specified when data is to be mapped from a tokenized format to the external line format.

DSLMLEF is called using the macro DSLMFS with the following parameters:

```
[name]   DSLMFS TYPE=PUT,MEDIUM=ELFORM,                          *
               MF=....,                                          *
               TOF=....,                                         *
               MSGID=...,                                        *
               OUTBUF=...,                                       *
               OPT= ( CONT )
```

This MFS program converts a message contained in the TOF in tokenized format into external line format.

Mapping from External Line Format to Tokenized Format

GET is specified when data is to be mapped from the external line format to the tokenized form.

DSLMLEF is called using the macro DSLMFS with the following parameters:

```
[name]   DSLMFS TYPE=GET,MEDIUM=ELFORM,                          *
               MF=....,                                          *
               TOF=....,                                         *
               MSGID=...,                                        *
               OUTBUF=...,                                       *
               OPT= ( CONT )
```

This call is used to convert a message in the TOF in external line format to a message in tokenized format. The option CONT specifies that the original format, either the external or the tokenized format, is not overwritten, but stays in the TOF. Using this option, both formats can be generated in the same TOF.

MFS Mapping for Screens and Printers

You can use the DSLMFS macro to map data from a TOF to a display device output buffer, or from a screen input buffer to a TOF.

```
(1) DSLMFS    TYPE=INIT,MEDIUM=LDS

(2) DSLMFS    TYPE=PUT,MEDIUM=LDS
             /*  BUILDS THE LDS FOR A COMPLETE
                   PRINTER  PAGE*/

(3) DSLMFS    TYPE=PUT,MEDIUM=HARDCOPY
              Print the segment in buffer
    IF        MFS REASON CODE=MFSRESRE  (407)
              /* MORE SEGMENTS AVAILABLE     */
                 THEN CONTINUE WITH (3)
               UNTIL REASON CODE=0
              /* THEN ALL SEGMENTS OF THE
                 PAGE ARE PRINTED            */

    DSLMFS    MEDIUM=MFS,TYPE=COMMAND
              /* EXECUTE  PAGE+1             */
    IF           MFS RETURN CODE=0
              /* THEN CONTINUE WITH (2)
              /* BUILD THE LDS FOR THE NEXT PAGE */

    IF        MFS REASON CODE=MFSRENAC    (419)
                 THEN NO ACTION
              /* END OF MESSAGE             */

    GET NEXT MESSAGE, CONTINUE WITH (1)
```

To prepare data for a display device, a program must carry out two steps:

1. Create a page of data as a logical data stream:

   ```
   DSLMFS MEDIUM=LDS,TYPE=PUT,...
   ```

2. Transform the logical page into a device-specific physical data stream in the I/O buffer:

   ```
   DSLMFS MEDIUM=device,TYPE=PUT,...
   ```

To receive data from an input buffer, the steps are reversed:

1. Get the physical data stream from the I/O buffer to update the LDS:

```
DSLMFS MEDIUM=SCREEN,TYPE=GET,...
```

2. Transfer the data to the TOF via the LDS:

```
DSLMFS MEDIUM=LDS,TYPE=GET,...
```

When a complete page has been sent or received, call the MFS command interpreter to request next page positioning. The command interpreter return code shows if there are no more pages in the message:

```
DSLMFS MEDIUM=MFS,TYPE=COMMAND,...
```

Data Areas

All executions of the DSLMFS macro require MFS permanent and temporary storage. To use screen and printer mapping, the caller must also provide the following data areas:

- DSLCOM, where the field COMTUCBA must contain the address of a valid TUCB.
- A TUCB, where device data must be complete and the field TUCBLDSA must contain the address of the LDS buffer. If the retype feature is in use, the field TUCBRKYA must contain the address of the retype buffer.
- An LDS buffer, where the size is normally equal to the field NPNICBUF in DSLPRM.
- An MFS edit buffer, where the size is normally equal to the size of the I/O buffer.
- A device I/O buffer, where the size is dependent on the device type. If the Terminal Feature Definitions table has been established, the I/O buffer size may be taken from the field TFDBUFSZ.
- A retype buffer, where the size is equal to the field NPMFSRK in DSLPRM.
- A TOF storage area.

MFS Message Mapping for a Logical Data Stream

The DSLMFS macro is used to transform data from the TOF to a logical data stream (LDS), or from a screen via an LDS to a TOF.

The LDS is always associated with a data buffer. For output, the MFS edit buffer is required. For input, the screen I/O buffer is used.

The macro transforms one page of data, depending on the device specification in the TUCB.

This form of the DSLMFS macro is also used to initialize the LDS work areas.

Calling MFS Internal Functions

This MFS program is called by the macro DSLMFS where the following parameters are specified:

```
[name]   DSLMFS TYPE=CHECK,MEDIUM=MSG,                          *
               MF=....                                          *
               [,TOF=....]
```

All fields of a message in the TOF are checked for errors of syntax and semantics, according to the checking routine supplied with the CHECK operand in the field specifications defined in the appropriate entries in the DSLFDTT or in the message description (DSLLDEV TYPE=MESSAGE) of the appropriate MCB or MCBs.

If no checking routine is specified a basic checking for the message fields is carried out according to the characteristics defined in the DSLLFLD macros or the DSLLMFLD macro instructions of the DSLLDEV TYPE=MESSAGE section of the MCB. The checking of the field contents is always carried out by DSLMCHE when a message is completely mapped to the TOF.

Checking is done in a sequence of processing steps:

1. Checking for permitted message type and message nesting as specified in the MERVA ESA Message Type Table.
2. Checking the contents of each field in the message.

   Calling of the Message Format Service field checking programs (DSLMCnnn) to check the field contents according to special, message-dependent criteria. "nnn" is specified by CHECK=nnn parameter of the DSLLFLD macro or the DSLLMFLD macro in the DSLLDEV TYPE=MESSAGE section of the MCB. If no checking routine is supplied, a standard checking of the field characteristics is carried out by the TOF supervisor DSLTOFSV. The following features are checked if applicable:

   - Checking for missing fields. Mandatory fields are defined by the MAND=YES parameter of the DSLLFLD macro or the DSLLMFLD macro in the DSLLDEV TYPE=MESSAGE section of the MCB.
   - Checking of the number of data areas against the definition of the DAMAX=nnn parameter of the DSLLFLD macro in the FDT or the DSLLMFLD macro in the DSLLDEV TYPE=MESSAGE section of the MCB.
   - Checking of the number of occurrences of a field within a repeatable sequence against what is defined in the REPSEQ= parameter in the DSLLUNIT macro of the DSLLDEV TYPE=MESSAGE section in the MCB.
   - Checking of the data length against what is defined by the LENGTH parameter of the DSLLFLD macro in the FDT or the DSLLMFLD macro instruction in the DSLLDEV TYPE=MESSAGE section of the MCB.
   - Checking of the option against what is defined by the OPTION=YES,OPTLIST=(...) parameters of the DSLLFLD macro instruction in the FDT or the DSLLMFLD macro in the DSLLDEV TYPE=MESSAGE section of the MCB.
3. Calling of a message-type-specific exit, specified in the Message Type Table (DSLMTT TYPE=ENTRY,CHECK=...).
4. Calling of the MFS user exit DSLMU009 to check the message according to specifications of the user, for example by calling one or more Message Format Service checking programs DSLMCnnn that check more than a single field and, therefore, cannot be referred to as described here.

The result of the check is provided as follows:

- The returned reason code is set accordingly.
- If OPT=ERRMSG is specified, a general error message is written to the TOF field DSLERR.
- Error messages (DSLERR) in the message are recorded in the field DSLMSG. These error messages show the specific errors found for the fields checked and can be viewed during processing of messages on a terminal screen using the **show 0err** command. Normally these error messages are network-dependent.

  Note: The field DSLMSG is written to the TOF using the function modifier FMODIF=IGNORE. For example, identical error messages are recorded only once and editing of these error messages via an editing routine (FMODIF=DEEDIT) is not possible.

DSLMXPND—Field Expansion of a Complete Message

All fields of a message are checked if an expansion exit is specified by the EXPAND parameter of the DSLLFLD macro instruction in the FDT or the DSLLMFLD macro in the DSLLDEV TYPE=MESSAGE section of the MCB.

Expansion can be invoked by MERVA ESA during processing of messages on terminal screens (DSLEUD), or during processing of the checking and expansion transaction DSLCXT, depending on the specification of the EXPAND and EXPNAM parameters of the DSLFNT macro. You can invoke expansion from your application program using the DSLMFS TYPE=EXPAND call.

MERVA ESA provides the following expansion exits:

DWSMX001 SWIFT Address Expansion

DWSMX002 Initialization of nested SWIFT messages

DWSMX003 Automatic generation of SWIFT field options.

Expansion is invoked by the following DSLMFS macro:

```
[name]   DSLMFS TYPE=EXPAND,MEDIUM=MSG,                    *
               MF=....                                     *
               [,TOF=....]
```

Calling MFS Data Manipulation Programs and Exits

Note: The *nnn* used in the exit and program names refers to the MODNUM used to call the program or exit.

DSLMCnnn—Checking the Data of a Field

This type of exit is used to check the content of the field for completeness and whether the data fulfills user-defined specifications.

The program is called by the macro DSLMFS where the following parameters are specified:

```
[name]   DSLMFS TYPE=CHECK,MEDIUM=FIELD,                   *
               MF=....,                                    *
               MODNUM=nnn,                                 *
               INBUF=...,                                  *
               FLD=....                                    *
              [,TOF=....]
```

A field in the TOF is checked using the user-written field-checking module DSLMCnnn, where nnn is specified by the parameter MODNUM. For the internal use of this Message Format Service request, refer also to the description of:

- The specification of DSLTSV TYPE=WRITE|READ|ADDDA, FMODIF=CHECK|DECHECK|EDITCHK
- CHECK=nnn parameter of the DSLLFLD macro (FDT)
- CHECK=nnn parameter of the DSLLMFLD macro, DSLLDEV TYPE=MESSAGE (MCB).

For the special conventions to be considered for this type of Message Format Service exit, refer to “Coding MFS Exit Programs” on page 75.

The following standard checking routines are provided by MERVA ESA:

```
                                                          MODNUM
ALPHA     alphabetic character set                       (nnn=901)
NUMERIC   character set 0-9                              (nnn=902)
ANUM      alphameric character set                       (nnn=903)
HEX       hexadecimal character set                      (nnn=911)
YYMMDD    for dates in the form of year month day        (nnn=904)
YYDDD     for dates in the form of year day              (nnn=905)
MMDD      for dates in the form of month day             (nnn=906)
DD        for dates in the form of day                   (nnn=907)
HHMM      for times in the form of hours minutes         (nnn=908)
HH        for times in the form of hours                 (nnn=909)
MM        for times in the form of minutes               (nnn=910)
LABEL     alphanumeric with first character alphabetic   (nnn=912)
EDIFACTA  UN EDI character set A                         (nnn=913)
YYYYMMDD  for dates in the form of 4-digit year month day (nnn=914)
YYYYDDD   for dates in the form of 4-digit year day      (nnn=915)
```

SWIFT Field Checking

The following standard checking routines are provided by MERVA ESA SWIFT Link:

- CHECK=1004 Standard Checking.

  This provides checking equivalent to the checks made by the SWIFT network.

- CHECK=1001 Extended Checking.

  This provides checking that is equivalent to the checks made by the SWIFT network (exactly the same as CHECK=1004), and in addition for INPUT messages that are not nested, the following extra checks are made:

  Field 18
  The number matches the number of occurrences.

  Field 22
  The first subfield (CODES) is checked for a valid code word. This code word is dependent on the message type.

  Note: The check is only made for messages for which the S.W.I.F.T. User Handbook states: *The following code words **must** be used*.

  Field 23
  All code words in the field are checked. If the field contains two or more code words the presence of a slash to separate the code words is verified. The code words are dependent on the message type.

  Note: The check is only made for messages for which the S.W.I.F.T. User Handbook states: *The following code words **must** be used*.

  Field 26 Option F Message Type 305
  The field is checked for a valid code word.

  Field 26 Option H Message Type 516
  The field is checked for a valid code word.

  Field 26 Options I and L
  The field is checked for a valid code word.

  Field 37 Options A, B, C, D, E, and F
  The Interest Payment subfield is checked for valid code words. The code words are dependent on the message type.

  Field 41 Options A and D
  The last line of the field is checked for a valid code word.

Field 49
The field is checked for a valid code word.

Field 61
When the sixth subfield (Code for type of transaction), has the option S for a SWIFT message, then the message type is checked against the message type table.

Field 68 Options B and C
Checks that currencies in the second data area are the same. Checks code words for option C.

Field 83 Option R
The field is checked for a valid code word.

Field 189
Checks ALL and numbers not both used.

DSLMDnnn—Setting a Default for a Message Field

This Message Format Service exit is called by the macro DSLMFS where the following parameters are specified:

```
[name]   DSLMFS TYPE=DEFAULT,                                  *
              MF=....,                                         *
              MODNUM=nnn,                                      *
              OUTBUF=...,                                      *
              FLD=....                                         *
             [,TOF=....]
```

Default values are provided for a field in the TOF using the user-written exit program DSLMDnnn, where nnn is specified by the parameter MODNUM. For the internal use of this Message Format Service request, refer also to the description of:

- DEFAULT=nnn parameter of the DSLLFLD macro (FDT)
- DEFAULT=nnn parameter of the DSLLMFLD macro of DSLLDEV TYPE=MESSAGE.

This class of programs is called under the following circumstances:

- During message initialization (DSLMTIN).
- Implicitly by a DSLTSV TYPE=READ macro, when the affected field is empty and the first data area is accessed.
- Implicitly by a DSLTSV TYPE=READ macro, when the affected field is empty and the first subfield is accessed.
- Implicitly by a DSLTSV TYPE=WRITE macro, when the affected field is not yet in the TOF.

Note: Data returned to the TOF supervisor by the default-setting exit, however, can be overwritten by the subsequent processing of the WRITE request.

For the special conventions to be considered for this type of Message Format Service exit, refer to “Coding MFS Exit Programs” on page 75.

The following standard default setting routines are provided by MERVA ESA:

```
                                                    MODNUM
YYMMDD    for dates in the form of year month day   (nnn=904)
YYDDD     for dates in the form of year day         (nnn=905)
MMDD      for dates in the form of month day        (nnn=906)
DD        for dates in the form of day              (nnn=907)
HHMM      for times in the form of hours minutes    (nnn=908)
```

```
HH        for times in the form of hours                  (nnn=909)
MM        for times in the form of minutes                (nnn=910)
YYYYMMDD  for dates in the form of 4-digit year month day (nnn=914)
YYYYDDD   for dates in the form of 4-digit year day       (nnn=915)
```

DSLMEnnn—Editing Program

This Message Format Service exit is called by the macro DSLMFS where the following parameters are specified:

```
[name]   DSLMFS TYPE=EDIT,                                    *
               MF=....,                                       *
               MODNUM=nnn,                                    *
               OUTBUF=...,                                    *
               INBUF=....                                     *
              [,OPT=DEEDIT]                                   *
              [,TOF=....]
```

A data component is transformed from an internal data format to an external data format or vice versa (OPT=DEEDIT), using the user-written exit program DSLMEnnn, where nnn is specified by the parameter MODNUM. For the internal use of this Message Format Service request, refer also to the description of:

- DSLTSV function with types WRITE, READ, or ADDDA together with function modifiers (FMODIF) EDIT, DEEDIT, DECHECK, or EDITCHK
- EDIT=nnn parameter of the DSLLFLD macro (FDT)
- EDIT=nnn parameter of the DSLLMFLD macro (MCB).

Note: Editing of data is called in the Message Format Service net, screen and print formatting service routines. For the special conventions to be considered for this type of Message Format Service exit, refer to "Coding MFS Exit Programs" on page 75.

The following editing routines for amounts are supplied with MERVA ESA:

1. nnn=901 for amount editing using points and decimal comma (European format, 1.000.000,00). The mnemonic AMOUNT can be used instead of 901.
2. nnn=902 for amount editing using commas and decimal point (American format, 1,000,000.00).

DSLMXnnn—Expanding Field Contents

This Message Format Service exit is called by the macro DSLMFS where the following parameters are specified (see also "DSLMXPND—Field Expansion of a Complete Message" on page 70):

```
[name]   DSLMFS TYPE=EXPAND,MEDIUM=FIELD,                     *
               MF=....,                                       *
               MODNUM=nnn,                                    *
               INBUF=...,                                     *
               OUTBUF=...,                                    *
               FLD=....                                       *
              [,TOF=....]
```

For the internal use of this Message Format Service request, refer also to the description of:

- DSLTSV TYPE=EXPAND
- EXPAND=nnn parameter of the DSLLFLD macro (FDT)
- EXPAND=nnn parameter of the DSLLMFLD macro (MCB).

A field can be changed according to an user-written exit program DSLMXnnn, where nnn is specified by the parameter MODNUM.

DSLMSnnn—Separating a Subfield from Its Main Field

This Message Format Service exit is called by the macro DSLMFS where the following parameters are specified:

```
[name]   DSLMFS TYPE=SEPR,MEDIUM=DATA,                    *
              MF=....,                                    *
              MODNUM=nnn,                                 *
              INBUF=...,                                  *
              OUTBUF=...,                                 *
              FLD=....                                    *
             [,TOF=....]
```

A subfield is separated from the main field in the TOF using the user-written exit program DSLMSnnn, where nnn is specified by the parameter MODNUM. This type of Message Format Service exit routine is called internally by DSLTOFSV whenever an access to a subfield of a main field is made using an DSLTSV TYPE=READ|WRITE|ADDDA|DELETE macro. The number of the exit program is defined by the parameter SEPR of the DSLLFLD macro.

For special conventions to be considered for this type of Message Format Service exit, refer to "Coding MFS Exit Programs" on page 75.

The following standard separation routines are provided by MERVA ESA:

nnn=901 Separation for fixed structures, subfield is optional, trailing hexadecimal zeros are stripped.

nnn=902 Separation of system fields (see "System Field Separation Routine" on page 88).

nnn=903 Reads the function explanation from DSLFNTT for a given function name.

nnn=904 Reads the message description from DSLMTTT for a given message type.

nnn=905 Separation for fixed structures, subfield is mandatory, trailing hexadecimal zeros are stripped.

nnn=906 Reserves space in the TOF for special subfields.

nnn=907 Converts a SWIFT II message acknowledgment to SWIFT I format.

nnn=908 Separation for fixed structures, subfield is optional, trailing hexadecimal zeros are not stripped.

nnn=909 Separation for fixed structures, subfield is mandatory, trailing hexadecimal zeros are not stripped.

nnn=910 Extracts the SWIFT NAK code and creates an explanatory text for it.

nnn=911 Reads the penultimate data area.

nnn=912 Splits EDIFACT field into data areas of length 75.

nnn=913 System fields used by the diagnosis information panel DSL0NIC; this panel displays the status of the inter- and intraregion communication.

nnn=914 Reads an existing UMR or creates a new UMR for a message when reading fields DSLUMRGT or DSLUMRNW.

nnn=915 Creates a 4-digit year from a 2-digit year input field.

nnn=916 Extracts data areas backward. Read with DAINDEX=1 reads the last data area of the mainfield, read with DAINDEX=2 reads the penultimate data area, and so on.

nnn=999 Default separation routine for fields with prefix $$$. This separation routine can be used to test new subfields, without the need to define the subfield in the field definition table DSLFDTT.

nnn=1001 Separation of SWIFT fields.

nnn=2001 Separation of EDIFACT subfields.

DSLMUnnn—Calling MFS User Exits

A user exit is called by the macro DSLMFS, where the following parameters are specified:

```
DSLMFS  TYPE=USER,MODNUM=nnn
```

Coding MFS Exit Programs

The following types of MFS exits are available:

- DSLMCnnn (checking)
- DSLMDnnn (default setting)
- DSLMEnnn (editing)
- DSLMSnnn (separation)
- DSLMUnnn (user)
- DSLMXnnn (expansion)

MFS exit programs can be coded with the following interfaces:

- High-level language interface
- DSLMMFS macro-level interface.

The interface used depends on the setting for the LANG parameter in the MFS program table (refer to the description of the macro DSLMPT in the *MERVA for ESA Macro Reference*).

Coding MFS Exit Programs with a High-Level Language Interface

MERVA ESA provides an MFS exit interface that allows exit routines to be written in any of the high-level languages supported by the MERVA ESA API: C, COBOL, and PL/I. Your exit routine can invoke MERVA ESA services using API calls. For more information about this interface, refer to the *MERVA for ESA Application Programming Interface Guide*.

Coding MFS Exit Programs with the DSLMMFS Macro-Level Interface

User-written Message Format Service exits can access the following MERVA ESA services:

- MFS services (DSLMFS)
- TOF services (DSLTSV)

- General services (DSLSRV)
- File services (DSLFLV)
- Trace services (DSLTRA).

All other MERVA ESA direct or central services can be accessed as well, but should be chosen with consideration of performance aspects, for example, journaling or queue accesses.

The machine readable material supplied with MERVA ESA contains the following sample MFS exit programs:

- Checking Exit DSLMC899
- Default Setting Exit DSLMD899
- Editing Exit DSLME899
- Separation Exit DSLMS899
- User Exit DSLMU099 (for CICS only).

For MERVA ESA running under MVS, these sample programs are in the library with the low level qualifier ADSLSAM0 in the data-set name.

These samples explain the interface and make the setup for that interface. The User Exit DSLMU099 shows CICS how to use the CICS Exec Interface. User exits can be called both in the batch environment or under control of CICS. Therefore they must check for CICS environment before using CICS commands or macros. The following two instructions check for a CICS environment:

```
TM   MFSLWORK,MFSLCECI   CICS ENVIRONMENT
BZ   MFSGOOD             ....NO,NO CICS CALLS
```

MFS Entry Coding

Building General Entry and Exit Code for MFS Programs

A Message Format Service exit must be started with a DSLMFS macro with the following parameters:

```
extname  DSLMFS MF=START,TYPE=type,MODNUM=nnn,OPT=(optlist)
```

See the *MERVA for ESA Macro Reference* for a complete description of all parameters.

The label *extname* is the external name of the Message Format Service exit. This name is specified in the Message Format Service program table DSLMPTT, together with an exit type and a module number. When defining external program or program entry names, care should be taken to avoid conflicting references during link-editing of the module DSLMMFS. In particular, the use of the same names for MCBs and programs must be avoided.

The keyword specified with the parameter TYPE refers to the type of Message Format Service exit. The following types are supported:

- MFS
- CHECK
- DEFAULT
- EDIT
- EXPAND
- SEPR

- USER.

The number of the program is specified by the parameter MODNUM as a five-digit number from 1 to 32767.

If the label is omitted, a name in the form DSLMfnnn is built, where:

- "f" is determined from the keyword specified for the parameter TYPE.
- "nnn" are the last three digits of the number specified for the parameter MODNUM.

Example:

```
DSLMC001 for TYPE=CHECK,MODNUM=1001
DSLMD904 for TYPE=DEFAULT,MODNUM=904
DSLME901 for TYPE=EDIT,MODNUM=901
DSLMX001 for TYPE=EXPAND,MODNUM=1001
DSLMS901 for TYPE=SEPR,MODNUM=901
DSLMU003 for TYPE=USER,MODNUM=3
```

The DSLMFS MF=START macro results in the generation of:

- DSECTs for MERVA ESA and Message Format Service storage available for the MFS exit. These are:

  DSLCOM The MERVA ESA communication area.

  MFSL The MFS parameter list passed to the MFS exit.

  MFSPS MFS permanent storage containing status information for the MFS session.

  MFSTS The basic MFS temporary working storage containing register save areas; if OPT=EXTTS is specified an MFS parameter list and a parameter list for internal TOF supervisor calls are generated. Message Format Service exit programs that use neither TOF supervisor services nor Message Format Service services can save storage space by omitting this parameter. The temporary storage is allocated dynamically during program linkage by the MFS interface. This storage area can be extended by data areas required by the Message Format Service exit.
- DSECTs of the MFS buffer prefix, MFS module header, and of the Message Type Table entry.
- Register Equates.
- Program entry code that carries out the MFS linkage and housekeeping functions:
  - When OPT=EXICAL is specified entry code for CICS dependent modules is generated. These programs can use the EXEC CICS interface, and they must be processed with the CICS language translator before assembly.
  - Setup of program save areas and save area chaining.
  - Initializing the program base register 10. When OPT=BASE11 is specified register 11 is setup as second base register.
  - Initializing the registers for MERVA ESA and MFS storage DSECTs.
  - When internal TOF supervisor and Message Format Service parameter lists are available these parameter lists are initialized.
  - Initializing the MFS debugging area (MFSTDEB) is done in the interface program DSLMMFS.

Note: The generation of storage DSECTs is suppressed when OPT=NOMAPS is specified. OPT=NOPRINT generates a 'PRINT NOGEN' statement, which results in a smaller assembly listing of the program.

To define additional entry points in a Message Format Service exit program the DSLMFS MF=ENTRY macro must be coded:

```
entname  DSLMFS MF=ENTRY,TYPE=type,MODNUM=nnn
```

The label *entname* is the entry name within the Message Format Service exit. This name is specified in the Message Format Service program table DSLMPTT together with the external name given in the DSLMFS MF=START macro, an exit type, and a module number.

The entry name does not appear in the linkage editor listing for the DSLMMFS module. It is an internal name connected to an exit type and module number via the Message Format Service program table.

Interface Conventions

- MFSLIBUF contains the address of a buffer that contains the input data for the Message Format Service exit depending on the type of exit. The buffer has the usual MERVA ESA buffer and data length fields.
- MFSLOBUF contains the address of a buffer where the output data of the Message Format Service exit depending on the type of exit must be provided. The buffer has the usual MERVA ESA buffer and data length fields.
- MFSLFLD contains the address of the field reference and shows the Message Format Service exit for which field it is called. The field reference has the following layout (use a DSLMFS MF=FLDREF macro to map it):

```
FLDFLDRF DC    0CL16' '           BASIC FIELD REFERENCE
FLDNI    DC    AL1(0)                  FIELD NESTING INDICATOR
FLDFG    DC    AL1(0)                  FIELD GROUP INDEX
FLDRS    DC    Y(0)                    FIELD REPEATABLE SEQ INDEX
FLDNAME  DC    CL8' '                  FIELD NAME
FLDDA    DC    Y(0)                    FIELD DATA AREA INDEX
FLDOPT   DC    CL1'D'                  FIELD OPTION INDICATOR
FLDSTAT  DC    XL1'00'                 FIELD STATUS
FLDSTEX  EQU   X'80'                     1ST EXTENSION EXISTS
FLDSTEM  EQU   X'40'                     DATA AREA EMPTY
FLDINIT  EQU   X'20'                     TOF REQUEST TYPE WAS INIT
FLDCHEK  EQU   X'08'                     TOF CHECKING REQUIRED
FLDSTX2  EQU   X'04'                     2ND EXTENSION EXISTS
FLDEXT   DS    0C         OPTIONAL FIELD REFERENCE EXTENSIONS
*  THE EXTENSIONS MAY BE THERE OR NOT, DEPENDING ON THE BIT-SETTINGS
*  IN BYTE FLDSTAT, BUT IF BOTH EXTENSIONS ARE THERE THEY ARE ALWAYS
*  IN THE FOLLOWING SEQUENCE:
*                           1ST EXT (BIT FLDSTEX)    FOR SUBFIELDS
FLDNAME0 DC    CL8' '              FIELD MASTER NAME
FLDOFF   DC    Y(0)                OFFSET OF SUBFIELD
FLDLEN   DC    Y(0)                LENGTH OF SUBFIELD
FLDLENM  DC    Y(0)                MAXIMUM LENGTH OF SUBFIELD
*
*                          2ND EXT (BIT FLDSTX2)    FOR NESTED RS
FLDRSXNN DS    AL2                 NO OF RS INDICES USED (FOLLOWING)
FLDRSX01 DS    AL2                 OCC INDEX IN FIRST REP SEQ
* VARIABLE NUMBER OF OCC INDEX FIELDS - GENERATED AS SPECIFIED
FLDRSX09 DS    AL2                 OCC INDEX IN NESTED REP SEQS
```

- MFSLMSG contains the address of a message identification. This parameter is used if a message is initialized, if a message is mapped in a predefined format, or if the message type of a message must be determined.

- The possible error status must be set in the reason code field before returning to the caller of the Message Format Service exit routine. An MFS reason code is inserted in the Message Format Service parameter list by the following instruction:

```
MVC   MFSLREAS,=Y(MFSRE...) error reason code
```

  To use the symbolic names for Message Format Service reason codes, include the copy book DSLMREAS in the exit program by specifying OPT=REASON in the DSLMFS MF=START macro.

  The MFS reason codes from 1 to 99 are reserved for TOF supervisor errors. The reason codes from 100 to 499 are used by MERVA ESA Message Format Services. The reason codes from 900 to 999 are general reason codes for severe errors. The reason code range 500 to 899 can be used concurrently by special applications or user programs.

  The SWIFT Link, for example, uses these reason codes. The resulting error messages are built by using the program prefix of the generating MFS exit program; for the SWIFT Link this prefix is "DWS".

  A user program can issue a reason code *xxx*, where *xxx* is a number between 500 and 899. When the name of the user-exit program starts with "DSL", an error message DSL3xxx must be defined in the operator message table to indicate the cause of the error.

  Note: The MFS reason code is only processed if an MFS return code not equal to 0 is also supplied. Use one of the program return points, or supply the return code with register 15 (R15).

- The following program return points are available for returning to the caller of the Message Format Service exit routine:

```
- MFSGOOD      Function successfully completed.           (RC = 0)
- MFSERWNG     Function partially or not completed.       (RC = 4)
- MFSERROR     Function erroneously completed.            (RC = 8)
- MFSERINV     Function call invalid, stop processing.    (RC = 12)
- MFSERINS     Processing environment invalid.            (RC = 16)
- MFSEXIT      Common Return Point when register 15 was already
               set to the appropriate return code. At entry to
               the user code in an Message Format Service exit
               routine, R14 contains the address of the Common
               Return Point MFSEXIT.
```

  The normal return codes to be given by an MFS exit program are 0 and 4. The other return codes should be used carefully, and only if required to terminate the MERVA ESA transaction or application program. At these return points, the MFS return code is set, and the exit returns to the MFS interface.

- The MFS exit can request CICS services using CICS commands.

  This must be indicated by specifying OPT=EXICAL in the DSLMFS MFS=START macro. This bit must be tested before issuing any EXEC CICS calls:

```
TM    MFSLWORK,MFSLCECI
BNO   BATCH
EXEC CICS ...
```

  When an MFS exit is called in a batch environment, CICS services are not available. An example for a program executing in batch environment is DSLSDY. This program uses the MFS Display and Print Services, which in turn calls the MFS edit and checking exits, and user exits: for example, the user exit DSLMU003 that can modify the top and bottom frames.

Usage Conventions for General Purpose Registers

In all MFS programs and exits, some general purpose registers are used for special purposes. This permits general program entry and return; it also allows special actions to be taken when calling another MFS exit, or when requesting a TOF supervisor service.

The following registers are initialized at program entry, and to ensure program integrity should only be used for the purposes described in the following:

The names R0 to R15 are provided for registers 0 to 15 by using EQU statements.

R5 Points to MFS permanent storage.

R6 Points to the temporary working storage for the MFS exit program.

R7 Points to the parameter list of the MFS exit.

R9 Points to the CICS EXEC interface block DFHEIB that is used to transfer the results of a CICS request, when OPT=EXICAL is specified.

R10 Is the first base register.

R11 Is optionally the second program base register. This register can be requested by specifying OPT=BASE11 in the DSLMFS TYPE=START macro.

R12 Points to the MERVA ESA communication area DSLCOM.

R13 Points to the save area of the MFS exit that is used for internal requests for services by MERVA ESA, CICS or IMS, or the operating system. If it has been used for other purposes, it must be reset using the instruction:

```
LA    R13,MFSSAVE
```

At return from user code and entry to the common MFS return point the register 13 *must* point to this internal save area.

Installation of MFS Exit Programs

The macro DSLMPT is used to generate the MFS exit program definition in the MFS program table (DSLMPTT).

This table is exclusively used by the module DSLMMFS to set up:

- The linkage to MFS programs
- User exits
- Field checking exits
- Default setting exits
- Editing exits
- Field component separation exits
- Field expansion exits.

Additionally, table entries for prelinkage of frequently used MCBs and PF key tables can be defined.

To define additional Message Format Service user exits, or entry points in these exit programs, the DSLMPT TYPE=ENTRY macro must be coded:

```
DSLMPT MF=ENTRY,NAME=(extname,entname),NUMBER=nnn,TYPE=c
```

When the external name and entry name are identical, only one name need to be coded.

All programs are identified by module numbers from 1 to 32767 for each class of modules and exits. These numbers are used to build internal names and as reference in the MERVA ESA FDT and MCB Definition macros DSLLFLD, DSLLSUBF, and DSLLMFLD. External program names and entry names in more complex modules can be used and specified in the appropriate entries.
MERVA ESA assigns number ranges for all exit classes to the Base Functions, to all external network links, such as the SWIFT Link, and to user coded Message Format Service exit programs.

Figure 4 shows the ranges of numbers used by the various components. *max* means the maximum number 32767.

	User	MERVA Base	SWIFT Link	EDIFACT	Telex Link	Reserved	National Networks	Reserved
Check	1–899	900 – 999	1000–1999	2000–2999	3000–3999	4000–8999	9000–9999	10000–max
Default	1–899	900 – 999	1000–1999	2000–2999	3000–3999	4000–8999	9000–9999	10000–max
Edit	1–899	900 – 999	1000–1999	2000–2999	3000–3999	4000–8999	9000–9999	10000–max
Expand	1–899	900 – 999	1000–1999	2000–2999	3000–3999	4000–8999	9000–9999	10000–max
Separation	1–899	900 – 999	1000–1999	2000–2999	3000–3999	4000–8999	9000–9999	10000–max
User	1000-1999	1 – 99	100– 199	200– 299	300– 399	400–899	900–999	2000-max

Figure 4. Overview of Program Number Ranges

MERVA ESA provides a facility to install MFS exit programs without modifying the MFS program table and re-linking DSLMMFS. For MERVA ESA running under MVS, you must use SMP/E to re-link DSLMMFS.

This can be a problem in an installation, where an application programming department is responsible for modifying the MFS user exits, but a system programming department is responsible for the installation of programs using SMP/E.

A user-defined exit program is loaded, when the MFS program table DSLMPTT contains an entry for this type class, specifying NUMBER=ALL. For example, the following entry is defined in the MERVA ESA MFS program table copy book DSLMPTTC:

```
DSLMPT NAME=DSLMCXXX,NUMBER=ALL,TYPE=C,LINK=NO
```

This defines that for all calls to a checking exit (TYPE=C), MFS should build the program name of the exit and load it dynamically from the load library (LINK=NO). The name of the exit is built by concatenating the first five characters of the name parameter (here: DSLMC) and the last three digits of the requested exit number.

This is done only if the requested number is not explicitly defined in the MFS program table.

For example, if CHECK=23 is coded in the field definition table DSLFDTT for a field, and the field should be checked, MFS tries to load the module DSLMC023.

Note: For CHECK=1023, for example, this module is also called, because only the last three digits of the exit number are used.

In the MERVA ESA MFS program table, the following entries with NUMBER=ALL are defined:

DSLMCnnn Checking exits

DSLMDnnn Default setting exits

DSLMEnnn Editing exits.

There is no such definition for expansion, separation and user exits in the copy book DSLMPTTC of the MFS program table. This means that these exits are called only, if the requested number is explicitly defined in the MFS program table.

MERVA Link: MFS Program Table Modification

The MERVA Link control MCBs, the sample MERVA Link MFS User Exit, and the sample MERVA Link MFS editing exits for information displayed by the MERVA System Control Facility must be specified in the DSLMPTT. The DSLMPTT entries to be added for MERVA Link are shown below. The copy book EKAMPTTC of the MERVA ESA macro library contains the macro instructions to generate these DSLMPTT entries.

```
*-------------------------------------------------------------*
*        MERVA LINK MESSAGE CONTROL BLOCKS                    *
*-------------------------------------------------------------*
         DSLMPT NAME=EKAMCTL,TYPE=M
         DSLMPT NAME=EKAACMM,TYPE=M
         DSLMPT NAME=EKAAC00,TYPE=M
         DSLMPT NAME=EKAAC01,TYPE=M
         DSLMPT NAME=EKAAC02,TYPE=M
         DSLMPT NAME=EKAAC03,TYPE=M
         DSLMPT NAME=EKAAC04,TYPE=M
*-------------------------------------------------------------*
*        MERVA LINK SAMPLE MFS USER EXIT                      *
*-------------------------------------------------------------*
         DSLMPT NAME=EKAMU010,NUMBER=7010,TYPE=U
*-------------------------------------------------------------*
*        MERVA LINK MFS EDITING EXITS                         *
*-------------------------------------------------------------*
         DSLMPT NAME=EKAME010,NUMBER=7010,TYPE=E
         DSLMPT NAME=EKAME011,NUMBER=7011,TYPE=E
         DSLMPT NAME=EKAME012,NUMBER=7012,TYPE=E
         DSLMPT NAME=EKAME015,NUMBER=7015,TYPE=E
```

MERVA Link assumes that you do not yet have MFS exits with numbers 7010 to 7015. If you do, you must modify the MERVA Link sample number in the applicable DSLMPTT entry, and in the sample partner table or the following MCBs:

- EKAAC00
- EKAAC01
- EKAAC02
- EKAAC03
- EKAAC04
- EKADEMO
- EKAMCTL

The editing exit number is specified in the screen definition of these MCBs. The user exit number is specified in the MFSEXIT parameter of the EKAPT TYPE=ASP macro of the sample partner table.

How to Process the Changed DSLMPTT

1. Edit the appropriate copy book of DSLMPTT and insert the definition for the new user exit.
2. Assemble DSLMPTT.
3. Link-edit DSLMMFS.

Step 2 and 3 are part of the MERVA ESA system generation. See the DSLGEN macro in the *MERVA for ESA Installation Guide*.

MFS Exit Program Classes

The purpose of and specific considerations for each type of exit are described in the following.

MFS Checking Exits (DSLMCnnn)

Checking Concept

A checking routine must be invoked when a field or subfield is accessed that is specified with the parameter CHECK in the Field Definition Table or in the MCB definition. For subfields only, checking can be carried out in the separation routine if so specified. The checking routine checks the contents of a field and returns the result of the checking operation in the form of an MFS reason code. This reason code is processed further by the MFS programs. For example, an error or information message is generated, or, for screen processing, the field in error is highlighted. The error message is displayed on the screen or printer. MERVA ESA application programs include the reason code in character format (length 4) in the subfield MSGTRERR of the message trace field MSGTRACE, where it can be inspected during message routing.

In MERVA ESA there are different types of checking:

Data Checking
A specific component, data area, subfield, or option of a message field is checked.

Field Checking
All components, data areas, and options of a message field are checked.

Character Set Checking
This is a special form of data checking. Input data is checked by a user-written checking program or by a MERVA ESA-provided character set checking routine. MERVA ESA supports the following character sets:

ALPHA	Alphabetic character set
NUMERIC	Numeric character set
ANUM	Alphanumeric character set
HEX	Hexadecimal character set
EDIFACTA	UN EDI character set.

Message Checking

The MERVA ESA message checking program checks the contents of the TOF field-by-field. If there are errors, a list of error messages is produced for display on the screen or printer.

A message-type-specific user exit and the checking user exit 9 is called at the end of message checking to change the checking results or to do additional message checking as required by the user.

Page Checking

Page checking is carried out by the screen and printer page formatting program (DSLMPBLD).

During creation of a page following a scroll command containing the parameter CHECK, each data component presented on the medium is checked. The component is checked to determine whether it is mandatory, and whether the contents of the data component are incorrect. For a missing mandatory field, a question mark is presented on the screen. If an error is found or mandatory fields are missing, the field on the screen is highlighted, and the cursor is positioned on it. The page is presented again to the user, accompanied by an error message for the first field in error.

The indication of missing mandatory fields is suppressed when the page is presented the first time for a data-entry function.

Special Conventions for Checking Routines

The standard field reference is passed in the parameter list referenced by MFSLFLD; see "Interface Conventions" on page 78. The basic field reference consists of the field name, the field qualifiers, an option modifier, and a field status. The field name can be a subfield name; in this case the field reference is extended by the subfield features including the main field name.

The field status is used to pass information from the TOF Supervisor DSLTOFSV to the checking routine and vice versa. The following status bits should be considered by the checking routine:

- FLDSTEX. This bit is set by DSLTOFSV to indicate that the field reference is extended (subfield).
- FLDSTEM. This bit is set by DSLTOFSV to indicate that the data area to be checked was not in the TOF. In addition the buffer referenced by MFSLIBUF contains no data.
- FLDCHEK. This bit is set by the checking routine to indicate to DSLTOFSV that standard checking of the field characteristics is required.

The checking routine is called either for field checking or data area checking; field checking is indicated by the data area index FLDDA = 0. In the case of data area checking, the specific data area index is supplied, and the input buffer contains the data of the component.

If data area checking is required for page checking, there are two ways for a checking routine to indicate that a question mark is to be presented on a screen:

1. Using the information passed in FLDSTEM (the component is empty), the checking routine supplies a reason code in the range 101 - 199. Then DSLTOFSV provides the question mark.
2. The checking routine found the input buffer to be empty and sets the bit FLDCHEK. Then DSLTOFSV provides the question mark for the first data area if the corresponding field is mandatory according to the specification in the Field Definition Table (FDT) or message control block (MCB).

MERVA ESA supplies a sample checking exit DSLMC899 in the machine readable material.

MFS Default Setting Exits (DSLMDnnn)

Default Setting Concept

A default setting routine is used to initialize a field automatically when a new message is generated. For example the address of the sender or the current date can be filled by a user- or installation-specific default setting routine. This data does not have to be entered manually by a user. The default setting routine is invoked when the DEFAULT parameter for a field is specified in the Field Definition Table or Message Control Block and you use one of the following functions:

- DSLMFS TYPE=INIT,MEDIUM=MSG for initialization of all fields of a message. A subfield is never initialized; therefore a default setting routine defined on a subfield is not called during message initialization. When subfield data must be initialized, this must be done on a field level default setting routine that can set a default data area containing all its subfields.
- DSLTSV TYPE=INIT,FDNAM=.... for initialization of one field.
- Implicitly when using DSLTSV TYPE=READ,FDNAM=....,DAINDEX=1, and when the field is empty. The FDNAM parameter specifies a main field or a subfield. The default setting routine is called for only the first empty subfield of a field.
- Implicitly when using DSLTSV TYPE=WRITE,FDNAM=.... and the field is not yet in the TOF. In this case, the default data is overwritten by the subsequent write.

Special Conventions for Default Setting Routines

When a default setting routine gets control, the field MFSLOBUF contains the address of a buffer. A default setting routine provides the data for the TOF field in either of the following ways:

- Move the default data to the output buffer pointed to by the address in MFSLOBUF, and use the exit point MFSGOOD for returning to the calling program. The TOF supervisor program then writes the data to the TOF field. The data is ignored when the routine was called from an implicit DSLTSV TYPE=WRITE request.
- Write the default data directly to the TOF field using a DSLTSV TYPE=WRITE macro. Set the reason code MFSRENM5 (105) when called for a field initialization, set the reason code MFSRENM7 (107) when called for an implicit field read or write. Use the exit point MFSERWNG (return code 4) for returning to the calling program. In this case the TOF supervisor program does not write data from the output buffer to the TOF field. Using this technique, the default setting routine can also provide data for other TOF fields or other data areas but the first one.
- Indication of the type of default setting required.
  - Option supplied in the MFS parameter list. For an implicit call from a DSLTSV TYPE=READ request the bit MFSLO2RD; for an implicit call from a DSLTSV TYPE=WRITE request the bit MFSLO2WR; and for an explicit call from a DSLTSV TYPE=INIT no bit is set.
  - Field status bit FLDINIT in the field reference. This bit is set by the TOF Supervisor DSLTOFSV to indicate to the default setting routine that it is called from a DSLTSV TYPE=INIT request.

MERVA ESA supplies a sample default setting exit DSLMD899 in the machine readable material.

MFS Editing Exits (DSLMEnnn)

Editing Concept

These programs enable the formatting of data for a network line, and the display of data on a screen or printer terminal, in a form different from the one contained in the TOF. This function can be called as follows:

- Using DSLTSV with one of the types READ, WRITE, or ADDDA together with one of the field modifiers (FMODIF) EDIT, DEEDIT, DECHECK, or EDITCHK.
- DSLMFS TYPE=EDIT,MF=(E,...)

An edit routine can be used to modify the screen attributes of a field on the screen. The current attributes of the field are stored in field MFSPUCOM when the edit routine is called. These attributes can be changed by the program. The sample edit routine DSLME899 in the sample library shows a coding example.

Special Conventions for Editing Routines

- MFSLIBUF points to the data to be translated (ML00LL00 data). ML contains the total buffer length and LL contains the actual data length + 4.
- MFSLOBUF points to the output buffer for translated data (ML00LL00 data). LL must be set to the actual data length + 4 after translation, but must not exceed ML - 4.
- The exit program must provide a working buffer in its temporary working storage (MFSTS) for intermediate results. This is necessary for the possible call coded as MFSLIBUF=MFSLOBUF.
- MFSLO2DE shows that de-editing must be done. For example, translating user input data into internal TOF format.
- MFSLO1RT shows that a retype-verification check must be done. This special function is necessary to allow a retype verification for fields that are edited. Normally the MERVA ESA screen services compare the fields to be checked literally. When editing is specified for a field, the internal representation of the data is different from the form the end-user sees.

  When the edit routine is called with option 'RETYPE' (MFSLO1RT), it may perform its own compare operation. For example, the edit routine may decide that the two amount fields '1000,' and '1.000,00' are equal, even when they are not physically equal. The edit routine is called by the MERVA ESA MFS screen services; a special reason code indicates that the edit routine expects the retype check to be performed by the MFS screen services. All edit routines that handle retype verification fields must support the RETYPE option, otherwise unpredictable results can occur (see coding example below).

  Refer to DSLBM05 in “Appendix D. MERVA ESA Sample Programs” on page 161 for an example of an MFS retype edit exit.

Coding Example

The following code for retype checking should be added to all user edit exits that work on fields used also for retype editing. The calling interface for edit with option RETYPE is as follows:

Input MFSLO1RT for OPTION=RETYPE is set. MFSLIBUF contains original field data from message. MFSLOBUF contains retyped input data from screen.

Output MFS return and reason code:

0 / 0 Retype check OK (fields are equal)

4 / MFSRERCH
Retype check failed (fields are not equal)

4 / MFSRENM1
Retype check not performed (check should be done by MFS screen services).

```
EDITEXIT DSLMFS MF=START,TYPE=EDIT,MODNUM=...
         TM    MFSOPT1,MFSLO1RT        retype check requested?
         BO    RETYPE                  ..yes, data not to be edited
         ...                           do edit/de-edit for data here
         B     MFSGOOD
RETYPE   DS    OH                      code for retype checking
         L     R8,MFSLIBUF             get buffer with original data
         L     R9,MFSLIBUF             get buffer with retyped data
         ...                           do retype checking here
         ...
         MVC   MFSLREAS,=Y(0)          retyped data accepted
         B     MFSGOOD                 return code is 0
         ...
         MVC   MFSLREAS,=Y(MFSRERCH)   retyped data rejected
         B     MFSERWNG                return code is 4
         ...
         MVC   MFSLREAS,=Y(MFSRENM1)   let MFS do the data compare
         B     MFSERWNG                return code is 4
```

MERVA ESA supplies a sample editing exit with the name DSLME899.

MFS Separation Exits (DSLMSnnn)

Separation Concept

A separation routine must be invoked when a subfield or system field is accessed that has specified the parameter SEPR in the Field Definition Table.

The separation routines are invoked to separate the data for a subfield from a field's data area; or, in the case of write, to insert data of a subfield into the appropriate position of a field's data area; or, in the case of delete, to remove the subfield data from a field or a field's data area. Therefore, separation can be understood as special field editing, which is different from screen (presentation medium) editing.

MERVA ESA provides means for defining structures; that is, a field can be separated into different sets of subfields. In MERVA ESA this is completely under control of the separation routine. The separation routine decides whether a requested subfield belongs to the current structure or not.

Subfields are processed by the following programs:

- Screen, printer, and line formatting programs (according to MCB definition)
- Routing Scanner (according to routing table definition)
- MFS checking, editing, default setting or separation exits
- Queue management (key definition in function table)
- External Network Programs.

Separation is transparent to an application. For example, the application has no need to know whether it accesses a subfield or a main field. The transformation into a separation read/write/delete request is done by the TOF supervisor.

General Separation Routine for Optional Subfields: A general separation module is provided by MERVA ESA. This routine can separate fields with a fixed subfield layout, so that, a subfield has always the same offset and length within the main field. In this case, subfields can be omitted at the end of the main field. The name of this routine is DSLMS901 and it can be used for optional subfields. This routine cannot set the mandatory indicator. The definition for this routine in the Field Definition Table entry is SEPR=STANDARD.

General Separation Routine for Mandatory Subfields: A general separation module is provided by MERVA ESA. This routine can separate fields with a fixed subfield layout, that is, a subfield always has the same offset and length within the main field. In this case, subfields can be omitted at the end of the main field. The name of this routine is DSLMS905 and it must be used for mandatory subfields as it sets the mandatory indicator in screen panels. The definition for this routine in the Field Definition Table entry is SEPR=905.

System Field Separation Routine: System fields contain user session dependent information, which is not in the TOF. This information is normally stored in a control block of the MERVA ESA application. The fields can be displayed on screen or printer, or mapped into a line buffer, by specifying the field's name in a device description of an MCB, and can also control the conditional processing of MCBs.

The extraction process is carried out via the environment definition. When a subfield is specified in the Field Definition Table entry with SEPR=SYSTEM, then the system field separation routine DSLMS902 is called.

System Field Table: The following fields are supported by the system field separation routine DSLMS902:

Table 1. System Field Table

Name	Function	Origin
DSLACCD	Current data area index	TUCB
DSLACCDA	Data area index field	FLDREF
DSLACCFG	Field group index field	FLDREF
DSLACCN	Current nesting identifier	TUCB
DSLACCNI	Access nesting identifier	FLDREF
DSLACCO	Current rep. seq. occurrence index	TUCB
DSLACCRS	Occurrence number field	FLDREF
DSLACCRn	Occurrence on nested rep. seq. n; n between 1 and 9	TUCB
DSLACTA	Active occurrence indicator	TUCB
DSLACTD	Actual data area *nnn*	TUCB
DSLACTF	Actual field name *cccccccc*	TUCB
DSLACTG	Actual group number *nnn*	TUCB
DSLACTI	Actual option indication D/O	TUCB
DSLACTL	Actual line number	TUCB
DSLACTM	Current message identifier	TUCB
DSLACTN	Actual nesting identifier	TUCB
DSLACTO	Actual occurrence number *nnn*	TUCB
DSLACTP	Actual page number	TUCB
DSLACTR	Current repeatable sequence index	TUCB
DSLACTS	Actual scroll mode	TUCB
DSLAID	PF key value for AID=DA	PF Key Set
DSLBLANK	Blank filler	
DSLCOLN	Number of presentation columns	TUCB

Table 1. System Field Table (continued)

Name	Function	Origin
DSLCOMP	Actual Compression	TUCB
DSLCOMPL	Line Compression YES/NO	TUCB
DSLCOMPU	Unit Compression YES/NO	TUCB
DSLCOND	Nesting identifier information	
DSLCOPR	MERVA ESA copyright information line	
DSLCOPYQ	Copy queue name	TUCB
DSLCOVID	Cover MCB name from function table	TUCB
DSLCURS	Actual cursor position	TUCB
DSLDATE	Date in format YYMMMDD	DSLSRVP
DSLDATE0	Date in format YY/MM/DD	DSLSRVP
DSLDATE1	Date in format YYMMDD	DSLSRVP
DSLDATE2	Date in format YYYYMMDD	DSLSRVP
DSLDATE3	Date in format YYYYDDD	DSLSRVP
DSLDATE4	Date in format YYYYMMMDD	DSLSRVP
DSLDATE5	Date in format YYYY/MM/DD	DSLSRVP
DSLDST	Originid	TUCB
DSLEFAUT	**aut** command is allowed	TUCB
DSLEFCPC	Copy queue is defined (YES/NO)	TUCB
DSLEFDEL	**delete** command allowed	TUCB
DSLEFEXP	Expansion is active	TUCB
DSLEFFQU	Function with a queue	TUCB
DSLEFHCO	Hardcopy print queue available	TUCB
DSLEFKEY	KEY1,KEY2,KEY1+KEY2 allowed	TUCB
DSLEFK1C	**get key1** command allowed	TUCB
DSLEFK2C	**get key2** command allowed	TUCB
DSLEFMTG	Message type generation allowed	TUCB
DSLEFNOP	NOPROMPT is allowed	TUCB
DSLEFNPD	NOPROMPT display allowed only	TUCB
DSLEFOKC	**ok** command allowed	TUCB
DSLEFPRT	Function is protected	TUCB
DSLEFREK	RETYPE function	TUCB
DSLEFROU	**route** command allowed	TUCB
DSLENV	MFS environment information	TUCB
DSLEXAFO	EXAFO=YES/NO specified (Automatic Force)	DSLPRM
DSLEXJRN	Journal display allowed	DSLPRM
DSLEXQUE	EXQUE=YES/NO specified (Queue Test commands)	DSLPRM
DSLEXSEC	EXSEC=YES/NO specified (Password Check bypassed)	DSLPRM
DSLEXUID	EXUID=YES/NO specified (Sign-on bypassed)	DSLPRM
DSLEXUSR	EXUSR=YES/NO specified (Origin ID Check in USR)	DSLPRM
DSLFPGM	Name of Function Program	TUCB
DSLFRMID	Line format identification for NOPROMPT	TUCB
DSLFUAPL	User Application Field in FNT	TUCB
DSLFUN	Current Function	TUCB
DSLFUNS	Allowed functions	User record
DSLHCOF	Associated hard copy function	TUCB
DSLHEX00	X'00' filler	
DSLHOOK	Field for cursor select	
DSLID	MERVA ESA identification	DSLPRM
DSLKFDL1	Name of key field 1 in FNT.	TUCB
DSLKFDL2	Name of key field 2 in FNT.	TUCB
DSLLANID	Language identifier for user	TUCB
DSLLFMT	Line format used last	TUCB

Table 1. System Field Table (continued)

Name	Function	Origin
DSLLINE	Top Line in message	TUCB
DSLLTERM	Logical Terminal Name	TUCB
DSLMIDB	Message id for bottom frame	TUCB
DSLMIDM	Message id for msg area	TUCB
DSLMIDT	Message id for top frame	TUCB
DSLMORE	Last/more indicator	TUCB
DSLMSTAT	Message status MSG/ FUN	TUCB
DSLNAME	UMR MERVA ESA identification	DSLPRM
DSLNUMR	UMR status NO/YES/IMM	DSLPRM
DSLNXTF	Next function	TUCB
DSLOPID	Master Operator ID	DSLPRM
DSLOPMAS	Master Operator YES/NO	DSLPRM / User record
DSLOTRMF	Online Trace for MFS ON/OFF	TUCB
DSLOTRPG	Processing Trace INT/EXT	TUCB
DSLOTRQE	Queue Trace OFF/SML/LRG	TUCB
DSLOTRRT	Routing Trace OFF/ALL/WNG/SEV	TUCB
DSLOTRTO	Online Trace for TOF ON/OFF	TUCB
DSLPFKEX	Previous PF key definition for *nn*	PF Key Set
DSLPFKEY	Current PF key definition for *nn*	PF Key Set
DSLPFKL	Current PF key information line	PF Key Set
DSLPFKLX	Previous PF key information line	PF Key Set
DSLPFKY	PF Key Table Name in FNT	TUCB
DSLPFSET	Name of Current PF key Set	TUCB
DSLPWC	Password during sign-on	TUCB
DSLPWN1	New password during sign-on	
DSLPWN2	Check new passw. at sign-on	
DSLRECON	RECON=YES/NO specified (traffic reconciliation)	DSLPRM
DSLROUTN	Routing Module Name in FNT	TUCB
DSLROWN	Number of presentation rows	TUCB
DSLTERMT	Terminal type	TUCB/Term table
DSLTHDS2	Header display (MODE in FNT)	TUCB
DSLTIME	Time in format HH:MM:SS	DSLSRVP
DSLTIME1	Time in format HHMMSS	DSLSRVP
DSLTRACE	Trace info in DSLCOM	
DSLUNAME	User Name	User record
DSLUSER	User identification	TUCB
DSLUSFPW	USFPW=YES/NO specified (Prompt for Password in USR)	DSLPRM
DSLUSGRP	USGRP=YES/NO specified (Categorize users into groups)	DSLPRM
DSLUTYP	User/operator type	User Record
DSLWNDOW	Frame or window ident.	TUCB

Function Name Separation Routine: The separation routine DSLMS903 generates an explanatory text for a queue or function name. This text is displayed on the user function selection panel. For example, 'USR' is translated to “User-File Maintenance.”

Name	Function	Origin
DSLFEXP	Explanation of functions	Function Table DSLFNTT

Message Identification Separation Routine: The separation routine DSLMS904 generates an explanatory text for a message identification. This text is displayed on the top of each message panel. For example 'S100' is translated to “Customer Transfer.”

Name	Function	Origin
DSLMIDN	Explanation of message types	Message Type Table DSLMTTT

Reserve TOF Space Separation Routine: The separation routine DSLMS906 reserves space in the TOF of the maximum size of the subfield as defined with the LENGTH operand in the FDT. This space is reserved with the first DSLTSV TYPE=WRITE request for this field.

Note: Once this space is reserved the subfield can be written even when the TOF is full. You should be careful in applying this separation routine as the TOF space available for other fields is reduced.

MERVA ESA supplies the sample separation routine DSLMS899 in the machine readable material.

Special Conventions for Separation Routines

- MFSLO2RD indicates that separation for a READ of a subfield must be done.
- MFSLO2WR indicates that separation for a WRITE or ADDDA of a subfield must be done.
- MFSLO2DL indicates that separation for a DELETE of a subfield must be done.

The input and output buffer contents depend on the request type (option). The supported request types are read, write, add, and delete a subfield. Buffer requirements for write and add are similar.

	Read		Delete		Write	
	Entry	Exit	Entry	Exit	Entry	Exit
MFSLIBUF → Input	field data	-	-	-	subfield data	-
MFSLOBUF → Output	field data	subfield data	field data	modified field	field data	modified field

Figure 5. Description of the Buffer Contents

MFS Expansion Exits (DSLMXnnn)

Expansion Concept

An expansion routine is invoked by the TOF supervisor program, when a field expand request is issued, and the field is specified with the parameter EXPAND=nnn in the Field Definition Table or in the MCB definition. The expansion routine evaluates the contents of the field and decides whether additional data must be written into TOF. This type of Message Format Service exit is used to expand a code name, such as a SWIFT address, into a correspondent name. The information might be displayed on screen and printer devices, or used during routing. MERVA ESA provides for expansion of all message fields in the end-user driver DSLEUD and in the checking and expansion transaction DSLCXT depending on the specification of the EXPAND and EXPNAM parameters of the DSLFNT macro.

Special Conventions for Expansion Routines

The standard field reference, as passed in the parameter list, contains the field name and the field qualifiers of the field to be expanded. The field name is always a main field name, never a subfield name.

The exit must evaluate the field to be expanded by reading the data area or areas. For example, the expansion operation for the SWIFT Link address expansion consists of an access to a file using MERVA ESA file services (DSLFLV) and of writing the data obtained from there into TOF. Writing to the TOF must be done by the expansion exit routine.

Adding a User Exit to DSLMMFS

The MFS user exits called by MERVA ESA allow for additional processing steps at specific points during the processing of a message.

Additional MFS user exits can be specified in the MERVA ESA user-written application programs.

Sample user exits are provided with the distributed material. The interface for MFS user exits in MERVA ESA provides the message in TOF format and the terminal user control block (TUCB); also a user exit communication field is provided by MFS. The TOF contains all message dependent information that can be accessed or changed or both by any of the MFS user exits. The TUCB contains information about the status of the message in process, the message processing function, and the general user information. All this information can be used, but only some of the information in the TUCB can be changed by the user exits. This information can be obtained by a DSLTSV TYPE=READ,FDNAM=DSLccccc and changed by a DSLTSV TYPE=WRITE,FDNAM=DSLccccc request. For a summary of the system fields DSLccccc available see “System Field Table” on page 88.

All user exits (except for DSLMU054) reside in the MERVA ESA load library and are dynamically loaded when the user exit is called by a program.

MERVA ESA provides the following MFS user exits available for modifications:

DSLMU001

Message Initialization User Exit.

This exit is called by DSLMTIN. The TOF for a new message is initialized. Additional defaults can be set into message fields in the TOF, or additional fields may be initialized using the TOF Supervisor request TYPE=INIT.

Note: This user exit is also called for function panels with a DSLLDEV TYPE=MSG definition in the MCB, for example, USR, AUT.

DSLMU003

Panel Frame User Exit.

The purpose of the MFS User Exit DSLMU003 is to change the TOF using the TOF supervisor program. This influences the page build-up. Its conditions are described in the MCBs.

The MFS parameter list contains pointers to the external areas:

- TOF
- MFS permanent storage
- MFS temporary storage

This exit is called twice:

1. Before the page build-up process is started, this cycle is indicated by the field MFSPUCOM containing 0. The size of the windows on the top and bottom of the panel can be determined by this user exit.
2. After the page build-up process has been completed, this cycle is indicated by the field MFSPUCOM containing 4. Now the data to be displayed on the top and bottom window can be changed if they are dependent from the contents of the message window. Data in the message window cannot be changed by the user exit at this time.

DSLMU004

Command Modification User Exit. This exit is called by DSLEFUN.

A command received via the command line is read from the TOF into the command buffer. While reading the command from the TOF, it is erased from the TOF. The input buffer parameter points to the command buffer. The contents of this buffer can be changed by the user exit, or the command execution can be suppressed by passing return code 4 and reason code MFSRECSU (113) to the caller of the exit.

The format of the command buffer is:

```
0    4    8        Offset in buffer
TL00|AL00|DATA     TL=total length
                   AL=actual data length + 4
```

Refer to DSLBM02 and DSLBM04 in "Appendix D. MERVA ESA Sample Programs" on page 161 for examples of MFS user exit 4.

DSLMU005

Changing command parameters.

This exit is called by DSLEFUN. The command input is contained in the parser buffer.

Access to the parser buffer:

- The field COMTUCBA contains the address of the control block TUCB
- The field TUCBCMPA in TUCB contains the address of the parser buffer
- The DSECT of the parser buffer can be generated with the macro DSLNPA MF=L.

The parser buffer contains the command code and the field NPACTIND contains the command table index:

1. Screen Command
2. Session Command
3. Function Command.

The contents of this buffer (command parameters) can be changed by the user exit, or the command execution can be suppressed by passing return code 4 and reason code MFSRECSU (113) to the caller of the exit.

DSLMU006

User Exit for Help Functions.

The help function is MCB driven and can be extended via user-written MCBs. This exit is called by DSLMPCMD after a **help** command was received.

Interface: The MFS parameter list field MFSLMSG points to an 8-byte area that contains an MCB name or a message identification. This panel is displayed as help information. The user can supply additional HELP information through

the user exit by writing this information into the TOF, or change the help panel message identification by changing the 8-byte message identification area.

DSLMU008
Change message data.

The input data has been transferred from screen to TOF and the fields residing on the current page have been checked.

DSLMU008 is called on termination of an input page (PROMPT mode).

Interface: Checking return and reason codes are passed to DSLMU008 in the MFS permanent storage field MFSPUCOM. MFSPUCOM is a 4-byte field, the first 2 bytes contain the return code, and the second 2 bytes contain the reason code.

DSLMU009
Additional message checking.

The complete message has been checked. DSLMCHE is called, for example, by DSLEMSG, if a message is completed by a user command or by EOM; or by DSLMLFP when called with option CHECK (for example, on each ENTER in NOPROMPT processing).

Interface: Checking return and reason codes are passed to DSLMU009 in the MFS permanent storage field MFSPUCOM. MFSPUCOM is a 4-byte field, the first 2-bytes contain the return code, and the second 2-bytes contain the reason code.

When the exit wants to indicate a checking error, the first two bytes of MFSPUCOM should be set to four, as the return code and the second two bytes of MFSPUCOM must be set to MFSREEM0 (checking error found). The error message is be written to the TOF field DSLMSG.

Refer to DSLBM01 in “Appendix D. MERVA ESA Sample Programs” on page 161 for an example of MFS user exit 9.

DSLMU010
Exit for message completion.

This exit is called by DSLEMSG. A message has been completed by a user command. The command could be a session command that implies end of message, that is, return to function selection (RET) or sign-off (SOF). Message processing commands that cause the calling of the user exit are EOM (end-of-message), ROUTE, REQUEUE, DELETE, OK, and HARDCOPY.

The message is in the TOF (but not yet in the queue) and, except for DELETE and HARDCOPY, completely checked. In DSLMU010, additional processing steps, such as accounting and journaling can be performed.

Interface: The TOF address can be obtained from the MFS parameter list; the TUCB address can be obtained from the MERVA ESA communication area DSLCOM.

The command that started the exit DSLMU010 is contained in the parser buffer (see DSLMU005).

The field NPACTINT contains one of the following command table indices:

01 Stands for screen commands.

02 Stands for session commands. For example, **sof**, **return**, **hco**.

03 Stands for function program commands. For example, **eom**, **route**, **requeue**, **delete**, **ok**.

Note: The command code (field NPATOCM) with the command table index gives the exact definition of the command. The command code in all three command tables need not be unique, it is unique just within one table.

DSLMU011
Skip Message exit.

This exit is called by DSLEMSG after a message has been read from a queue. The exit can set the MFS reason code MFSRESKM if the actual message must be skipped (it is not presented on the screen), if the user has no allowance to process that message. The next message is then automatically read from the queue.

Refer to DSLBM03 in "Appendix D. MERVA ESA Sample Programs" on page 161 for an example of MFS user exit 11.

DSLMU020
Exit in DSLSDI.

The exit is called by DSLSDI after a message has been formatted into the TOF. The message may be modified before it is stored into the queue data set. The user exit may decide to skip the message indicated by return code 4 and MFS reason code MFSRESKM.

Before calling this exit, field MFSPUCOM is loaded with the actual function table entry address.

DSLMU021
Exit in DSLSDY.

The exit is called by DSLSDY after a message has been moved into the TOF. The message may be modified, before it is printed. The user exit may decide to skip the message indicated by return code 4 and MFS reason code MFSRESKM.

DSLMU022
Exit in DSLSDO.

The exit is called by DSLSDO when a message is in the TOF before it has been formatted and written to the sequential data set. The message may be modified, before it is written to the output data set. The user exit may decide to skip the message indicated by return code 4 and one of the MFS reason codes MFSRESKP, MFSRESKR, or MFSRESKM. MFSRESKP does not purge the message (it is written to the end of the queue); MFSRESKM purges the message, and MFSRESKR routes the message.

Before calling this exit, field MFSPUCOM is loaded with the actual function table entry address.

DSLMU023
Exit in DSLCXT.

The exit is called by DSLCXT after a message has been moved into the TOF. The message may be modified, before it is checked or expanded. The user exit may decide to skip the message indicated by return code 4 and MFS reason codes MFSRESKM or MFSRESKP. MFSRESKM skips and purges the message; MFSRESKP skips the message and leaves it in the queue. A reason code MFSRECSU can be given to stop the processing of DSLCXT. The message in process is freed and stays in the queue.

DSLMU024
Exit in DSLHCP.

The exit is called by DSLHCP after a message has been moved into the TOF. The message may be modified, before it is printed. The user exit may decide to skip the message indicated by return code 4 and MFS reason code MFSRESKM or MFSRESKP. MFSRESKM means that the message is not printed, and purged from the queue. MFSRESKP means that the message is not printed and stays in the queue.

DSLMU027
Exit in DSLHCP.

This exit can manipulate the output device, the output data stream, or both. The exit is called at three different processing stages in DSLHCP message printing. The different stages are indicated in the field MFSPUCOM:

- Call before the first output segment is prepared. Additional initialization of the output device may be performed, for example, sending a FORMFEED to the printer, or modifying the device characteristic. The terminal user control block (TUCB) contains the logical terminal name and the device characteristic.

  For IMS only: the name of the logical terminal can be changed because this exit is called before the IMS CHANGE call. The name of the logical terminal is in the field TUCBLTN. MFSPUCOM contains 0.
- Call when an output segment for a message is prepared. The output segment may be modified or routed to an additional destination, for example a spooling system. The field MFSLIBUF contains the address of the I/O-segment buffer. This can occur several times in a message. MFSPUCOM contains 4.
- Call when an output segment for a warning panel is prepared. The output segment may be modified or routed to an additional destination, for example a spooling system. The field MFSLIBUF contains the address of the I/O-segment buffer. The printing of the warning panel can be suppressed by issuing MFS reason code MFSRESKP (111) and return code 4. MFSPUCOM contains 8.
- Call after the last output segment has been sent. Additional termination steps on the output device may be performed. MFSPUCOM contains 12.

DSLMU054
Message Type and Network Determination.

This exit is called by DSLMLFP when transforming a message from the net format to the TOF.

The input to this program consists of:

1. The MFS parameter list containing the address of the message buffer, and the address of the message identification contains the network identifier in the first byte.
2. MFSPUCOM in the MFS permanent storage, which contains the address of the current position in the message buffer.

If user-supplied message types and networks are to be supported, the appropriate program selection must be coded in DSLMU054. Otherwise a network-dependent program is selected for the evaluation of the message header. In register 15, a return code is issued:

- 0 means that the header was evaluated. The exit returns a valid 8-byte message identification in the area pointed to by the field MFSLFLD.
- 4 means that the evaluation was not successful. The default message identification 0DSL will be used.

This user exit is linked to DSLMMFS. If DSLMU054 is changed, DSLMMFS must be link-edited.

DSLMU090
MFS termination exit.

This exit is called by DSLMMFS when a termination of MFS is requested. The exit may process any cleanup needed for other user exits which have been called earlier.

DSLMU099
Sample User Exit.

This user exit is not used by MERVA ESA or the SWIFT Link. It shows:

- The setup for a user exit
- How the CICS commands are used in a user exit (CICS only).

DSLMU240
Exit in DSLCESI. DSLCESI is called by DSLCEST.

This exit is called by DSLCESI (EDIFACT to SWIFT conversion). At this point a SWIFT message has been extracted from the EDIFACT message and has been formatted in the TOF. The message may be modified before it is stored in the queue data set. The user exit may decide to skip the message (only if this is the first part of the EDIFACT message) indicated by return code 4 and MFS reason code MFSRESKM.

Before calling this exit, field MFSPUCOM is loaded with the address of the queue name.

DSLMU241
Exit in DSLCSEI. DSLCSEI is called by DSLCSET and DSLSDO.

This exit is called by DSLCSEI (SWIFT to EDIFACT conversion). At this point a SWIFT message has been read from the queue and has been formatted in the TOF. The next step is the concatenation of the message to the EDIFACT message. The message may be modified before it is concatenated. The user exit may decide to skip the message indicated by return code 4 and one of the MFS reason codes MFSRESKP, MFSRESKR, or MFSRESKM.

MFSRESKP does not purge the group of SWIFT messages that is part of the EDIFACT message (the group is written to the end of the queue). MFSRESKM purges the group of SWIFT messages that is part of the EDIFACT message. MFSRESKR routes the group of SWIFT messages that is part of the EDIFACT message.

Before calling this exit, field MFSPUCOM is loaded with the address of the queue name.

DSLMU242
Exit in DSLCES2. DSLCES2 is called by DSLCEST and DSLSDI.

This exit is called by DSLCES2 (EDIFACT to SWIFT conversion). The exit must extract the SWIFT fields from the EDIFACT message (MFSLIBUF) and pass the data back to DSLCES2 (MFSLOBUF).

DWSMU126
Exit in DWSDGPA.

This exit is called by DWSDGPA (SWIFT general purpose application). At this point the message is read from the ready queue and moved into the TOF. In the subsequent steps the message is prepared for sending to the SWIFT network.

This exit can inspect the message in the TOF and modify it. The address of the TOF is contained in the field MFSLTOF. No other information is available in this exit but the TOF.

If this exit decides that the message must be sent, it must give a return code of 0.

If this exit decides to skip a message, it must give a return code of 4 and a reason code, for example, MFSRESKM (112). Then the message DWS689I is written to the MSGACK field in the TOF and routing is invoked. The message DWS689I can be found during routing. It contains the reason code set by this exit. Processing of DWSDGPA continues with the next message in the ready queue.

Chapter 9. Using the Intertask Communication Facility (DSLNIC)

The intertask communication facility is used by application programs that are not linked to the MERVA ESA nucleus DSLNUC.

Through DSLNIC, programs can communicate with the task servers of MERVA ESA to use the central services of MERVA ESA, such as the queue services, operator-command services, user-file services, write-to-operator services, authentication service of the SWIFT Link, and other services defined in the MERVA ESA task-server request table DSLNTRT (see also "Chapter 1. Types of MERVA ESA Application Programs" on page 1).

The communication between the task servers and applications is carried out by the DSLNICT program, which is invoked via the DSLNIC macro. The methods of communication are:

- Interregion
- Intraregion (CICS only)
- Via CICS temporary storage queues (CICS only)
- Via APPC/MVS (MVS only)
- Via MQSeries® for MVS/ESA™

The method of communication is determined by the customization parameter ITC in DSLPRM and by the environment where the requestor is executing.

For transactions running under IMS or batch programs under MVS, the communication is normally via interregion communication. This communication method uses the MERVA ESA SVC DSLNICPM and requires the requestor and the MERVA ESA nucleus to execute on the same MVS image. Alternatively, the MERVA ESA intertask communication via APPC/MVS or MQSeries for MVS/ESA is available. These methods allow you to run the MERVA ESA nucleus and the requestor application on different MVS images. The servers used for the intertask communication are the DSLNTSA (APPC/MVS) and DSLNTSM (MQSeries for MVS/ESA) programs, respectively. These programs under direct control of DSLNUC.

For batch programs under VSE, interregion communication via XPCC is used.

For transactions running under CICS, the communication method is intraregion, using direct buffer transfers in storage to move the data. This method prevents the use of the CICS storage protection facility for MERVA ESA transactions.

If MERVA ESA is not available in the same region or partition, DSLNICT invokes the interregion communication when CINTER=YES is specified in DSLPRM.

It is possible to select the intertask communication via CICS temporary storage queues. This method allows the use of the CICS storage protection facility for MERVA ESA transactions and user programs using MERVA ESA services.

For details of the DSLNIC macro, see the *MERVA for ESA Macro Reference*.

In the following paragraphs, the intertask communication is described as it proceeds for the requesting MERVA ESA application (called the "requestor"). Coding examples show how to use the DSLNIC macro.

Storage Definition

If you set up your own MERVA ESA environment and you want to use the services of DSLNICT, then you must provide the following fields of DSLCOM (see "Filling the Fields of DSLCOM" on page 5 for more information on how to fill these fields):

- COMPRMA
- COMSRVPA
- In a CICS task also provide the fields:
  - COMCWAA
  - COMEIB
  - COMEISTG.

The data-area definitions for a requestor must include the definition of a parameter list for DSLNICT using the macro:

```
NICPLST  DSLNIC MF=L
```

NICPLST is the parameter-list name used in the examples below (it can be any other name, however). The parameter list is required when a DSLNIC MF=E macro instruction is used. It contains data needed during communication with DSLNTS.

All buffers used for intertask communication follow the rules shown in "Chapter 2. Buffer Standard of MERVA ESA" on page 3.

Starting Communication

Before making a request, you must establish the communication with DSLNTS by allocating an intertask communication block (ICB):

```
DSLNIC TYPE=ALLOC,MF=(E,NICPLST)
```

TYPE=ALLOC causes DSLNICT or DSLNICP to search for a free ICB, set it to "in use" status, and store its address in the DSLNIC parameter list of the requestor.

Note: General register 13 must point to a save area of 18 fullwords defined in the storage of the requestor, because DSLNICT uses MVS linkage conventions with save area chaining. In the MF=(E,NICPLST) parameter, E shows that DSLNICT is to be called, and NICPLST is the label of the DSLNIC MF=L parameter list. After a successful TYPE=ALLOC, the address of this parameter list must not be changed for subsequent DSLNIC TYPE=REQ or FREE requests.

If, after return from DSLNICT, the return code in general register 15 is 0 (or (decimal) 16 after the **shutdown** command), the ALLOC request was successful.

The requestor can now issue requests for MERVA ESA queue management, journal, user file, write-to-operator, command, or SWIFT Link authentication services; or check the status of MERVA ESA.

Requesting a Service

A MERVA ESA central service request is set up by preparing the data needed for its execution (a parameter list or data buffer or both); and then initiate the service request:

```
DSLNIC TYPE=REQ,NAME=DSLQMGT,                                  *
       PL=(R5),BUF=(R6),MF=(E,NICPLST)
```

TYPE=REQ sets up a central-service request.

Specify the name of the program that will execute the central service request (for a queue request, the name is DSLQMGT). This name must also be contained in the MERVA ESA task-server request table DSLNTRT.

Other valid names can be DSLNCS for the operator command service, DSLNUSR for the user file service, DSLJRNP for the journal service, DSLNMOP for writing messages to the MERVA ESA operators, and DWSAUTP for the SWIFT Link authentication service.

In the PL parameter, R5 (general register 5) contains the address of the queue parameter list. R6 (register 6) contains the address of the queue data buffer. Other requests can use only PL or only BUF. For the area not needed, the parameter can be omitted or coded with PL=0 or BUF=0.

The storage areas addressed with both PL and BUF follow the rules shown in "Chapter 2. Buffer Standard of MERVA ESA" on page 3. The first length field is needed by DSLNICT to determine if this area is large enough for the data returned from the servicing module. This length field is never changed. The second length field shows the actual data length to be transferred. If an area is required to pass information in one direction only, the actual length field can contain zero to avoid unnecessary data transfers. For a DSLQMGT TYPE=GET, the buffer is empty when the request is set up, then the actual length can be zero; but when the request is serviced by DSLQMGT and the response is returned, the actual length is no longer zero, and the retrieved message is transferred to the requestor.

The description of the DSLNIC macro in the *MERVA for ESA Macro Reference* lists whether PL and BUF must be specified for the known central services.

The MF parameter is the same as that in the TYPE=ALLOC macro instruction described here.

If the request setup is not successful, an appropriate return code is given by DSLNICT. If the request setup is successful, DSLNICT issues a WAIT for the completion of the service request.

After completion of the WAIT, DSLNICT checks if the request was serviced (as far as the intertask communication is concerned), and the appropriate return code is set.

If the request was serviced, DSLNICT moves all data back to the requester.

If, after return from DSLNICT, the return code in general register 15 is 0 (or (decimal) 16 after the **shutdown** command), the intertask communication was successful; with all other return codes the central service was not processed.

After successful intertask communication, the requester must check the return code of the servicing module. If a MERVA ESA command request is serviced, the return code of the command server is always zero. If a command is in error, this is indicated by an error response that is returned in the same response area of the buffer as the good responses.

MERVA ESA supports dynamic buffers for the central service request. This request allows you to retrieve large messages up to 2MB without knowing the length of the message in advance. TYPE=REQDYN is used to set up a central service request with dynamic buffers. The interface is the same as for the central service request with static buffers.

```
DSLNIC TYPE=REQDYN,NAME=DSLQMGT,                              *
      PL=(R5),BUF=(R6),MF=(E,NICPLST)
```

The TYPE=REQDYN returns a larger buffer when the data to be transferred does not fit into the provided buffer. It is the responsibility of the application program to check whether a larger buffer was returned and to release the storage of this buffer after use.

```
         C     R6,NICBUF           New buffer returned?
         BE    LABEL               ..No, use old buffer
         DSLSRV TYPE=FREEMAIN,ADSTOR=(R6),SIZE=0  Release old buffer
         L     R6,NICBUF           Get address of new dynamic buffer
         ST    R6,MYBUF            Save new buffer address for future
LABEL    DS    0H
```

Requesting a Status Check

A MERVA ESA status check is requested by repeating the ALLOC request:

```
DSLNIC TYPE=ALLOC,MF=(E,NICPLST)
```

As there is already an ICB address in the DSLNIC parameter list, only the MERVA ESA status is checked.

For requests, DSLNICT indicates, with return codes, whether MERVA ESA is being shut down or terminated, or whether it has been restarted. Then the requester can decide either to continue or to terminate, depending on the DSLNICT return code.

Terminating Communication

In the termination routines, the requester must free the allocated ICB:

```
DSLNIC TYPE=FREE,MF=(E,NICPLST)
```

This request can be issued regardless of the status of MERVA ESA, because DSLNICT itself checks whether it can free the ICB or not. Therefore, the requester need not check the return code of the FREE request.

If a requester terminates without freeing its ICB, that ICB cannot be used by another program until MERVA ESA is restarted.

Chapter 10. Using the Queue Management (DSLQMG)

The queues of all message-processing functions are collected in the queue data set which is held in a VSAM cluster and, if your system has been customized for large messages, in the large message cluster, or in a DB2® data base. If you do not use QDS on DB2, it is recommended that you customize the system for large message support.

There is no difference in using the queue management macro and API calls for QDS on DB2.

MERVA ESA queue management supervises and controls the activities concerned with the storage and retrieval of the messages in these data sets. It considers the queue data set and the large message cluster to be a unit and checks their integrity.

For a detailed description of MERVA ESA queue management services, refer to the *MERVA for ESA Concepts and Components*.

Keep the following rules in mind when you use the queue management macro:

- Only registers 2 through 11 can be specified if you are using register notation.
- When a queue management call is completed, the contents of registers 0, 1, and 14 are unpredictable. The content of register 15 is unpredictable for callers of a central service. For programs linked to DSLNUC, register 15 contains the return code of DSLQMGT.
- Return codes are in any case returned in the parameter list in the field QPLRETCD.
- You can overwrite parameter-list fields by specifying MF=(E,addr) and the appropriate parameters.
- A parameter-list field is not changed if the appropriate parameter is not specified in an MF=(E,addr), except for the MODIF parameter.

Building the Parameter List for a Queue Management Request

Use a DSLQMG MF=L macro to map the queue-parameter list for a queue management request. This macro reserves space for the queue-parameter list and assigns a symbolic name to each of the queue-parameter list fields. It also lists the DSLQMGT request types and return codes. With the parameter EXT=YES, the queue-parameter list extension is mapped also. This extension provides space for the result of a DSLQMG TYPE=ROUTE if more than 3 target queues have been set.

You can enter values in the queue parameter list either directly, using the symbolic field names assigned by the MF=L form of DSLQMG, or via the macro DSLQMG MF=(E,xxxx),..., where xxxx stands for the label of the queue-parameter list.

The two length fields at the beginning of the queue-parameter list must contain the appropriate values. This allows DSLQMGT to recognize whether the extension is used, and it allows the MERVA ESA intertask communication to transfer the queue-parameter list to DSLQMGT and back to the requestor.

Requesting Queue Management Services

Use a DSLQMG MF=L macro to define the queue-parameter list. Then invoke queue management using a DSLQMG MF=E macro to prepare the queue-parameter list for the request.

The parameter EP=DSLQMGT of the DSLQMG macro is reserved for programs linked to DSLNUC. Use the MERVA ESA intertask-communication macro DSLNIC for programs that are not linked to DSLNUC.

DSLQMGT only returns information after a successful DSLNIC TYPE=REQ,NAME=DSLQMGT call. The field QPLRETCD in the queue-parameter list contains binary zeros or one of the return codes listed in the queue-parameter list. The queue management return codes are also listed and explained in *MERVA for ESA Messages and Codes*.

Checking the Queue Status

In the following, the individual queue management requests, together with appropriate examples of their use, are discussed.

Use a DSLQMG TYPE=TEST macro to check the queue status. An example is shown below:

```
         DSLQMG TYPE=TEST,                                        *
                QUEUE=QUENAME,     QUEUE NAME                     *
                MF=(E,QPL)         QUEUE PARAMETER LIST USED
         DSLNIC TYPE=REQ,..        CALL INTERTASK COMMUNICATION
         ...
QUENAME  DS    CL8                QUEUE NAME FIELD
```

QUENAME is an 8-byte field containing the queue name that must be available in the MERVA ESA function table.

QPL is the label of the DSLQMG MF=L macro that defines the queue parameter list for this program.

On return, the fields of the queue-parameter list listed as shown in the following will have these contents:

QPLNQE The number of messages in the queue

QPLQSN The last queue sequence number used in the queue

QPLTRESH The threshold number of messages specified in the appropriate function-table entry

QPLRETCD The information code indicating whether this threshold has been reached.

Storing Messages

Messages are stored in up to twelve queues using DSLQMG. Use DSLQMG TYPE=PUT, TYPE=MPUT, or TYPE=ROUTE macros to store messages in queues. You can store messages in various ways:

- In one specific queue (PUT)
- In one, two, or three known queues (MPUT)
- In one to twelve queues determined by MERVA ESA routing (ROUTE).

These types can be changed by using keys or automatic delete, or both.

A queue element can be identified by one or two keys (KEY1 and KEY2) as described in the DSLQMG programming notes for key 1 and key 2 parameters in the *MERVA for ESA Macro Reference*.

There are four ways to store a message with a key in a queue:

1. DSLQMGT retrieves the keys from the message. The message must be in the MERVA ESA queue format, and the TOF field names must be defined in the KEY1 and KEY2 parameters of the appropriate function-table entry. The KEY=(0,0) parameter must be specified in the DSLQMG macro, or the two key fields in the QPL must be cleared to binary zeros before the DSLNIC or DSLQMG call.
2. Use the KEY parameter in the macro call KEY=(KEY1,KEY2). KEY1 and KEY2 must be fields of 24 bytes each. If the key is shorter, it must be padded with binary zeros or blanks. DSLQMGT takes from each key only the length specified in the function-table entry for each queue.
3. Use the DSLQMG macro without the KEY parameter. The KEY parameter can be omitted if values for KEY1 and KEY2 are stored in the fields QPLKEY1 and QPLKEY2 of the QPL.
4. Use the user exit of DSLQMGT for supplying keys independently of the items 1 to 3. The name of the exit program is DSLQKEY, and the material distributed contains a sample that explains the interface and makes the setup for that interface. In this user exit, any keys can be provided in the actual queue parameter list (for each queue of an MPUT or ROUTE) after inspecting the queue parameter list and the message in the data buffer. If the user exit does not provide keys, and also no keys have been provided by the requestor of the queue management service, DSLQMGT tries to get the keys in the way described under 1.

No matter which way is chosen for providing the keys, DSLQMGT will only use the keys when they are defined for the particular queue in the associated function table entry.

Automatic deletion of a message must be used if a message is:

- Retrieved from a queue (original queue)
- Updated and stored in another queue using the PUT, MPUT, or ROUTE request
- Deleted in the original queue.

Automatic deletion can be done even after a break-down of MERVA ESA in a restart of DSLQMGT. The information needed for automatic deletion is therefore also called “back-chaining information for restart.” The automatic deletion is provided by DSLQMGT if the input-queue name and the input-queue sequence number are specified with the RES and QSN parameters, respectively.

The programs delivered with MERVA ESA and the SWIFT Link use automatic deletion whenever it is appropriate.

Some coding examples follow.

PUT without Keys and without Automatic Delete

In the example below, a message is stored in a specific queue without a key:

```
          .
          .
          .
         L     R7,QUEBUFA          QUEUE BUFFER ADDRESS
         DSLQMG TYPE=PUT,          PUT TO QUEUE                        *
               DATA=(R7),          DATA FROM QUEUE BUFFER              *
               QUEUE=QUENAME,      QUEUE NAME FIELD                    *
               KEY=(0,0),          NO KEYS FROM THIS PROGRAM           *
               RES=0,              NO AUTOMATIC DELETE                 *
               QSN=0,              NO AUTOMATIC DELETE                 *
               MF=(E,QPL)          QUEUE PARAMETER LIST USED
         DSLNIC TYPE=REQ,...       USE INTERTASK COMMUNICATION
          .
          .
          .
QUENAME  DS    CL8                QUEUE NAME FIELD
```

Register 7 (R7) points to the queue buffer with the message. This message must be produced by the following call if the message is written to a queue that is processed by one of the MERVA ESA and the SWIFT Link programs:

```
DSLMFS TYPE=PUT,MEDIUM=QUEUE,...
```

If the message is processed by a user-written program, you can use any format after you have specified the buffer-length and message-length fields. The message contained in this buffer is written to the queue, with the queue name contained in the field QUENAME. The parameters QSN=0 and RES=0 prevent automatic deletion of the original message if this message was retrieved from a MERVA ESA queue.

After the DSLQMG macro is invoked, a DSLNIC TYPE=REQ,... macro is used to execute the MERVA ESA queue management request as a central service. The DSLQMGT return codes must be checked when DSLNICT shows successful completion of the intertask communication. If the caller is linked to DSLNUC, you must code the EP parameter of the DSLQMG macro instead of the DSLNIC macro.

PUT without Keys and with Automatic Delete

In the example below, a message is stored in a specific queue without keys and with automatic deletion:

```
         L     R7,QUEBUFA          QUEUE BUFFER ADDRESS
         DSLQMG TYPE=PUT,          PUT TO QUEUE                        *
               DATA=(R7),          DATA FROM QUEUE BUFFER              *
               QUEUE=QUENAME,      QUEUE NAME                          *
               KEY=(0,0),          NO KEYS FROM THIS PROGRAM           *
               QSN=INPQSN,         INPUT QUEUE SEQUENCE NUMBER         *
               RES=INPQNAME,       INPUT QUEUE NAME FOR AUTOM.DELETE   *
               MF=(E,QPL)          QUEUE PARAMETER LIST USED
         DSLNIC TYPE=REQ,...       USE INTERTASK COMMUNICATION
          .
          .
          .
QUENAME  DS    CL8                QUEUE NAME FIELD
INPQNAME DS    CL8                INPUT QUEUE NAME FIELD
INPQSN   DS    CL4                INPUT QUEUE SEQUENCE NUMBER FIELD
```

Register 7 (R7) points to the queue buffer where the message is stored. The message is written to the queue specified in the 8-byte field QUENAME. The message was retrieved from the queue with the name saved in the field INPQNAME and with the QSN saved in the field INPQSN. It was updated and is now to be stored in the next queue, and automatic deletion is established with the RES and QSN parameters.

MPUT with Keys and with Automatic Delete

In the example below, a DSLQMG macro specifies a message to be stored in one or more queues with keys and with automatic deletion of the original message:

```
         L     R7,QUEBUFA                   QUEUE BUFFER ADDRESS
         DSLQMG TYPE=MPUT,                  MESSAGE TO UP TO 3 QUEUES*
               DATA=(R7),                   DATA FROM QUEUE BUFFER   *
               QUEUE=(QUENAME1,QUENAME2,QUENAME3), QUEUE NAMES       *
               KEY=(KEY1,KEY2),             KEYS                     *
               QSN=INPQSN,                  INPUT QSN FOR AUTOM.DELET*
               RES=INPQNAME,                INPUT QUEUE NAME FOR A.D.*
               MF=(E,QPL)                   QUEUE PARAMETER LIST USED
          .
          .
          .
QUENAME1 DS    CL8              FIRST  QUEUE NAME FIELD
QUENAME2 DS    CL8              SECOND QUEUE NAME FIELD
QUENAME3 DS    CL8              THIRD  QUEUE NAME FIELD
INPQNAME DS    CL8              INPUT QUEUE NAME FIELD
INPQSN   DS    CL4              INPUT QUEUE SEQUENCE NUMBER FIELD
KEY1     DS    CL24             KEY 1 FIELD
KEY2     DS    CL24             KEY 2 FIELD
```

The message that is contained in the queue buffer pointed to by R7 is stored in the queues specified by the queue name fields QUENAME1, QUENAME2 and QUENAME3. If the MERVA ESA application supplies only one or two queue names, the fields QUENAME2 or QUENAME3 must be filled with binary zeros and the message is stored in one or two queues only.

In addition, automatic deletion is established to show that this message was retrieved from the original queue where DSLQMGT is to delete it directly after the MPUT. After the retrieval from the original queue, the QSN was saved in INPQSN, and the original queue name was saved in INPQNAME.

The fields KEY1 and KEY2 contain the keys supplied by the requesting program padded to 24 bytes with binary zeros or blanks.

ROUTE without Keys and with Automatic Delete

The DSLQMG macro causes MERVA ESA routing to be called by DSLQMGT. The message is stored in one to twelve queues, and automatic deletion is requested. The keys depend on the function-table entry specifications of the resulting queues:

```
         L     R7,QUEBUFA                   QUEUE BUFFER ADDRESS
         DSLQMG TYPE=ROUTE,                 LET DSLQMGT ROUTE        *
               DATA=(R7),                   DATA FROM QUEUE BUFFER   *
               QUEUE=INPQNAME,              ROUTE FROM THIS QUEUE    *
               KEY=(0,0),                   KEYS BY DSLQMGT          *
               QSN=INPQSN,                  INPUT QSN FOR AUTOM.DELET*
               RES=INPQNAME,                INPUT QUEUE NAME FOR A.D.*
               MF=(E,EUDQPL)
          .
          .
          .
INPQNAME DS    CL8              INPUT QUEUE NAME FIELD
INPQSN   DS    CL4              INPUT QUEUE SEQUENCE NUMBER FIELD
```

The message contained in the queue buffer pointed to by R7 is stored into the queues that result from the routing call using the input queue function defined by QUEUE=.

In addition, automatic deletion is requested from the original queue directly after the ROUTE. After the retrieval from the original queue, the QSN was saved in INPQSN and the original queue name was saved in INPQNAME. For a TYPE=ROUTE call, the QUEUE and RES parameters must specify the same input queue name if both are used.

The routing result is made available to the requestor of the queue management service in the three queue-name fields QPLNAM1, QPLNAM2, and QPLNAM3 of the QPL for the first three queues, and in the queue-parameter list extension for up to twelve queues. The routing result may be received without performing the PUT and DELETE functions, by specifying MODIF=RTNONLY. In this case, the routing criteria from the function table or the routing module, or both are analyzed and the target queue names are placed in the queue parameter list, but the PUT and DELETE functions are not performed.

Retrieving Messages

There are several ways to retrieve messages from specific queues: direct retrieval is carried out using a key or queue sequence number; sequential retrieval is carried out by obtaining one message after the other.

Note: Only one of the two possible keys is verified. If both keys are specified, only KEY1 is used by DSLQMGT. Use the MODIF parameter of the DSLQMG macro to request functional variations of direct or sequential retrieval.

The message retrieval methods are:

1. GET with key
2. GET with QSN
3. GETNEXT (sequential get)
4. GET or GETNEXT with MODIF=WRITEBCK (flag the message as being read if it is read another time)
5. GET or GETNEXT with MODIF=FREE (do not change the “in service” indicator)
6. GET or GETNEXT with MODIF=IGNINS (ignore the “in service” indicator)
7. GET or GETNEXT with MODIF=IGNHOLD (ignore the “hold” indicator)
8. GET or GETNEXT with MODIF=DYNBUF (allocate a larger message buffer if required).

The values of the MODIF parameter can be combined, for example, MODIF=(IGNINS,IGNHOLD).

The following coding examples show the retrieval methods.

GET with Key

This method is used to retrieve a specific message (previously stored with key) from a specific queue. The DSLQMG macro below shows how to directly retrieve a message by its key:

```
L     R7,QUEBUFA         BUFFER TO STORE THE MESSAGE
DSLQMG TYPE=GET,         RETRIEVE DIRECTLY                  *
       DATA=(R7),        BUFFER FOR DATA                    *
       QUEUE=QUENAME,    QUEUE NAME FIELD                   *
       KEY=(KEY1,0),     RETRIEVE WITH FIRST KEY            *
       QSN=0,            DO NOT USE QUEUE SEQUENCE NUMBER   *
       MF=(E,QPL)        QUEUE PARAMETER LIST USED
```

```
          .
          .
          .
          MVC   INPQSN,QPLQSN      SAVE QSN FOR ROUTE OR PUT
          MVC   INPQNAME,QUENAME   SAVE QUEUE NAME FOR ROUTE OR PUT
           .
           .
           .
QUENAME   DS    CL8                QUEUE NAME FIELD
INPQNAME  DS    CL8                INPUT QUEUE NAME FIELD
INPQSN    DS    CL4                INPUT QUEUE SEQUENCE NUMBER FIELD
KEY1      DS    CL24               KEY 1 FIELD
```

If the message is found in the queue with the specified key, it is returned to the storage area pointed to by R7.

The queue-parameter list contains the queue-sequence number of the message just read in the field QPLQSN; if the queue also uses the second key, it is contained in the QPLKEY2 field. The message is flagged "in service" in the queue from which it was retrieved. It cannot be retrieved by another GET/GETNEXT request until a DSLQMGT request with TYPE=FREE is issued, or a retrieve request with MODIF=IGNINS.

The two MVC instructions show how to save the QSN and queue name for other DSLQMGT requests that use the QSN for direct access (FREE, REPLACE, and DELETE) or the QSN and the queue name for automatic delete (PUT, MPUT and ROUTE).

GETNEXT (Sequential Read)

This method is used to retrieve the next message from the queue specified in the QUENAME field and return it to the storage area provided by the data parameter. Messages are stored in a queue, and a unique queue sequence number is assigned to each message in ascending order. For the message specified in the call or contained in the field QPLQSN, the next message is either:

- The message that has the next highest sequence number and that is not in service (if MODIF=IGNINS is not specified)
- The message with the next highest sequence number (if MODIF=IGNINS is specified).

When QPLQSN contains zero, the message with the lowest sequence number is retrieved.

The DSLQMG macro below shows how to retrieve messages sequentially:

```
          L     R7,QUEBUFA         ADDRESS OF QUEUE BUFFER
          DSLQMG TYPE=GETNEXT,     RETRIEVE NEXT AVAILABLE MESSAGE     *
                DATA=(R7),         BUFFER FOR DATA                     *
                QUEUE=QUENAME,     QUEUE NAME FIELD                    *
                QSN=0,             GET FIRST AVAILABLE MESSAGE         *
                KEY=(0,0),         DO NOT USE KEYS                     *
                MF=(E,QPL)         QUEUE PARAMETER LIST USED
          .
          .
          .
          MVC   INPQSN,QPLQSN      SAVE QSN FOR ROUTE OR PUT
          MVC   INPQNAME,QUENAME   SAVE QUEUE NAME FOR ROUTE OR PUT
           .
           .
           .
```

```
QUENAME  DS    CL8               QUEUE NAME FIELD
INPQNAME DS    CL8               INPUT QUEUE NAME FIELD
INPQSN   DS    CL4               INPUT QUEUE SEQUENCE NUMBER FIELD
```

After successful completion of the queue management request, the message in the queue buffer is ready for processing. In its fields QPLKEY1 and QPLKEY2, the QPL contains the key(s) of the message just read; the queue sequence number is in the QPLQSN field. The message is flagged as being in service in the queue from which it was retrieved. It cannot be retrieved by another GET/GETNEXT request until a DSLQMGT request with TYPE=FREE is issued, or a retrieve request with MODIF=IGNINS.

The two MVC instructions show how to save the QSN and queue name for other DSLQMGT requests that use the QSN for direct access (FREE, REPLACE, and DELETE) or the QSN and the queue name for automatic deletion (PUT, MPUT and ROUTE).

If you want sequential reading with MODIF=IGNINS, the returned QSN value, rather than QSN=0, must be used in a later GETNEXT request. This permits sequential reading of a complete queue without previously deleting or freeing the messages.

GET with MODIF=DYNBUF

This method is used to retrieve a message of unknown length into a buffer. If the buffer provided by the calling program is too small, MERVA ESA allocates a buffer large enough for the message. The message can be retrieved either with key or sequentially.

The handling of the allocated buffer in the calling program differs dependent on whether or not the calling program is link-edited to DSLNUC. The following two examples show the required coding.

The DSLQMG macro below shows how to directly retrieve a message by its key. The GET request is a central service request of a program not link-edited to DSLNUC.

```
         L     R7,QUEBUFA         BUFFER TO STORE THE MESSAGE
         DSLQMG TYPE=GET,         RETRIEVE DIRECTLY                   *
               DATA=(R7),         BUFFER FOR DATA                     *
               QUEUE=QUENAME,     QUEUE NAME FIELD                    *
               KEY=(KEY1,0),      RETRIEVE WITH FIRST KEY             *
               QSN=0,             DO NOT USE QUEUE SEQUENCE NUMBER    *
               MODIF=DYNBUF,      REQUEST DYNAMIC BUFFER              *
               MF=(E,QPL)         QUEUE PARAMETER LIST USED
         DSLNIC TYPE=REQDYN,...   CENTRAL SERVICE AND DYNAMIC BUFFER
         C     R7,NICBUF          NEW BUFFER RETURNED?
         BE    LABEL              ..NO, USE CURRENT BUFFER
         DSLSRV TYPE=FREEMAIN,ADSTOR=(R7),SIZE=0   FREE CURRENT BUFFER
         L     R7,NICBUF          GET NEW DYNAMIC BUFFER ADDRESS
         ST    R7,QUEBUFA         SAVE NEW BUFFER ADDRESS FOR FUTURE
LABEL    DS    0H
         .
         .
         .
         MVC   INPQSN,QPLQSN      SAVE QSN FOR ROUTE OR PUT
         MVC   INPQNAME,QUENAME   SAVE QUEUE NAME FOR ROUTE OR PUT
          .
          .
          .
QUENAME  DS    CL8                QUEUE NAME FIELD
```

```
INPQNAME DS    CL8               INPUT QUEUE NAME FIELD
INPQSN   DS    CL4               INPUT QUEUE SEQUENCE NUMBER FIELD
KEY1     DS    CL24              KEY 1 FIELD
```

If the message is found in the queue with the specified key, it is returned to the storage area pointed to by R7.

If DSLNICT allocated a new buffer, its address is returned in the field NICBUF of the DSLNIC parameter list. In this case the current buffer is freed and the address of the new buffer is saved in the field QUEBUFA for further processing. At the end of the program the storage of the new buffer whose address is in field QUEBUFA must be freed.

The DSLQMG macro below shows how to retrieve messages sequentially. The GETNEXT request is a direct service request of a program link-edited to DSLNUC.

```
         L     R7,QUEBUFA          ADDRESS OF QUEUE BUFFER
         DSLQMG TYPE=GETNEXT,      RETRIEVE NEXT AVAILABLE MESSAGE     *
               DATA=(R7),          BUFFER FOR DATA                     *
               QUEUE=QUENAME,      QUEUE NAME FIELD                    *
               QSN=0,              GET FIRST AVAILABLE MESSAGE         *
               KEY=(0,0),          DO NOT USE KEYS                     *
               MODIF=DYNBUF,       REQUEST DYNAMIC BUFFER              *
               EP=DSLQMGT,         DIRECT SERVICE REQUEST              *
               MF=(E,QPL)          QUEUE PARAMETER LIST USED
         C     R7,QPLAD            NEW BUFFER RETURNED?
         BE    LABEL               ..NO, USE CURRENT BUFFER
         DSLSRV TYPE=FREEMAIN,ADSTOR=(R7),SIZE=0   FREE CURRENT BUFFER
         L     R7,QPLAD            GET NEW DYNAMIC BUFFER ADDRESS
         ST    R7,QUEBUFA          SAVE BUFFER ADDRESS FOR FUTURE
LABEL    DS    0H
         .
         .
         .
         MVC   INPQSN,QPLQSN       SAVE QSN FOR ROUTE OR PUT
         MVC   INPQNAME,QUENAME    SAVE QUEUE NAME FOR ROUTE OR PUT
          .
          .
          .
QUENAME  DS    CL8                QUEUE NAME FIELD
INPQNAME DS    CL8                INPUT QUEUE NAME FIELD
INPQSN   DS    CL4                INPUT QUEUE SEQUENCE NUMBER FIELD
```

The next message from the queue specified in the QUENAME field that is not flagged as being in service is returned to the storage area pointed to by R7.

If DSLQMGT allocated a new buffer, its address is returned in the field QPLAD of the DSLQMGT parameter list. This buffer is now the user's buffer. At the end of the program the storage of this buffer allocated by DSLQMGT **must be freed** by the **calling** program.

Deleting Messages

Queue management permits the deletion of a message from a queue. The TYPE=DELETE keyword is used.

Message deletion consists of freeing the queue-key table entry for that message and removing the message from the queue data set. Messages must be deleted directly with their queue sequence numbers. Though the message deletion is also allowed by specifying key 1 or key 2, this method is not recommended, as the keys in a queue need not be unique, but the QSN is unique.

To empty a queue, you must delete one message after the other until you get the "queue empty" return code. The DELETE function with queue-sequence number causes a specific message to be removed from a specific queue. The message to be removed is identified by its QSN. The DSLQMG macro is used as shown in the following example:

```
          DSLQMG TYPE=DELETE,      DELETE A MESSAGE                    *
                 QUEUE=QUENAME,    QUEUE NAME FIELD                    *
                 QSN=INPQSN,       QSN FROM GET OR GETNEXT             *
                 KEY=(0,0),        DO NOT USE KEYS                     *
                 MF=(E,QPL)        QUEUE PARAMETER LIST USED
           .
           .
           .
QUENAME   DS    CL8               QUEUE NAME FIELD
INPQSN    DS    CL4               INPUT QUEUE SEQUENCE NUMBER FIELD
```

The field QPLQSN remains unchanged after completion of the DSLQMGT request. Sequential deletion of a logical queue can be done as follows:

- Issue a GETNEXT request with QSN=0 and MODIF=(IGNINS,IGNHOLD) to get the message with the lowest QSN in the queue (if you start DELETE with QSN=0, you may have many unsuccessful tries before you find the first message).
- Issue DELETE requests, increasing the QSN by one each time. In this way you delete all messages until the queue is empty.

Updating Queue Elements

The REPLACE function is used to write an updated queue element back to its queue with the same QSN so that the sequence of messages is not changed in the queue. The queue element to be updated must be identified by its QSN, as the QSN is always unique. Though the REPLACE function is also allowed by specifying key 1 or key 2, this method is not recommended as the keys in a queue need not be unique, but the QSN must be.

The length of the message can be changed. REPLACE does not change the **in service** indicator.

The DSLQMG macro is used as shown below:

```
          DSLQMG TYPE=REPLACE,     REPLACE A MESSAGE IN ITS QUEUE      *
                 DATA=(R7),        ADDRESS OF UPDATED MESSAGE          *
                 QUEUE=QUENAME,    QUEUE NAME FIELD                    *
                 QSN=INPQSN,       USE QSN FROM GET OR GETNEXT         *
                 KEY=(0,0),        DO NOT USE KEYS                     *
                 MF=(E,QPL)        QUEUE PARAMETER LIST USED
           .
           .
           .
QUENAME   DS    CL8               QUEUE NAME FIELD
INPQSN    DS    CL4               INPUT QUEUE SEQUENCE NUMBER FIELD
```

Freeing Messages

To free a specific message in a specific queue (that is, to change its status from **in service** to **not in service**), use the DSLQMG macro with TYPE=FREE.

The queue element to be freed must be identified by its QSN, as the QSN is always unique. Though the FREE function is also allowed by specifying key 1 or key 2, this method is not recommended as the keys in a queue need not be unique.

The FREE function switches off the **in service** indicator, thus enabling a waiting message-processing function to access this message. If the message was already flagged as being not in service, the FREE request shown below has no effect:

```
          DSLQMG TYPE=FREE,        FREE A MESSAGE                    *
                 QUEUE=QUENAME,    QUEUE NAME FIELD                  *
                 QSN=INPQSN,       QSN FROM GET OR GETNEXT           *
                 KEY=(0,0),        DO NOT USE KEYS                   *
                 MF=(E,QPL)        QUEUE PARAMETER LIST USED
           .
           .
           .
QUENAME   DS    CL8               QUEUE NAME FIELD
INPQSN    DS    CL4               INPUT QUEUE SEQUENCE NUMBER FIELD
```

Setting an ECB Address for a Queue

Programs linked to DSLNUC can be informed when a message is stored in a specific queue. To do this, an event control block (ECB) address is stored into the appropriate function table entry. DSLQMGT checks with each PUT, MPUT, ROUTE, and FREE request for an ECB address, and posts the ECB if an address is found.

To set an ECB address for a queue in its function table entry, a program linked to DSLNPTT must use the DSLQMG macro with TYPE=SET. Take care that, when posting is no longer needed, you use the TYPE=RESET to remove the ECB address. Also take care that you do not try to post more than one program from the same function table entry, that is, more than one program uses the TYPE=SET for the same queue name.

The DSLQMG macro is used as shown below:

```
          DSLQMG TYPE=SET,         SET AN ECB ADDRESS                *
                 QUEUE=QUENAME,    QUEUE NAME FIELD                  *
                 ECB=USERECB,      LABEL OF USER ECB                 *
                 EP=DSLQMGT,       CALL DSLQMGT DIRECTLY             *
                 MF=(E,QPL)        QUEUE PARAMETER LIST USED
          ...
QUENAME   DS    CL8               QUEUE NAME FIELD
USERECB   DC    F'0'              USER ECB
```

Note: The ECB is posted depending on the environment. The DSLSRVP parameter list in DSLCOM defines the post bit and the offset in the ECB. The test for the ECB being posted, and clearing the post bit is shown below:

```
          TM    USERECB+SRVPOFFS,SRVPOST ECB POSTED?
          BO    YES                      YES, PROCESS THE QUEUE
          BZ    NO                       NO, NOTHING TO DO
          ...
YES       DS    0H                       ECB IS POSTED
          NI    USERECB+SRVPOFFS,255|SRVPOST CLEAR POST BIT
          ...
          ...
NO        DS    0H                       ECB IS NOT POSTED
          ...
```

Resetting an ECB Address for a Queue

When a DSLQMG TYPE=SET macro was used to set an ECB address in the function table entry of a queue, this ECB address must be removed when posting is no longer needed. Otherwise unpredictable results can occur.

To reset an ECB address for a queue in its function table entry, use the DSLQMG macro with TYPE=RESET.

The DSLQMG macro is used as shown below:

```
          DSLQMG TYPE=RESET,       RESET AN ECB ADDRESS               *
                QUEUE=QUENAME,     QUEUE NAME FIELD                   *
                EP=DSLQMGT,        CALL DSLQMGT DIRECTLY              *
                MF=(E,QPL)         QUEUE PARAMETER LIST USED
           .
           .
           .
 QUENAME  DS    CL8                QUEUE NAME FIELD
```

Requesting a Queue List

Use the DSLQMG TYPE=LIST macro to request a list of the messages that are currently in a queue. The request must specify:

- Where to start in the queue
- Which messages to select
- How many list items to return.

To define the starting position, use the parameter QSN= to indicate the queue sequence number of the message where the list should begin. If the QSN is not found, the next highest QSN is used. When QSN=0 is specified, the list starts at the beginning of the queue.

You can also use the list modifier LMOD=FIRST to force the list processing to start with the first message in the queue.

Use the KEY= parameter to select only messages that contain a given value in the key fields. You can use either a specific or a generic key value. You can specify Key 1, Key 2, or both. When KEY=(0,0) is specified, all messages are eligible for the list response. Use the list modifier LMOD=BUSY to select only messages that are in-service.

The LNQE= parameter indicates the number of list items to be returned. The response buffer must be big enough to hold the requested items. Calculate the required buffer size as follows:

```
Buffer size = (Key 1 length + Key 2 length + 13) * LNQE  + 108
```

Use the macro DSLQMG MF=LIST to map the list response buffer, and ensure that the field QKLBUFLL contains the correct buffer length. The example below shows how to request information about the first 25 messages in a queue:

```
           L     R7,QUEBUFA         RESPONSE BUFFER ADDRESS
           DSLQMG TYPE=LIST,        REQUEST QUEUE LIST              *
                 QUEUE=QUENAME,     QUEUE NAME FIELD                *
                 DATA=(R7),         BUFFER FOR LIST RESPONSE        *
                 QSN=0,             NO STARTING QSN                 *
                 KEY=(0,0),         NO KEY FIELD SELECTION          *
                 LNQE=25,           RETURN 25 LIST ITEMS            *
                 MF=(E,QPL)         QUEUE PARAMETER LIST USED
           DSLNIC TYPE=REQ,...
           ...
  QUENAME  DS    CL8                QUEUE NAME FIELD
```

The list response buffer returns one list item for each message, with the following fields:

QKLQEQSN Queue sequence number

QKLQERBN Relative block number where the message resides in the queue data set

QKLQESTA In-service status of the message

QKLQEKEY Values of Key 1 and Key 2 for the message.

The length of the list item returned varies as the length of the keys vary from queue to queue. You find the lengths of Key 1 and Key 2 in the response fields QKLKLEN1 and QKLKLEN2 respectively. The total length of a list item is found in the field QKLQELEN. Other information about the queue is also returned:

QKLNAME Queue name

QKLNQE Number of messages currently in the queue

QKLTRESH Queue threshold value

QKLNSO Number of users currently signed on to the queue

QKLQSNF First QSN currently in the queue

QKLQSNL Last QSN currently in the queue

QKLQSNH Highest QSN assigned in this queue

QKLKFLD1 Name of Key 1 field

QKLKFLD2 Name of Key 2 field

QKLCOUNT Number of message elements is this response.

To continue the list, save the QSN of the last list item in the response and use this value plus one (+1) in the QSN= parameter of the next TYPE=LIST request. When the end of the queue is reached, the LISTEND reason code is returned in the field QPLRTNRS of the queue parameter list.

Extra Keys with DB2

With queue management using DB2, you can specify that additional keys should be stored for messages. To do this, specify XKEYS=YES for the corrsponding function in the function table DSLFNTT, and define the extra keys in the DB2 table DSLTQXDEF, as shown in the example below:

```
-----------------------------------------------------------------------
QUEUE  KEYNO  ACTIVE  KEYFIELD  STARTPOS  LENGTH  KEYDESC
-----------------------------------------------------------------------
L1DE0      3  Y       DSLEXIT          1       8  MERVA MESSAGE TYPE
L1DE0      4  Y       SWBH             1      48  SWIFT BASIC HEADER
L1DE0      5  Y       SWBHLT           1      48  LT ADDRESS
L1DE0      6  Y       SWAH             1      48  SWIFT APPL. HEADER
L1DE0      7  Y       SWAHID           1      48  INPUT / OUTPUT
L1DE0      8  Y       SWAHMT           1      48  MESSAGE TYPE
L1DE0      9  Y       SW103            1      48  SERVICE CODE
L1DE0     10  Y       SW108            1      48  MUR
L1DE0     11  Y       SW20             1      48  TRANSACTION REF. NO.
L1DE0     12  Y       SW32             1      48  SW32
L1DE0     13  Y       SW32DATE         1      48  VALUE DATE
L1DE0     14  Y       SW32CUR          1      48  CURRENCY
L1DE0     15  Y       SW32AMNT         1      48  AMOUNT
L1DE0     16  Y       SW50             1      48  ORDERING CUSTOMER
L1DE0     17  Y       SW59             1      48  BENEFICIARY CUSTOMER
```

Afterwards, whenever a message in that queue is inserted, updated, or deleted, MERVA ESA maintains these extra keys. For example, when an MT100 message is inserted into L1DE0, the following extra key values could be stored:

```
-------------------------------------------------------------------------------
QUEUE   QSN  KEYNO  KEYFIELD  ST.   LEN. KEYDESC              KEYVALUE
-------------------------------------------------------------------------------
L1DE0   270     3  DSLEXIT     1     8  MERVA MESSAGE TYPE   S100
L1DE0   270     4  SWBH        1    48  SWIFT BASIC HEADER   F01VNDEBET2A...
L1DE0   270     5  SWBHLT      1    48  LT ADDRESS           VNDEBET2AXXX
L1DE0   270     6  SWAH        1    48  SWIFT APPL. HEADER   I100VNDOBET2...
L1DE0   270     7  SWAHID      1    48  INPUT / OUTPUT       I
L1DE0   270     8  SWAHMT      1    48  MESSAGE TYPE         100
L1DE0   270    10  SW108       1    48  MUR                  MUR12345
L1DE0   270    11  SW20        1    48  TRANSACTION REF. NO. TRN12345
L1DE0   270    12  SW32        1    48                       990707USD12,34
L1DE0   270    13  SW32DATE    1    48  VALUE DATE           990707
L1DE0   270    14  SW32CUR     1    48  CURRENCY             USD
L1DE0   270    15  SW32AMNT    1    48  AMOUNT               12,34
L1DE0   270    16  SW50        1    48  ORDERING CUSTOMER    ADAM AMEISE
L1DE0   270    17  SW59        1    48  BENEFICIARY CUSTOMER /1234567890
```

DSLQMGT User Exits

There are four user exits available in MERVA ESA queue management. The material distributed with MERVA ESA contains samples that explain the interface and how to use it.

DSLQKEY This exit allows keys to be supplied for a new message in a queue.

DSLQPUT This exit allows you to access every message immediately before it is written to a queue. The QDS block may be written to a user data set.

DSLQTRA This exit is called in DSLQMGT for each request that processes a message. The same information is prepared for DSLQTRA as for the queue trace (as for QTRACE=LARGE in DSLPRM), but, the queue trace need not be active. DSLQTRA can inspect the journal buffer for the queue trace and perform additional processing. If the queue trace is active, DSLQTRA can decide if this particular record is written to the MERVA ESA journal or not after DSLQTRA has completed its processing.

DSLQUMR This exit allows you to record the unique message reference (UMR) when it is assigned.

DSLQMGD User Exits for Queue Management Using DB2

There are three user exits available in MERVA ESA queue management using DB2. The material distributed with MERVA ESA contains samples that explain the interface and how to use it.

DSLQKEY This exit allows keys to be supplied for a new message in a queue.

DSLQTRAB This exit is called in DSLQMGD for each request that processes a message. The same information is prepared for DSLQTRAB as for the queue trace (as for QTRACE=LARGE in DSLPRM), but the queue trace need not be active. DSLQTRAB can inspect the journal buffer for the queue trace and perform additional processing. If the queue trace is active, DSLQTRAB can decide if this particular record is written to the MERVA ESA journal or not after DSLQTRAB has completed its processing.

DSLQUMR This exit allows you to record the unique message reference (UMR) when it is assigned.

Chapter 11. Using the Journal Service (DSLJRN)

Depending on the calling program, the MERVA ESA journal service can be used as:

- Direct service
- Central service.

See also "Chapter 1. Types of MERVA ESA Application Programs" on page 1. The DSLJRN macro is described in *MERVA for ESA Macro Reference*.

Defining the Parameter List

Regardless of the way the journal service is used, the parameter list for journal program DSLJRNP is defined with the following macro:

```
JRNPL    DSLJRN MF=L
```

The buffer used for journal requests follows the rules shown in "Chapter 2. Buffer Standard of MERVA ESA" on page 3.

Using the Journal Service as Direct Service

Writing a Journal Record Directly

When DSLJRNP is used directly, a record is written to the journal using the following macro:

```
DSLJRN TYPE=PUT,MF=(E,JRNPL),DATA=...,JID=...,              *
       EP=DSLJRNP
```

Retrieving a Journal Record Directly

When DSLJRNP is used directly, a record is read from the journal using the following macros:

```
DSLJRN TYPE=GET,MF=(E,JRNPL),DATA=...,UKEY=...,             *
       EP=DSLJRNP
```

Only records from the current journal data set can be retrieved. If MERVA ESA has switched from journal A to journal B, the journal A is no longer accessible for retrieving records.

Using the Journal Service as Central Service

The MERVA ESA journal program DSLJRNP is also implemented as a central service.

Writing a Journal Record

A record is written to the journal using the following macros:

```
DSLJRN TYPE=PUT,MF=(E,JRNPL),DATA=...,JID=...
DSLNIC TYPE=REQ,NAME=DSLJRNP,PL=...,BUF=...
```

The DSLNIC macro must be followed by a check of the return codes.

Retrieving a Journal Record

A record is read from the journal using the following macros:

```
DSLJRN TYPE=GET,MF=(E,JRNPL),DATA=...,UKEY=...
DSLNIC TYPE=REQ,NAME=DSLJRNP,PL=...,BUF=...
```

The DSLNIC macro must be followed by a check of the return codes.

Chapter 12. Using the Operator Interfaces

MERVA ESA provides the following types of operator interfaces:

- The operator interface program (DSLNMOP)
- The write-to-operator program (DSLWTOP)
- The write-to-operator user exit (DSLWTOEX).

For details on the macros DSLNMO and DSLWTO refer to the *MERVA for ESA Macro Reference*.

Using the Operator Interface Program (DSLNMOP)

The MERVA ESA operator Interface DSLNMOP is used by programs linked to DSLNUC to present unsolicited operator messages to the MERVA ESA operators, and, optionally, to add them to the MERVA ESA journal or write them to the operating system console or both. The MERVA ESA operators can see the messages with the **dm** command.

Depending on the calling program, the MERVA ESA Operator interface can be used as:

- Direct service
- Central service

(See also Chapter 1. Types of MERVA ESA Application Programs.)

Defining the Parameter List

Regardless of the way the operator interface is used, the parameter list for DSLNMOP is defined with the following macro:

```
NMOPL    DSLNMO MF=L
```

The buffer used for the operator interface follows the rules shown in "Chapter 2. Buffer Standard of MERVA ESA" on page 3.

Using the Operator Interface as Direct Service

As a direct service, presentation for the **dm** command, the journal and the operating system console can be used.

The operator message can be prepared using the facilities of DSLOMSG. The DSLNMO macro must be coded as follows:

```
DSLNMO TYPE=PUTJC,MF=(E,NMOPL),FROM=MSGBUF,                *
       EP=DSLNMOP
```

TYPE=PUTJC Presents the operator message for the **dm** command, the journal and the operating system console.

TYPE=PUTJ Presents the operator message for the **dm** command and the journal.

TYPE=PUTC Presents the operator message for the **dm** command and the operating system console.

TYPE=PUT Presents the operator message for the **dm** command only.

MSGBUF Is the buffer with the operator message.

Using the Operator Interface as Central Service

As a central service, presentation for the **dm** command, the journal and the operating system console can be used. The presentation for the operating system console is only recommended for MERVA ESA applications running in the same region as DSLNUC, that is, in MERVA ESA running under CICS for CICS tasks, as the presentation for the operating system console is done in the region of DSLNUC. This can cause confusion if the MERVA ESA application is running in a region other than DSLNUC. DSLNMOP does not, however, reject such a request.

The operator message can be prepared using the facilities of DSLOMSG.

The two length fields at the beginning of the DSLNMO parameter list must be filled for the intertask communication.

The DSLNMO macro must be coded as follows:

```
DSLNMO TYPE=PUTJC,MF=(E,NMOPL),FROM=MSGBUF
DSLNIC TYPE=REQ,NAME=DSLNMOP,PL=NICPLST,BUF=MSGBUF,...
```

The DSLNIC macro must be followed by a check of the return codes.

For the types PUTJC, PUTJ, PUTC, and PUT the same functions are carried out as for using DSLNMOP as direct service.

Using the Write-to-Operator Program (DSLWTOP)

The MERVA ESA Write-to-Operator program DSLWTOP is used by programs not linked to DSLNUC to present unsolicited operator messages on the operating system console, and, if MERVA ESA is ready, also to the MERVA ESA operators (for the **dm** command) and the MERVA ESA journal. For the last two functions, DSLWTOP uses DSLNMOP (DSLNMO TYPE=PUTJ) and the MERVA ESA intertask communication (DSLNIC TYPE=REQ). For this purpose, the field COMNICPL must be filled (see “Filling the Fields of DSLCOM” on page 5 for how to fill this field). When the field COMNICPL is not filled, the unsolicited operator messages are only written to the operating system console.

The MERVA ESA Write-to-Operator interface is a direct service for programs that are not linked to DSLNUC. Programs linked to DSLNUC must use DSLNMOP.

Defining the Parameter List

The parameter list for DSLWTOP is defined with the following macro:

```
WTOPL    DSLWTO MF=L
```

The buffer used for DSLWTOP interface follows the rules shown in “Chapter 2. Buffer Standard of MERVA ESA” on page 3.

Using the Write-to-Operator Interface

The operator message can be prepared using the facilities of DSLOMSG. The DSLWTO macro must be coded as follows:

```
DSLWTO MF=(E,WTOPL),FROM=MSGBUF
```

MSGBUF is the buffer with the operator message.

Using the Write-to-Operator User Exit (DSLWTOEX)

MERVA ESA provides a user exit for the two operating console interfaces, DSLNMOP and DSLWTOP, which is given control before a WTO message is given to the MVS system console. This exit is not used in VSE. The exit DSLWTOEX allows a MERVA ESA installation to set the routing and descriptor codes for the WTO message. The material distributed with MERVA ESA contains a sample that explains the interface and makes the setup for that interface, and also the routing and descriptor codes available in MVS. The message for the operating system console is available for inspection to set the routing and descriptor codes depending on the message. The exit can leave the routing and descriptor codes unchanged (return code = 0 in register 15) or provide them in the work field for insertion in the WTO message (return code 4 in register 15).

Chapter 13. Coding MERVA ESA Applications for Automatic Start

User-written applications defined as transactions can be started automatically if they are associated with a MERVA ESA function. To associate a transaction with a function, use the DSLFNT parameters TRAN and LTERM to specify a transaction code and a logical terminal name (if applicable) in the function-table entry of the associated function. If the TRAN parameter value is specified as an entry in the transaction table DSLTXTT, the specifications made there are used to start the transaction. If the function is in NOHOLD status, or, in IMS, if the function is in ACTIVATED status and the parameter IGNACT (ignore activated) is specified, the specified transaction is started automatically whenever a message is written to the queue of that function.

It is also possible to delay the start of the transactions for a specific time period or until a batch of messages has accumulated. You can use this feature to reduce the initialization overhead of transactions. How to do this is explained in the descriptions of the DSLFNT and DSLTXT macros in the *MERVA for ESA Macro Reference*.

Examples of automatically started transactions are the MERVA ESA hard-copy printer program DSLHCP, and the MERVA ESA checking and expansion program DSLCXT.

Automatically started transactions differ from those started off-line only in the way by which they determine which MERVA ESA queue to process.

When executing the PUT, MPUT, ROUTE, FREE, or START request, MERVA ESA queue management checks the appropriate function-table entry for a transaction name. MERVA ESA then looks up the transaction name in the transaction table to determine which method to use to start the transaction. If the transaction is not specified, or if no method is specified for the transaction, it is started according to the local DC-environment where MERVA ESA is running.

In MERVA ESA running under CICS, the transaction is started using the command `EXEC CICS START` together with the logical terminal name, if it is available.

In MERVA ESA IMS, the transaction is started using:

- A CHANGE call to specify the transaction name
- An INSERT call to enter the data into the IMS message queue
- A PURGE call to show the end of the message.

The transaction to be started automatically must be a nonconversational IMS transaction.

For MVS, the transaction can also be started using the external CICS interface or using APPC/MVS services. These methods allow CICS or IMS transactions to be started from within MERVA ESA when it is running as a native batch program.

The data delivered with the EXEC CICS START command or the IMS INSERT is contained in the MERVA ESA terminal/user control block (TUCB) which is defined with the following macro:

```
DSLMFS TYPE=MAP,MF=TUCB
```

Under CICS, this data area must be defined in transaction storage; under IMS, it can be defined in the working storage of the program.

To retrieve the TUCB under CICS, an EXEC CICS RETRIEVE command is used:

```
EXEC CICS RETRIEVE INTO(TUCBAREA) LENGTH(TUCBLL)
```

In this example, TUCBAREA is the label of the TUCB as defined in the example, and TUCBLL is the length of the TUCB.

To retrieve the TUCB under IMS, a GET UNIQUE call is used:

```
          .
          .
          .
USRGU00  DS    0H
          .
         LA    R1,USRGUPL          ADDR OF GET UNIQUE PARM LIST
         L     R15,=V(ASMTDLI)     ADDR OF IMS INTERFACE
         BALR  R14,R15             GET UNIQUE CALL
         L     R2,USRGUPL+8        PCB ADDR
         CLC   10(2,R2),=C'QC'     QUEUE EMPTY ?
         BE    USRTRM00            ..YES, TERMINATE
         CLC   10(2,R2),=C'  '     CALL SUCCESSFUL ?
         BNE   USRDUMP0            ..NO, PRODUCE DUMP
          .
          .
          .
```

When the transaction is started using APPC/MVS, the area retrieved with the GET UNIQUE call contains a prefix area before the TUCB. This prefix area contains another 4-byte length field, the transaction code of 1 to 8 characters, and a blank. The real TUCB immediately follows this prefix area, and begins with another length field. An application program using this interface must handle the TUCB appropriately.

After successful retrieval of the TUCB, a copy of the MERVA ESA function-table entry is available with information such as the function (queue) name, the queue sequence number of the new queue element (only for PUT, MPUT and ROUTE), the transaction code, the logical terminal name, and format specifications. This information enables the MERVA ESA application to access the MERVA ESA queue and to use the MERVA ESA services to process the message as needed.

As additional information, the byte TUCQUEST of the TUCB contains the bit TUCSTART, which shows if the transaction was started by the MERVA ESA operator command **sf** (start function) instead of by a message written to the MERVA ESA queue.

CICS transactions can always process a MERVA ESA queue until it is empty.

IMS transaction should, however, only process one message (or as many as indicated in the field FNTMLIM), and should then insert the TUCB in their own IMS message queue and give back control to IMS. IMS will give them control again later but is able to process other transactions in the meantime.

Chapter 14. Changing the MERVA ESA End-User Driver (DSLEUD)

Figure 6 on page 128 gives an overview of the MERVA ESA End-User Driver (DSLEUD). All function programs of DSLEUD are defined in the End-User-Driver program table DSLEPTT, except DSLEERR, DSLEU001, DSLEU002, and DSLEU003 and DSLEU004. Programs loaded by DSLEUD are not shown in the overview.

DSLEPTT describes the function programs and their command tables.

Changing DSLEUD means:

- Changing DSLEPTT
- Changing user commands
- Coding DSLEUD user-exit routines
- Adding a new function program

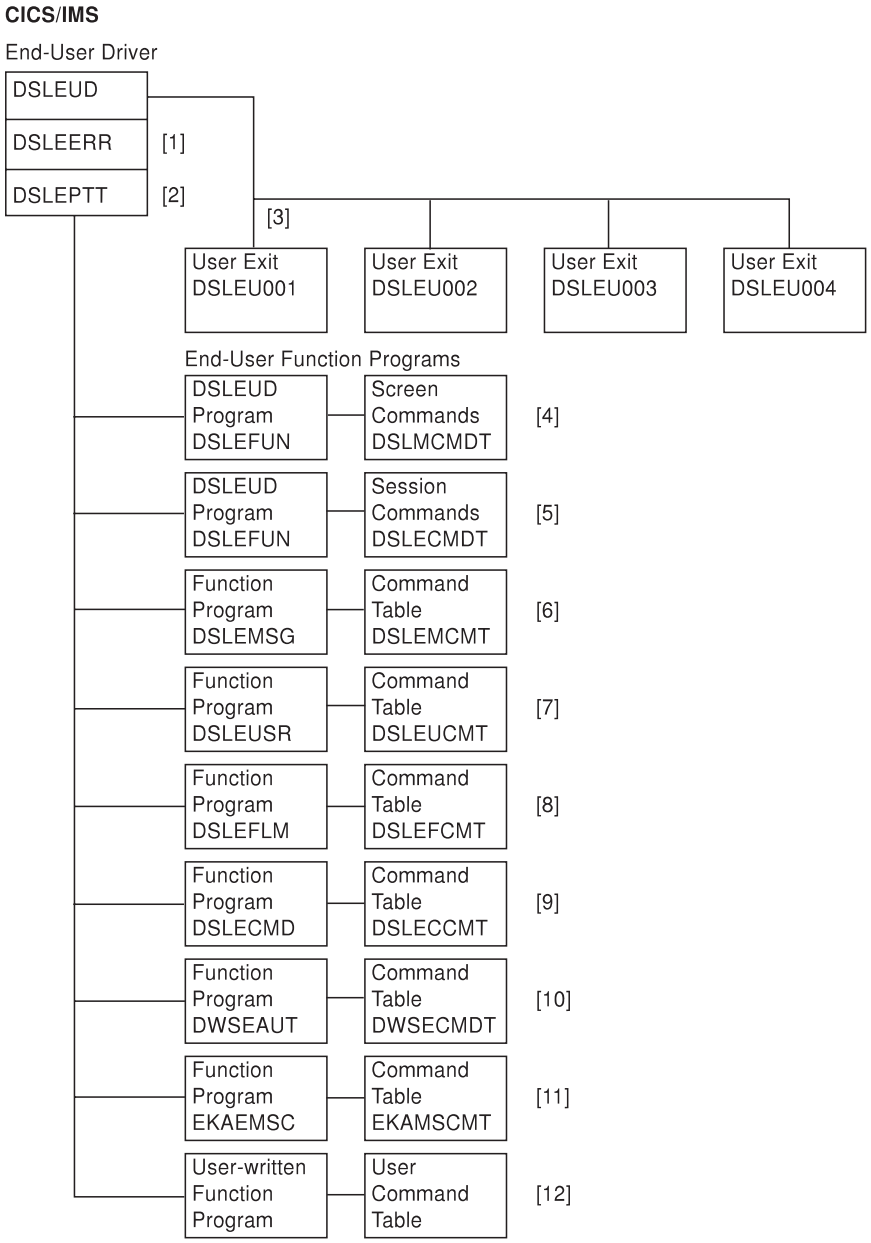


Figure 6. Overview of the End-User Driver

The following notes refer to Figure 6.

Notes:

1. DSLEERR is the error message routine of DSLEUD.

   It is called using the macro DSLEEMSG. DSLEERR uses DSLOMSG to create the error messages. The error messages of the End-User Driver are defined in the copy member DSLEMSC, which is used in the message table DSLMSGT.
2. DSLEPTT is the program table of DSLEUD that defines the function programs and their command tables.
3. End-User-Driver User-Exits are:

   DSLEU001 called at sign-on time.

   DSLEU002 called at every screen input time after sign-on.

DSLEU003 called at sign-off time.

DSLEU004 called for password validation in USR function.

4. DSLEFUN is the function selection program of DSLEUD, and the only purpose of this entry of DSLEPTT is to define the display and edit commands in the command table DSLMCMDT. The display and edit commands are executed by MERVA ESA Message Format Service programs.
5. DSLEFUN is the function selection program of DSLEUD, and its command table DSLECMDT defines the user session commands.
6. DSLEMSG is the message processing program of DSLEUD, and its command table DSLEMCMT defines the user message selection and message processing commands.
7. DSLEUSR processes the User File maintenance, and its command table DSLEUCMT defines the User File maintenance commands.
8. DSLEFLM is the general file maintenance program of DSLEUD, and its command table DSLEFCMT defines the file maintenance commands.
9. DSLECMD is the operator command program of DSLEUD and its command table DSLECCMT defines the MERVA ESA queue commands. All other commands are the MERVA ESA and SWIFT Link operator commands that are passed to the MERVA ESA command server (DSLNCS, linked to DSLNUC).
10. DWSEAUT is the SWIFT Link Authenticator Key-File Maintenance program of DSLEUD, and its command table DWSECMDT defines the authenticator-key file maintenance commands.
11. EKAEMSC is the MERVA ESA System Control program. Its command table EKAMSCMT defines the MERVA Link control commands.
12. A user-written program is added to the End-User Driver via DSLEPTT, and it can have an own user command table.

Changing DSLEPTT

Changing the user program table means:

1. Adding a user-written function program.
2. Changing the name of an existing command table. This can be useful if the name of the commands are changed in a MERVA ESA installation, and the source code of the command table is to be kept for maintenance purposes.
3. Deleting function programs which are not used in a MERVA ESA installation, for example, the general file maintenance program DSLEFLM, or the authenticator-key file maintenance program if these services are not used, for example, if these files are maintained with the batch utilities only.

Changing End-User Command Tables

General-use programming interface

The MERVA ESA user has commands for various purposes. There are three groups of commands defined in separate command tables:

- Display and edit command table DSLMCMDT
- Session command table DSLECMDT
- Function command tables:
  - Message selection/processing commands DSLEMCMT
  - User file maintenance commands DSLEUCMT

- Authenticator-key file maintenance commands DWSECMDT (SWIFT Link)
- General file maintenance commands DSLEFCMT
- Queue commands DSLECCMT
- MERVA ESA System Control commands EKAMSCMT.

The customizing of these command tables is discussed in the following.

End of General-use programming interface

Display and Edit Command Table (DSLMCMDT)

General-use programming interface

The display and edit command table contains the definition of the display and edit commands.

Note: The display and edit command table allows the user to define synonyms, translations and abbreviations for display and edit command codes. Abbreviated display and edit command names can be defined for the existing commands, such as LIN for LINE. Both must have the same display and edit command code 03.

New display and edit commands can be defined, if the user writes a program to execute these commands. Figure 7 contains an example of a display and edit command table.

```
DSLMCMDT DSLNCM TYPE=INITIAL                          [1]
*
SHOW     DSLNCM 01,(8)
END      DSLNCM 02
LINE     DSLNCM 03,(5),(5)                            [2]
LIN      DSLNCM 03,(5),(5),DESC=LINE                  [3]
PAGE     DSLNCM 04,(5),(5)
PAG      DSLNCM 04,(5),(5),DESC=PAGE
OCC      DSLNCM 05,(8)
HELP     DSLNCM 06,(8)
FORM     DSLNCM 07,(8)
NOPROMPT DSLNCM 08
NOP      DSLNCM 08,DESC=NOPROMPT
PROMPT   DSLNCM 09,(5),(5)
PRO      DSLNCM 09,(5),(5),DESC=PROMPT
PFKEYS   DSLNCM 10,(8)
SOCC     DSLNCM 11,(5),(5)
DOCC     DSLNCM 12
ERASE    DSLNCM 13,(8)
FIND     DSLNCM 14,(24)
INSERT   DSLNCM 15,(8)
SREP     DSLNCM 16,(5),(5)
UL       DSLNCM 17,(8)
*
NEWCMD   DSLNCM 91,NAME=NEWCMDEX                      [4]
NC       DSLNCM 91,NAME=NEWCMDEX,DESC=NEWCMD          [5]
*
         DSLNCM TYPE=FINAL                            [6]
         END
```

Figure 7. Display and Edit Command Table

Notes:

1. DSLNCM TYPE=INITIAL

   This must be the first macro, and it assigns the label DSLMCMDT to the display and edit command table.

2. LINE DSLNCM 03,(5),(5)

   The name of this display and edit command is LINE, and it is assigned the command code '03'. Two optional parameters with maximum length of 5 each are specified.

3. LIN

   The program accepts the input of LIN as an abbreviated command name for LINE. The DESC parameter is specified to indicate to MERVA ESA the full command name for command responses and checking for allowed commands from the user profile.

   All commands (including abbreviated command names) must be unique names.

4. NEWCMD DSLNCM 91,NAME=NEWCMDEX

   A new display and edit command NEWCMD (code = 91) is defined; no parameter is defined. The command is executed by the user-written routine NEWCMDEX.

   This and all other command execution routines must give back control to DSLEUD after having executed the command.

5. NC

   The abbreviation of NEWCMD is NC. The same command code and execution routine are defined. DESC=NEWCMD specifies that, in command responses, the full command word 'NEWCMD' is shown.

6. DSLNCM TYPE=FINAL

   This must be the last macro and is followed by the Assembler END statement.

Note: If no command execution module is defined for a display and edit command, a DSLMFS TYPE=COMMAND service is called for execution.

End of General-use programming interface

How to Process the Changed Display and Edit Command Table

General-use programming interface

After modification, the display and edit command table DSLMCMDT must be assembled and DSLEUD must be link-edited.

End of General-use programming interface

Session Command Table (DSLECMDT)

General-use programming interface

The following is an example of a Session Command Table:

```
DSLECMDT DSLNCM TYPE=INITIAL                          [1]
SIGNOFF  DSLNCM 04,NAME=0                             [2]
SOF      DSLNCM 04,NAME=0,DESC=SIGNOFF                [3]
```

```
RETURN   DSLNCM 08,(8),NAME=0
RET      DSLNCM 08,(8),NAME=0,DESC=RETURN
HARDCOPY DSLNCM 16,NAME=0
HCO      DSLNCM 16,NAME=0,DESC=HARDCOPY
RETRIEVE DSLNCM 24,NAME=0
RET1     DSLNCM 24,SYN='?',NAME=0,DESC=RETRIEVE       [4]
REPEAT   DSLNCM 28,NAME=0
REP1     DSLNCM 28,SYN='=',NAME=0,DESC=REPEAT
         DSLNCM TYPE=FINAL
         END
```

Notes:

1. DSLNCM TYPE=INITIAL

This must be the first macro and assigns the label DSLECMDT to the display and edit command table.

2. SIGNOFF DSLNCM 04,NAME=0

This macro assigns command code 04 to the session command SIGNOFF. NAME=0 need not be specified as NAME=0 is the default.

3. SOF DSLNCM 04,NAME=0,DESC=SIGNOFF

This macro defines an abbreviation SOF for the session command SIGNOFF. DESC=SIGNOFF specifies that SIGNOFF is displayed as command name after executing 'SOF'.

4. RET1 DSLNCM 24,SYN='?',NAME=0,DESC=RETRIEVE

This macro defines a synonym '?' for the session command RETRIEVE. DESC=RETRIEVE specifies that RETRIEVE is displayed as command name after executing '?'.

Note: SYN='?' must be specified because '?' is a special sign not allowed as an Assembler label.

Note: The command codes specified for the session commands are multiples of 4. This is required by the command processing program which is used as the default for session commands. If you supply additional session commands executed by your own program, you are free to interpret other command codes.

End of General-use programming interface

Function Command Tables

General-use programming interface

The function command tables can be customized by defining new command names and adding new commands, as described for the screen and session command tables.

End of General-use programming interface

Command Processing Restriction of the End-User Driver

General-use programming interface

For command processing, the End-User Driver uses the three command tables in a fixed order:

1. First, the display and edit command table (DSLMCMDT)
2. Second, the session command table (DSLECMDT)
3. Third (optional), the command table of a function program.

Throughout these tables, all command names, abbreviations and synonyms must be unique. When the MERVA ESA operator command processing function is active, the command table of DSLNUC (DSLNCMT) is included in the consideration of unique names. As only the command table of one DSLEUD function program is used at one time, it is possible to use the same command names in the command tables of different function programs.

Commands must have names that are different from the function names defined in the table DSLFNTT, which is described in *MERVA for ESA Customization Guide*. This is because input in the command line of a panel used for function selection is checked for being a function name before it is checked for being a command, so a command with the same name as a function would not be available.

End of General-use programming interface

Interface of an End-User Command Execution Routine

General-use programming interface

A command execution routine linked to DSLEUD has the following information in the general registers when it receives control:

Register 0:	Undefined.
Register 1:	Address of the DSLEUD Interface Area. This area can be mapped by a DSLECOFN macro.
Registers 2 to 12:	Undefined.
Register 13:	Address of the save area of DSLEUD. In this save area the command execution routine must save the DSLEUD registers according to MVS linkage conventions.
Register 14:	Return address to DSLEUD.
Register 15:	Entry-point address of the command-execution routine.

A command-execution routine must work in the following way:

- The DSLEUD registers must be saved and registers set up following MVS linkage conventions.
- The working storage required must be specified in the DSLEUD interface area.
- The DSLCOM must be addressed from the field FNCOM. The terminal user control block (TUCB) must be addressed from the field COMTUCBA. The

MERVA ESA parsing parameter list must be addressed from the field TUCBCMPA. The MERVA ESA parsing parameter list is mapped by the DSLNPA MF=L macro.

- The command input in the DSLNPA parameter list must be checked for validity; that is, the command must have been entered properly with all its required parameters.
- Command-execution processing must be done. The command response is either an operator message in the field FNMSLN (length including 4) and FNMSGT (message text), or a message identification (which must be known to DSLEUD) in the field FNMSID. All these fields are defined in the DSLEUD interface area.

  The response-message skeleton can be retrieved from the message table DSLMSGT using the DSLOMS macro that calls the DSLOMSG program. The addresses of DSLOMSG and DSLMSGT are contained in the fields COMOMSGA and COMMSGTA of DSLCOM.
- The command execution routine can set on the bit FNCCEX to show to DSLEUD that it wants to have control when the next input from the screen terminal is available, no matter what the end user has entered. This function is required when the present command wants an immediate answer from the user for a specific action, and an action must also be taken when the answer is not given (confirmation of the deletion of a record during an on-line file maintenance function).

  The command execution routine can set on the bit FNCEXI to show to DSLEUD that there is still a command to be executed when control is returned to DSLEUD. DSLEUD then processes the command passed. This facility can be used when one possible action to be taken can be processed by a command defined in one of the command tables.
- Return to DSLEUD. A command-execution routine must always return to DSLEUD after a command is executed and the response prepared.

When you return to DSLEUD, the general registers must contain the following information:

Registers 0 to 14:	Contents are the same as when control was received from DSLEUD.
Register 15:	A return code of zero.

End of General-use programming interface

Coding User Exits of DSLEUD

General-use programming interface

The following user exits are available for DSLEUD: DSLEU001, DSLEU002, DSLEU003, and DSLEU004.

The machine-readable material supplied with MERVA ESA contains samples of these user exits that explain the interface and make the setup for that interface. The user only needs to add his additional code in the samples.

The DSLCOM of DSLEUD allows for accessing other control blocks, for example, the address of the Terminal User Control Block (TUCB) is found in the field COMTUCBA. The TUCB supplies additional information for the user exit. The layout of the TUCB can be obtained by means of a DSLMFS MF=TUCB macro.

The DSLEUD user exits are not allowed to use Message Format Service, TOF services, or MERVA ESA queue management services. If you want to use such services, you must use one of the DSLMUnnn user exits.

The user exits of DSLEUD can have a permanent storage area throughout a user session. Its size is defined in DSLPRM with the USERSTO parameter of the DSLPARM macro. This permanent storage can also be accessed by all user function programs.

- DSLEU001 is called at sign-on time. It allows the use of an external security manager such as RACF® for sign-on, starting user applications, and so on. It has access to the sign-on data in the interface buffer where the items user identification, password, selected function and new password are already expanded to 8 bytes.
- DSLEU002 is called whenever screen input is entered after having signed on. This makes it possible to check, for example, the availability of user applications.
- DSLEU003 is called at sign-off time. It allows for stopping user applications.
- DSLEU004 is called when the password is checked at the entry to the User File Maintenance. When customized appropriately, it checks the password against an external security manager such as RACF. This is described in *MERVA for ESA Customization Guide*.

End of General-use programming interface

Writing a DSLEUD Function Program

General-use programming interface

The machine-readable material supplied with MERVA ESA contains a sample of a DSLEUD function program with the name DSLEFUPR. This sample explains the interface and makes the setup for that interface. Additional code can then be added to the sample.

The DSLCOM of DSLEUD allows for accessing other control blocks; for example, the address of the Terminal User Control Block (TUCB) is found in the field COMTUCBA. The TUCB supplies additional information for the user exit. The layout of the TUCB can be seen in the expansion of the DSLMFS MF=TUCB macro.

The DSLEUD function programs are allowed to use all MERVA ESA services such as Message Format Service, TOF services or MERVA ESA queue management services.

The function program is called for:

1. Initialization.

   This call is used to show to DSLEUD how much storage the function program needs. The program can have permanent storage (in the scratchpad area SPA) and temporary storage. The temporary storage is cleared after return to DSLEUD. The contents of the permanent storage are held throughout user session. The temporary storage is limited to 32760 bytes. The size of permanent storage depends on the amount of storage defined for buffers in the module DSLPRM, which is described in *MERVA for ESA Customization Guide*. The maximum is 4096 bytes. If the storage definitions exceed the maximum, the programmer is informed by an error message.
2. Processing. The program can do its function.

3. Termination. The program must terminate, that is, release all resources.

End of General-use programming interface

Error Messages of DSLEUD

General-use programming interface

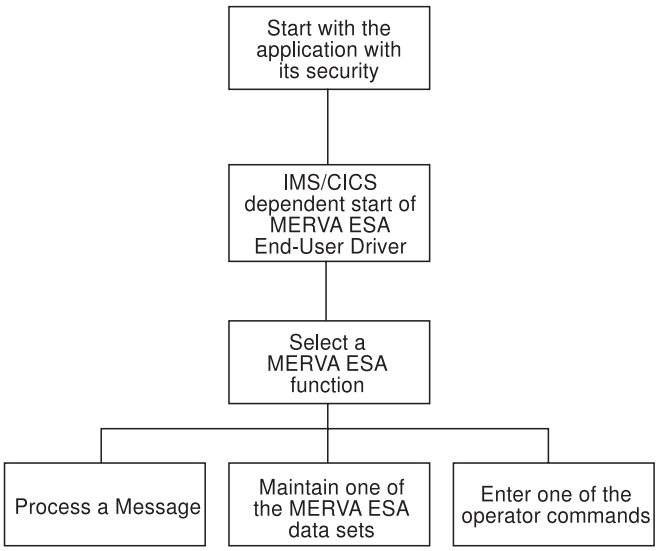
Error messages of DSLEUD are defined in the MERVA ESA message table (DSLMSGT) in the copy member DSLEMSC. For the DSLEUD user exits and user written function programs new error messages can be necessary.

End of General-use programming interface

Calling the End-User Driver by an IMS/CICS Application Program

General-use programming interface

You can start MERVA ESA by calling it from a user-written application program.



An IMS/CICS application program can call the End-User Driver (DSLEUD) to install MERVA ESA into your application flow. This is referred to as **program-to-MERVA switch**.

The program-to-MERVA switch must be defined specifying parameter PGCALL=YES of the DSLPARM macro.

Note: The password checking within MERVA ESA must be suppressed specifying parameter EXSEC=YES or EXSEC=(YES,NOCHECK) of the DSLPARM macro. The End-User Driver then assumes that all password security checks are done before it is called. For more information about the DSLPARM macro, refer to the *MERVA for ESA Macro Reference*.

The program-to-MERVA switch can force the End-User Driver to return into a predefined next transaction when the MERVA ESA session is signed off. This

depends on a particular field within the message buffer sent to the End-User Driver at start time. The message buffer, sent to the next application program when the End-User Driver terminates, starts with a MERVA ESA message identified by DSL*nnnn*. This MERVA ESA message indicates normal session end (=DSL1000) or an error (DSL1*nnn*). The next transaction program must verify the status.

End of General-use programming interface

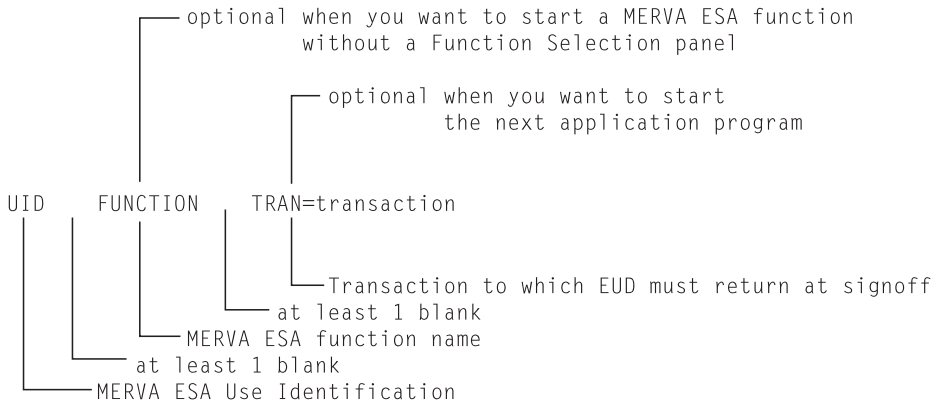
IMS Rules for the Program-to-MERVA Switch

General-use programming interface

You must consider the IMS rules for program-to-program switches as described in *IMS/ESA Application Programming: Transaction Manager* where passing the conversation to another conversational program is discussed.

The following shows the image of the message buffer that is needed to start an end-user driver session under IMS.

Note: This is different from the standard MERVA ESA buffer structure.



You must use the following IMS commands to start the End-User Driver by its predefined transaction name:

CHNG	Using alternate PCB to change TRAN name
ISRT	Inserting the scratchpad area (SPA)
ISRT	Inserting the message buffer to start the End-User Driver.

The "next" application program (started by the End-User Driver) has the usual IMS program initialization: Get Unique (GU) of SPA followed by a Get Next (GN) message. To verify that the program was started by EUD, check the MERVA ESA message "DSL*nnnn*" at the beginning of the buffer.

Notes:

1. The MERVA ESA End-User Driver needs the SPA in the length of 320 bytes. The started "next" transaction receives this area.
2. Before the SPA is inserted, your program must modify the SPA as follows:
   - It must override the transaction name in the SPA by the transaction name used in the CHNG command.

- Unless EXSEC=(YES,NOCHECK) is specified in the MERVA ESA customization parameter module DSLPRM, it must write at least one character not equal to X'00' to the user work area of the SPA. The user work area is the area following the transaction name up to the end of the SPA.

3. Even if the message buffer is to contain only the UID, at least one blank following the UID must be contained in the message buffer.

End of General-use programming interface

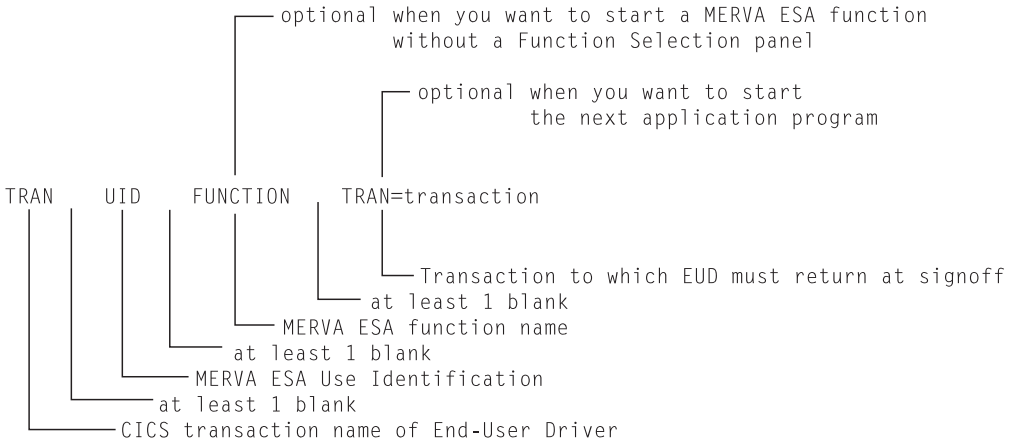
CICS Rules for the Program-to-MERVA Switch

General-use programming interface

The MERVA ESA nucleus must be started before a program-to-MERVA switch can take place. The MERVA ESA nucleus can be started automatically during CICS startup or manually.

The following shows the image of the message buffer that is needed to start an end-user driver session under CICS.

Note: This is different from the standard MERVA ESA buffer structure.



The following CICS command must be used to start the End-User Driver by its predefined transaction name; *aaa* is the message buffer to start the End-User Driver.

```
EXEC CICS START TRANSID(xxx) TERMID(EIBTRMID) FROM(aaa)
               LENGTH(nn)
```

The End-User Driver uses the same technique to start the “next” transaction program (when requested). The started program must use the following command to get the message from the End-User Driver:

```
EXEC CICS RETRIEVE .....
```

To verify that the program was started by EUD, check the MERVA ESA message “DSL*nnnn*” at the beginning of the buffer.

End of General-use programming interface

Writing the DSLEUD SPA File Program in IMS

In MERVA ESA for IMS, the storage used by DSLEUD for each end-user transaction is saved in the MERVA ESA SPA file between conversation steps. Only a SPA of 320 bytes is given to IMS. In MERVA ESA, the program DSLEOSPA saves the transaction storage in a SPA file (BDAM). The macro DSLEISPA describes and establishes the interface between DSLEUD and DSLEOSPA. Using the DSLEISPA macro, you can write your own SPA file program.

The SPA consists of four buffers in MERVA ESA. The fourth buffer is optional and allocated dynamically with a maximum size specified in the MAXBUF parameter of DSLPRM.

In considering the logical terminal name and the user ID, DSLEOSPA performs two functions:

1. If the command field in the interface area contains the four characters 'ISRT', the four SPA buffers of this user are saved in the SPA file.
2. If the command field in the interface area contains the four characters 'GU ', the four SPA buffers of this user are got from the SPA file.

The IMS SPA of 320 bytes is processed by DSLEUD.

Before the MERVA ESA sign-on, the logical terminal name is used rather than the user identifier.

The DSLEISPA macro defines the following:

- The layout of the interface area
- The register contents on entry to DSLEOSPA
- The field contents of the interface area on entry to DSLEOSPA
- The possible return and reason codes of DSLEOSPA.

Using an HDAM Database as SPA File

DSLEOSPA, a DSLEUD exit program, manages the SPA file of MERVA ESA as a BDAM file. An alternative exit program, DSLEOSPB, is provided to manage this SPA data in an IMS HDAM database. The installation steps to activate this alternative method are described in the *MERVA for ESA Installation Guide* in the chapter on the SPA File.

HDAM Database Structure

Segments: The SPA data for each end-user is held in one database record. The database record key is the logical terminal name.

The four parts (buffers) of the SPA data are stored in four segment types defined as children of the root segment. The first part of the End-User Driver permanent storage is stored in the root.

The segments are:

SPA1 Fixed length, the root segment. The first four words in the segment contain the length of the four SPA parts stored in this database record. The next 8 bytes contain the key, the logical terminal name. The remainder of the segment contains the first part of the End-User Driver permanent storage.

SPA2 Variable length, contains the remainder of the End-User Driver permanent storage. As many SPA2 segments are written, chained using twin-forward pointers, as are necessary to hold this permanent storage.

SPA3 Variable length, contains the TOF. As many SPA3 segments are written, chained using twin-forward pointers, as are necessary to hold the TOF.

SPA4 Variable length, contains the Logical Data Stream. As many SPA4 segments are written, chained using twin-forward pointers, as are necessary to hold the LDS.

SPA5 Variable length, contains the dynamic TOF extension, if any. As many SPA5 segments are written, chained using twin-forward pointers, as are necessary to hold this data.

The sizes of segments are defined by the database definition (DBD). They are also defined in DSLEOSPB. The size or name of segments cannot be changed.

The segment sizes have been chosen assuming a VSAM control interval size of 16384 bytes. This size is efficient on both 3380 and 3390 devices. The sizes are defined as follows:

- SPA1: 4068 bytes
- SPA2: 4084 bytes
- SPA3: 4084 bytes
- SPA4: 4084 bytes
- SPA5: 16366 bytes

These sizes are the amount of application data stored in each segment. For variable-length segments, the size includes the 2-byte segment length field.

The maximum overall segment size, including segment code, delete byte, and pointers is 4090 bytes for all segments except for SPA5, which has an overall length of 16372.

Pointers: Database records are always accessed in hierarchical sequential sequence, so it is not necessary to have backward pointers.

Because the standard IMS randomizer might generate the same RAP number for different keys, the root segment must have a physical twin-forward pointer. The root segment also has a physical child-first pointer to each of the child segments, SEG2 to SEG5. Each of the child segments can have an unlimited number of twin segments, so these segments have physical twin-forward pointers.

Including the segment code and delete bytes, the segment prefix sizes are:

- SEG1: SC + Del + PTF + 4 x PCFs = 22
- SEG2 - SEG5: SC + Del + PTF = 6

Refer to the *IMS/ESA Administration Guide: Database Manager* if you need more information about these calculations.

Chapter 15. Application Programs Linked to DSLNUC

General-use programming interface

There are three types of application programs linked to DSLNUC:

- Nucleus programs (NPT programs) defined in the DSLNPTT
- Central services defined in the DSLNTRT
- Command execution routines defined in the DSLNCMT.

When one of these programs gets control, all MERVA ESA resources are directly available, and there is no distinction between direct and central services.

These resources are made available by DSLNUC via addresses in DSLCOM. This allows for loading program parts that also use these services.

Programs linked to DSLNUC use a DSLCOM provided by DSLNUC. The address of the DSLCOM is in general register 12. The application program linked to DSLNUC must not change general register 12 or use it for any other purpose. All fields of DSLCOM are filled by DSLNUC, except the fields COMUSER1, COMUSER2, COMUSER3, and COMUSER4. These are for free use for user-written MERVA ESA applications.

A program linked to DSLNUC can request a MERVA ESA termination by setting the bit COMSTCAN in the DSLCOM.

When a program linked to DSLNUC wants to use the fields of DSLCOM, it must define a DSECT of DSLCOM with the following macro:

```
DSLCOM DSECT=YES,NUC=YES
```

A program linked to DSLNUC can run as a separate task when using parallel processing. The definitions in the nucleus server table DSLNSVT determine whether a program runs under direct control of DSLNUC or as a separate task. Each nucleus server running as a subtask has its own DSLCOM. The DSLCOM of DSLNUC can be accessed via the address in field COMNUCOM.

The calling interface for programs linked to DSLNUC is standardized. Such a program must be defined with LANG=HLL in the DSLNPT or DSLNCM definition. When the program gets control the following information is contained in the general registers.

Register 0: Input/output parameter as described below.

Register 1: Input parameter as described below.

Registers 2 to 11:
Undefined.

Register 12: Address of the DSLCOM area provided for the server.

Register 13: Address of the save area. The program must save the registers according to MVS linkage conventions.

Register 14: Return address.

Register 15: Entry point address of the program.

On return, the general registers must contain the following information:

Register 0: Unchanged, if not otherwise stated.

Registers 1 to 14:
The content must be the same as when control was received.

Register 15: Contains a return code that shows the result of the processing as described below.

Coding an NPT Program (DSLNPT)

Application programs of this type are linked to DSLNUC via DSLNPTT. These programs are event driven; they start processing only when specific events occur (for example, DSLNTS is given control when other MERVA ESA applications request MERVA ESA central services). In the following description, all parameter references refer to the DSLNPT macro defining the discussed program.

The programs named in DSLNPTT are called for initialization, processing, and termination. Each of these steps is discussed in the following.

The following interface description applies to programs defined with LANG=HLL in the DSLNPT. When the program gets control, general register 1 contains the address of the parameter list, which consists of six address fields.

Parameter 1: Reserved.

Parameter 2: Address of the DSLNPT entry of the NPT program. This entry must not be modified.

Parameter 3: Address of a 4-byte field containing the request code. The request codes can be the start request, the stop request, or a processing (event) request. The code for each type is defined in the DSLNPT entry.

Parameter 4: Address of the posted ECB for an event request, or 0 for other requests. This address is filled for an event request only.

Parameter 5: Address of a field to contain the returned ECB list address. This field must be filled by the application when a start request is processed. The area NPTECBA in the DSLNPT entry of the program can be used to return the list of ECB addresses.

Parameter 6: Address of the server table (DSLNSV) entry of the NPT program. This entry must not be modified.

Start Request for an NPT Program

A MERVA ESA **start** command is issued for the program. The request code field contains the value defined for the STRTREQ parameter as defined in the DSLNPT macro for the program. The program acquires main storage, opens data sets, and loads modules, as necessary for its execution.

Because CICS does not provide a fresh copy of the program when it is restarted, you must establish reusability. On return to the caller, the output field that is pointed to by parameter 5 must contain the address of a list of ECB addresses. The maximum number of ECB addresses is determined by the ECB parameter of the DSLNPT macro.

Incorrect ECB addresses can cause an abnormal end of MERVA ESA or give unpredictable results. Therefore, do one of the following:

- Set address fields that are not used to 0.
- Mark the last entry in the address list by setting the high-order bit.

The return code in general register 15 indicates the action to be taken:

0 Program initialization was successful. The program is set active in the DSLNPT entry. The ECB address list returned is saved in the DSLNPT entry and is also added to the multiple wait list of the server.

not 0 The program initialization was not successful. An ECB list is not returned. The status of the program in DSLNPT remains inactive. The program frees the resources acquired up to that point in initialization where the error was met. (The program should have processed its end routines as far as possible before returning indicating an initialization failure.)

Note: No MERVA ESA operator commands must be issued by the program during initialization, because the **start** command is being processed at this time.

Event Request for an NPT Program

One or more of the program's ECBs are posted. The request code field contains the value defined for the ECBREQ parameter as defined in the DSLNPT macro for the program. The program does the processing necessary for this event. The address of the ECB found posted is passed as the fourth parameter. Other ECBs from that program can also be posted. The program determines which event to process first, whether to process more than one event, or whether to ignore events. The time used for processing should be carefully calculated, since this time can be needed by other DSLNPT programs. The post bit in the ECB or ECBs must be cleared to prevent loops. The return code indicates which action should be taken. This return code is saved in the DSLNPT entry and is shown in a display program (**dp**) command:

0 Processing was successful.

4 The request type is invalid. There is a discrepancy between the program coding and the ECBREQ parameter. The program must, however, clear the post bit in its ECBs to prevent a loop. DSLNUC issues message DSL381I.

>4 An error occurred during processing. Any value greater than 4 can be given for error identification. The program must carry out its own termination as it is not called for termination by DSLNUC. If required, the program should take a dump as DSLNUC does not provide a dump. The status of the program in the DSLNPT entry is changed to INACTIVE and DSLNUC issues the DSL382I message. MERVA ESA remains active.

The program can request a MERVA ESA termination by setting the bit COMSTCAN in the DSLCOM.

Stop Request for an NPT Program

A MERVA ESA **stop** command is issued for the program. The program frees all main storage, closes all its open data sets, and deletes all loaded modules to free resources.

When the program gets control for termination, the request code field contains the value defined for the STOPREQ parameter as defined in the DSLNPT macro for the program. The return code indicates which action should be taken. The program

is set to inactive status regardless of the return code, and its ECB addresses are removed from the DSLNUC multiple wait list. If the return code is:

0 Program termination was successful. DSLNCMD issues the DSL061I message.

not 0 Program termination was not successful. DSLNCMD issues the DSL068I message.

Note: No MERVA ESA operator commands must be issued by the program during termination.

Coding a Central Service Program (DSLNTR)

User-written central service programs can be added to MERVA ESA.

A MERVA ESA central service program is a program that is accessed either directly by the programs linked to DSLNUC or indirectly by programs not linked to DSLNUC.

All MERVA ESA central service programs are defined in the MERVA ESA task server request table DSLNTRT.

If a central service program needs to be initialized before being able to execute a service request, or a termination needs to be called to keep data intact for the next startup of MERVA ESA, then the initialization or termination must be done either automatically with the DSLNPT or via the MERVA ESA **start** and **stop** commands. This can be done by defining the central service program in the DSLNPT or another program that calls the central service program for initialization and termination. An example of such a central service program is the authenticator-key file program DWSAUTP. The initialization and termination call is done by DWSAUTIN or by the SWIFT Link service program for the SWIFT network DWSDGPA.

Alternatively, the initialization can be carried out automatically when the first central service request is executed. However, a termination cannot be invoked this way.

In MERVA ESA running under CICS, the programs contained in the DSLNTRT must be reusable as CICS does not provide a fresh copy of the program if it is started again.

When a central service program gets control, it gets the following information in the general registers:

Register 0: If called by a MERVA ESA task server, general register 0 contains the data buffer address.

The data buffer address corresponds to the address of the data delivered with BUF= parameter of the DSLNIC TYPE=REQ macro.

If called directly by a requestor, general register 0 contains the value 0. In this case, the data buffer address is contained in the parameter list of the central service.

Register 1: Parameter list address.

The parameter list address corresponds to the address of the parameter list specified by the PL= parameter of the DSLNIC TYPE=REQ macro.

Both the parameter list and the data buffer follow the rules for MERVA ESA buffers described in "Chapter 2. Buffer Standard of MERVA ESA" on page 3. Both addresses are always available as the MERVA ESA nucleus task server always passes both buffer addresses, even if one of them is not used by the central service program. The buffer length fields in these buffers must be set.

The central service program can work in these two areas only with the restriction of the buffer length in the first length field.

Note: A central service program must support a buffer size that is larger than 32KB, even if the actual data that is moved is smaller than 32KB.

On return, the general registers must contain the following information:

Register 0: Data buffer address, or 0 if the buffer was not used.

If the data buffer address stored in this register is not the same as it was when receiving control, the new address is considered to be that of a new dynamic buffer created by the central service program on behalf of the caller.

Register 15: If called with a buffer address in general register 0, the return code should be passed in the parameter list of the central service only. General register 15 should be set to 0.

If called with a value of 0 in general register 0, the return code of the central service should be passed in the parameter list and in general register 15.

Creating MERVA ESA Operator Commands (DSLNCM)

MERVA ESA, SWIFT Link and Telex Link provide a set of operator commands and associated command execution routines linked to DSLNUC via the operator command table DSLNCMT.

The following chapter is intended for the user who wants to add commands and the appropriate command execution routines to MERVA ESA.

Rules for Defining MERVA ESA Commands

MERVA ESA operator commands are defined in the DSLNCMT command table. These commands are evaluated by DSLNCS. This evaluation includes validity checks of the command, whether the processing module is available, and whether the mandatory parameters have been specified. Command execution routines are linked to DSLNUC via DSLNCMT. Keep the following rules in mind when you define new commands:

- Use the DSLNCM macro to define commands. DSLNCM is described in the *MERVA for ESA Macro Reference*.
- All MERVA ESA commands are checked by the operator-command parsing program DSLNPAR, which allows commands to be abbreviated to a minimum of four characters without special definition.

  Shorter commands are allowed, but abbreviations of less than four characters must be specified separately in DSLNCMT, for example:

```
T for TERMINAT
```

You can define a new abbreviation for an existing command. New abbreviations must be unique and must not conflict with existing commands or abbreviations within DSLNCMT.

If an abbreviation is not unique, it is substituted for the command defined first in the command table.

When abbreviations are used, the DESC= parameter of the DSLNCM macro allows for giving the full command name in the command response.

- The name of the command-execution routine must be specified for each new command in the DSLNCMT command table. This routine can process one or more commands. If several commands are processed by one command-execution routine, the command code for each command must be unique. Several definitions of the command word for the same command should have the same command code.

  The command-execution routine can be a valid external reference in another program (if it shares storage areas with that program), or it can be a separate program.

  All command responses should be defined in the MERVA ESA message table DSLMSGT.

The following is an interface description for command-execution routines for the command tables DSLNCMT.

After command input is processed by DSLNPAR, it is available in the parameter list of DSLNPAR. This parameter list can be mapped using the DSLNPA MF=L macro. The command code is contained in the field NPATOCM, and the parameter tokens are in the field NPATOKS, which follows field NPATOCM. Each token consists of a 1-byte length field that contains the actual parameter length, followed by the token in the length defined in the command table (the length can be 1 to 24 bytes as defined by the DSLNCM macro). If a parameter was omitted, the associated token is filled with blanks.

Numeric parameters are right-justified; other parameters start at the leftmost byte in the token field.

The calling interface for command execution routines linked to DSLNUC is described in the following. This interface is used when LANG=HLL is specified in the DSLNCM macro. LANG=HLL should be specified for all user-written command execution routines linked to DSLNUC. The parameter addresses are passed in a list of addresses pointed to by general register 1. Register 1 points to a list of four fullwords, containing the following addresses:

- Reserved address field pointing to a DSLCOM. The command execution routine should use the DSLCOM address in general register 12.
- Address of the MERVA ESA parsing parameter list, mapped by or acquired with a DSLNPA MF=L macro.
- Address of the MERVA ESA command and response buffer, mapped by or acquired with a DSLNMO MF=BUF macro.
- Address of the 248-bytes continuation information; this area is either set to X'00' when no continuation information is available, or it contains a 2-byte length field, the 8-byte program name, the 2-byte command code, followed by up to 236 bytes of user information from the previous command execution.

A command-execution routine must work in the following way:

- The registers must be saved and registers set up following MVS linkage conventions.
- The command input in the DSLNPA parameter list must be checked for validity; that is, the command must have been entered properly with all its required parameters.
- Command-execution processing must be done and the command response must be built in the DSLNMO command and response buffer. The response-message skeleton can be retrieved from the message table DSLMSGT using the DSLOMS macro that calls the DSLOMSG program. The addresses of DSLOMSG and DSLMSGT are contained in the fields COMOMSGA and COMMSGTA of DSLCOM.

  If command execution involves other programs and other actions, only indicators and data can be prepared for these programs. To transfer control to the next program, an appropriate ECB can be posted, which is contained in the multiple wait list of DSLNUC.

  The continuation information must be returned in the 256-byte area, the address of which was passed as the fourth parameter in the calling parameter list.
- Set the appropriate return code for DSLNCS and return to DSLNCS. A command-execution routine must always return to DSLNCS after a command is executed and the response prepared.

On return to DSLNCS, the general register 15 must contain the return code to indicate what action DSLNCS should take:

0	Command execution has been successful; command and response must be written to the journal and must be returned to the operator or to the program that issued the command.
4	Same as 0, but used to show that no record is to be written to the journal. This can be used for error responses or information messages not requiring journaling.
8	The command is not known to the command-execution routine (for example, an unknown command code), no response is provided, and DSLNCS responds with the message “DSL084I Command not known.”
12	The command parameters were invalid; no response is provided. DSLNCS responds with the message “DSL085I Command parameters invalid.”
>12	Will cause DSLNCS to respond with the message “DSL089I Command execution module return code too high.”

You can find coding examples for the definition of commands in DSLNCMT.

Note: The coding rules of commands in DSLNCMT and in the end-user driver command tables differ slightly.

Adding an Operator Command

To add a new command, the following steps are required:

1. Code a DSLNCM macro in DSLNCMT to define the command.
2. Code and assemble the command-execution routine.
3. Assemble DSLNCMT.
4. Link-edit DSLNUC.

5. Define a command response by coding a DSLMSG macro. This macro can be inserted in a copy member created by the user and included to DSLMSGT via the DSLGEN process or manually.
6. Assemble and link-edit DSLMSGT.

Using the SWIFT Link User Exits

DWSDU021

DWSDU021 is the user exit for the SWIFT network. The material distributed with SWIFT Link contains a sample setup and an explanation of the sample.

When MERVA ESA is customized to run multiple SWIFT Link servers, the user exit must be coded reentrant. The provided sample in the source library is coded reentrant and can be used as a skeleton.

The second calling parameter is a 72-byte save area. If more working storage is needed, the storage must be obtained dynamically.

DWSDU021 is called by DWSDGPAS after a message is completely prepared for sending to the SWIFT network and before the message is:

- Given to DWSNAEVV for sending
- Put to the MERVA ESA journal.

All SWIFT messages including login, select, quit, logout and abort are presented. Depending on the events on the SWIFT line, the message may not be sent to the SWIFT network after having called DWSDU021.

DWSMU126

Refer on page 97 for a description of SWIFT Link user exit DWSMU126.

End of General-use programming interface

Chapter 16. Using the SWIFT Link MAC Authentication Algorithm

General-use programming interface

The MAC authenticator is calculated by calling the program **DWSMAC** with register 1 pointing to the following 5-word parameter list:

1. Address of a 16-byte area containing a full key (see "Padding the Key").
2. Address of a 2-byte area containing the key length, which must always be 16.
3. Address of the first byte of data in the message buffer.
4. Address of a 2-byte area containing the length of the data in the message buffer.
5. Address of the 8-byte area to which DWSMAC is to return the authentication result.

Register 13 must point to a 72-byte save area.

Below is an example of an excerpt from a SWIFT message that shows data in the message buffer. The first byte of data in the message buffer is CRLF, and the data length is 2+9+2+9+2+1=25 bytes.

```
SWIFT =   ...{4:CRLF:20:TRN 1CRLF:20:TRN 2CRLF-}...
                  |                             |
                  |<--------Data Length-------->|
```

Below is an example of calling DWSMAC from an Assembler program:

```
         LA    R1,AUTMACPL             R1 PTS TO PARM LIST
         L     R15,=V(DWSMAC)
         BALR  R14,R15
         .....
AUTMACPL DS     0D                     DWSMAC PARAMETER LIST
         DC     A(AUTMACKY)            ADDR OF KEY
         DC     A(AUTMACKL)            ADDR OF KEY LENGTH
AUTMACDT DS     A                      ADDR OF DATA COMPONENT
         DC     A(AUTMACDL)            ADDR OF DATA LENGTH
         DC     X'80'
         DC     AL3(AUTMACAT)          ADDR OF AUTHENTICATION RESULT
AUTMACKY DS     CL16                   KEY FOR DWSMAC
AUTMACKL DC     H'16'                  KEY IS ALWAYS 16 BYTES
AUTMACDL DS     H                      DATA LENGTH
AUTMACAT DS     CL8                    AUTHENTICATION RESULT
```

Padding the Key

The program **DWSMAC** expects a full 16-byte key. If you are using a key that is not 16 bytes long you must pad it to 16 bytes. The padding routine is described in the S.W.I.F.T. publication **S.S.I. SA2 Algorithm**, and is described in the following:

1. If the number of bytes is lower than 8, add 'F0' until the number of bytes is a power of 2.
2. Repeat the padded key of (1) until the number of bytes is 16.
3. Add **00000000000000000123456789ABCDEF** modulo 16 to the resulting key of (2).

End of General-use programming interface

Appendix A. List of MERVA ESA Tables

Overview of the Base Functions Tables

The Base Functions tables contain the definition of the environment in which MERVA ESA runs. Using tables in this way, Base Functions can be adapted to work in different environments merely by specifying different tables.

Sample tables for an installation are shipped together with the MERVA ESA machine readable material. Some of the tables must be adapted for a particular installation of MERVA ESA and the SWIFT Link (for example, the function table and the routing tables); others need not or should not be changed as they already contain information necessary for the correct processing of MERVA ESA and the SWIFT Link (for example, the message type table and the Field Definition Table contain the information for correct processing of SWIFT messages).

General MERVA ESA Tables

DSLFNTT Function Table

Defines the organization of a bank (MERVA ESA installation) in MERVA ESA terms (queues and functions).

xxxxxxx Routing Tables

The names of the routing tables are defined by the particular installation of MERVA ESA. The routing tables control the flow of messages between the functions (queues) defined in the MERVA ESA function table. All routing tables supplied refer to one of the following:

- Routing of SWIFT messages
- Routing of Telex messages
- Routing of MERVA Link messages.

DSLMPFxx Program Function Key Tables

The Program Function Key Tables define which functions the program functions keys have for users working at screen terminals. The MERVA ESA sample PF Key Table is DSLMPF00. The PF Key Table, ENLMPF00 is supplied for the Telex Link functions.

DSLFLTT File Table for General File Services

Defines the MERVA ESA general files, for example the MERVA ESA Nicknames File, the SWIFT Correspondents File, and the Telex Correspondents File.

DSLTFDT Terminal Feature Definition Table

Defines the page sizes and other features of screen (IMS only) and printer terminals (CICS and IMS).

DSLFDTT Field Definition Table

Defines all fields used in messages (for example, SWIFT messages).

DSLMSGT Message Table

Defines all information and error messages for operators and users.

DSLTXTT Transaction Table

Defines extended information for transaction codes specified in function table entries.

Message Format Service Tables

DSLMPTT MERVA ESA Message Format Service Program Table

Defines all MERVA ESA Message Format Services and program exits available for the formatting of messages.

DSLMTTT MERVA ESA Message Type Table

Connects the symbolic identification of a message with the description of the structure and the fields of a message (Message Control Block).

MERVA ESA End-User Driver Tables

DSLEPTT End-User Driver Program Table

Defines the connection between message processing functions and MERVA ESA function programs.

DSLMCMDT Display and Edit Command Table

Defines the display and edit commands for the MERVA ESA users.

DSLECMDT Session Command Table

Defines the session commands for the MERVA ESA users.

DSLEMCMT Message Selection/Processing Command Table

Defines the message selection and message processing commands for the MERVA ESA users.

DSLEUCMT User File Maintenance Command Table

Defines the user file maintenance commands for the MERVA ESA users.

DSLEFCMT General File Maintenance Command Table

Defines the general file maintenance commands for the MERVA ESA users.

DSLECCMT Queue Utility and Test Command Table

Defines the optional queue utility and test commands for the test period of MERVA ESA.

MERVA ESA Nucleus Tables

DSLNPTT MERVA ESA nucleus program table

Defines all programs linked to the MERVA ESA Nucleus program DSLNUC in a MERVA ESA installation, such as the operating console interface or external network interfaces.

DSLNTRT MERVA ESA task server request table

Defines all MERVA ESA central service programs available in a MERVA ESA installation, such as user file access or MERVA ESA queue management services.

DSLNCMT MERVA ESA operator command table

Defines all MERVA ESA operator commands, the parameters required, and the names of the command execution routine.

DSLNSVT MERVA ESA nucleus server table

Defines the MERVA ESA nucleus servers, and whether these servers run as separate tasks or under direct control of DSLNUC.

Overview of the MERVA-MQI Attachment Tables

DSLKPROC MERVA-MQI Attachment Process Table

Defines the characteristics of the cooperating message transfer between MERVA ESA and the MQSeries.

Overview of the SWIFT Link Tables

The SWIFT Link tables contain the definition of the environment in which the SWIFT Link runs. Using tables in this way, the SWIFT Link can be adapted to work in different environments by specifying different tables. All tables are discussed in this manual. The SWIFT Link tables are:

DWSLTT SWIFT Link Logical Terminal Table

Defines the names of the destinations agreed on with SWIFT for the communication with the SWIFT network.

DWSECMDT Authenticator Key File Maintenance Command Table

Defines the authenticator key file maintenance commands for the SWIFT users.

DWSLINx SWIFT Link Line Definition

Defines a communication line to the SWIFT network.

DWSMCCRT Currency Codes Table

Defines all valid currency codes known to SWIFT and the maximum number of numeric characters allowed in the fraction part of an AMOUNT.

DWSRxxxx Routing Tables

DWSLxxxx Defines the flow of messages between functions (queues) defined for the SWIFT Link in the MERVA ESA function table.

DWSCIT Defines the Central Institutes participating in the SWIFT PREMIUM and FIN-Copy services.

Overview of the Telex Link Tables

ENLRxxxx Routing Tables

Defines the flow of messages between functions (queues) defined for the Telex Link in the MERVA ESA function table.

ENLTKRQT Test-key Requirement Table

Defines the test-key requirements dependent on the message type of the processed telex message.

Overview of the MERVA Link Tables

EKAPT MERVA Link Partner Table

Defines the characteristics of the cooperating message transfer among partner applications.

Appendix B. Cross-References, Macros, and Tables

Base Functions: Macros and Tables	Macro	Table
Name		
Nucleus Program	DSLNPT	DSLNPTT
Task Server Request	DSLNTR	DSLNTRT
Nucleus Command	DSLNCM	DSLNCMT
Nucleus Server	DSLNSV	DSLNSVT
MFS Program	DSLMPT	DSLMPTT
MFS Message Type	DSLMTT	DSLMTTT
Message (Operator and User)	DSLMSG	DSLMSGT
Function	DSLFNT	DSLFNTT
DSLEUD program	DSLEPT	DSLEPTT
File	DSLFLT	DSLFLTT
Field Definition	DSLLFDT DSLLFLD DSLLSUBF	DSLFDTT
Display and Edit Command	DSLNCM	DSLMCMDT
Session Command	DSLNCM	DSLECMDT
Program Function Key	DSLMPFK	DSLMPFxx
Routing	DSLROUTE	.....
Terminal Feature Definition	DSLTFD	DSLTFDT
Transaction	DSLTXT	DSLTXTT

MERVA-MQI Attachment: Macros and Tables	Macro	Table
Name		
Process	DSLKPROC	DSLKPROC

SWIFT Link: Macros and Tables	Macro	Table
Name		
Currency Codes	DWSCUR	DWSMCCRT
Logical Terminal	DWSLT	DWSLTT
Central Institutes	DWSCI	DWSCIT

Telex Link: Macros and Tables	Macro	Table
Name		
Test-Key Requirement	ENLTKREQ	ENLTKRQT

MERVA Link: Macros and Tables	Macro	Table
Name		
Partner	EKAPT	EKAPT

Appendix C. Table of User Exits

The MERVA ESA source library contains the following user exits. More information about the function and the interface can be found in the module header of each user exit.

Table 2. Table of User Exits

Name	Place	Purpose	Parameters
DSLEU001	DSLEUD	Check user ID and password during sign on, for example, with RACF in MVS	DSLCOM, TUCB
DSLEU002	DSLEUD	Check availability of user applications	DSLCOM, TUCB
DSLEU003	DSLEUD	To stop user applications	DSLCOM, TUCB
DSLEU004	DSLEUSR	Check the user ID and password at entry to User File maintenance	DSLUSR parameter list, User File record, password unscramble routine
DSLJR001	DSLJRNP	To support an additional user journal file	Open, close, put (record buffer, journal record)
DSLKQ001	DSLKQS	Provide data for an MQI datagram, request, or reply message	TOF, DSLAPI INTWSTOR, MFS parameter list
DSLKQ002	DSLKQR	Get data from a received MQI datagram, request, or reply message	TOF, DSLAPI INTWSTOR, MFS parameter list
DSLKQ100	DSLKQS DSLKQR	Telex message data in an MQI datagram, request, or reply message	TOF, DSLAPI INTWSTOR, MFS parameter list
DSLMU001	After TOF initialization in MTIN	TOF processing	TOF address
DSLMU003	Entry of DSLMPBLD	TOF preprocessing	TUCB TOF
DSLMU004	Before DSLNPAR in EUD	Command translation	Command from DSLCMDL
DSLMU005	After DSLNPAR in EUD	Command translation	Parsed data
DSLMU006	In DSLMPCMD Help process	TOF preparation for Help	Access to TOF services
DSLMU008	End of DSLMPUTF	TOF processing	TOF
DSLMU009	DSLMCHE	Reason code processing	Reason code
DSLMU010	DSLEMSG	Message completion	DSLCOM, TUCB
DSLMU011	DSLEMSG	After queue access	DSLCOM, TUCB
DSLMU020	DSLSDI	Before put to queue	DSLCOM, TOF, message in line format
DSLMU021	DSLSDY	Message is in TOF, before printing	DSLCOM, TUCB, TOF

Table 2. Table of User Exits (continued)

Name	Place	Purpose	Parameters
DSLMU022	DSLSDO	Message is in TOF, before formatting for sequential data set	DSLCOM, TOF
DSLMU023	DSLCXT	Before checking and expansion	DSLCOM, TUCB, TOF
DSLMU024	DSLHCP	Message is in TOF, before printing	DSLCOM, TUCB, TOF
DSLMU027	DSLHCP	Printer buffer	DSLCOM, TUCB, TOF
DSLMU054	DSLMLFP	Message type determination	Pointer to message
DSLMU090	DSLMMFS	MFS Termination processing	
DSLMU240	DSLCESI	EDIFACT to SWIFT conversion, before SWIFT message put in queue	DSLCOM, TOF, MFS parameter list
DSLMU241	DSLCSEI	SWIFT to EDIFACT conversion, before SWIFT message concatenated to EDIFACT message	DSLCOM, TOF, MFS parameter list
DSLMU242	DSLCES2	EDIFACT to SWIFT conversion, to extract SWIFT field information from EDIFACT message	DSLCOM, MFS parameter list
DSLNU003	DSLNUSR	Check user ID and password during sign-on, for example, with RACF in MVS	User File record, parameter list, password unscramble routine
DSLNU004	DSLNUSR	To support additional User File checking (authorization)	Parameter list, User File records
DSLNU005	DSLNUSR	Check the password of a user (USR function)	Parameter list, User File record, active user entry, password unscramble routine
DSLQKEY	DSLQMGT DSLQMGD	Set the keys for a new message in a queue	Queue parameter list and data buffer (TOF)
DSLQPUT	DSLQMGT	Access every message immediately before it is written to a queue	Queue parameter list and data buffer (TOF)
DSLQTRA	DSLQMGT	Inspect Queue Trace	Queue parameter list and data buffer (TOF) and queue trace buffer.
DSLQTRAB	DSLQMGD	Inspect Queue Trace (QDS on DB2)	Queue parameter list, DSLQMDIO return info, data buffer (TOF) and queue trace buffer.
DSLQUMR	DSLQMGT DSLQMGD	Capture UMR data when it is assigned to a message	Queue parameter list and data buffer (TOF)

Table 2. Table of User Exits (continued)

Name	Place	Purpose	Parameters
DSLWTOEX	DSLNMOP DSLWTOP	Set routing and descriptor codes for WTOs issued by MERVA ESA	Operator message and default routing and descriptor codes
DWSDU021	DWSDGPA	Exit gets control when message is ready to be sent to SWIFT	SWIFT message in TOF and buffer
DWSLOG2	DWSDGPA	Login Authorization for SWIFT network	Logical Terminal Table Entry (Login Sequence Number)
DWSMU126	DWSDGPA	Modify message for SWIFT network	Message in TOF
DWSMU141	DWSMCCUR	Check currency codes	Currency code
ENLMC095	Checking exit sample	Telex character set checking	MFS parameter list
ENLMU398	Sample exit	Automatic Test-key calculation	
EKAME010	EKAAC01 EKAAC02 EKAAC04	Determine ASP list line color, Y2K support	Fields EKADA, EKAPTDT
EKAME011	EKAAC03	Determine SCP list line color, Y2K support	Field EKADA
EKAME012	EKAAC00	Determine CMD response color	Field EKADA
EKAME015	EKADEMO EKAMCTL	Y2K support	Date field
EKAMU000	EKAAS10 EKAAR10	Call FMT/ESA with MERVA Link	EKAXCPL, TOF
EKAMU010	EKAAS10 EKAAR10	Control message processing	EKAXCPL, TOF
EKAMU033	EKAAS10 EKAAR10	Former PS/2 based Telex and USE message processing	EKAXCPL, TOF
EKAMU034	EKAAS10 EKAAR10	Telex message key processing	EKAXCPL, TOF
EKAMU045	EKAAS10 EKAAR10	ASP specific customization of FMT/ESA with MERVA Link	EKAXCPL, TOF
EKAMU133	EKAAS10 EKAAR10	Workstation based Telex and USE message processing	EKAXCPL, TOF

Appendix D. MERVA ESA Sample Programs

The MERVA ESA sample library contains the following types of sample programs:

- Sample MFS exits to demonstrate how to code MFS exits
- Sample MFS exits to perform certain useful functions
- A sample user exit program
- Sample nucleus programs
- Sample API programs
- A sample API application for a CICS online environment
- Sample scenarios for using MERVA Link

The location of the MERVA ESA sample library depends on your operating system:

- In MVS, the MERVA ESA sample library is a partitioned data set with the low level qualifier SDSLSAM0.
- In VSE, the MERVA ESA sample programs are part of the source library.

Sample MFS Exits as Coding Examples

These sample exits are written in Assembler language.

DSLMC899 MFS field checking exit.

DSLMD899 MFS default setting exit.

DSLME899 MFS editing exit. This exit contains an example on how to change screen attributes.

DSLMS899 MFS separation exit.

DSLMU099 MFS user exit with EXEC CICS calls.

Sample MFS Exits to Perform Certain Functions

These sample exits are written in Assembler language.

DSLBM01A MFS User Exit DSLMU009.

This user exit checks that a SWIFT Link authenticator record exists for the correspondent during data entry of SWIFT messages. This exit requires authentication (DWSAUTP) to be running.

DSLBM02A MFS User Exit DSLMU004.

This user exit rejects a user's attempt to authorize a message (that is, to issue the **ok** command) if that user entered or verified the message. This can be done much more easily with the DSLFNT FOUREYE parameter, which is described in the *MERVA for ESA Macro Reference*.

DSLBM03A MFS User Exit DSLMU011.

This user exit skips messages in a queue that a user has already processed. It can stop users from verifying messages that they

entered. This can be done much more easily with the DSLFNT FOUREYE parameter, which is described in the *MERVA for ESA Macro Reference*.

DSLBM04A MFS User Exit DSLMU004.

This user exit is used to segregate commands depending on data in user file. The user data area 1 in a user-file record contains a list of commands with a restricted parameter. Each command is separated by a comma, and there must not be blanks before or after the comma. For example:

```
DQ T1P,DF D1
```

The user data area 1 is displayed on the user file maintenance panel after the **User Data** label. The following table shows the result of each command entered:

Command	Result
DQ	DQ T1P is executed
DQ D1	Error: invalid parameter
DQ T1	Error: invalid parameter
DQ T1P	DQ T1P is executed
DQ T1PR0	DQ T1PR0 is executed
DF	DF D1 is executed
DF D	Error: invalid parameter
DF D1	DF D1 is executed
DF T1	Error: invalid parameter
DF D1PR0	DF D1PR0 is executed

DSLBM05A MFS Retype Edit Exit.

This user exit is a retype edit routine for field SWIHADDR, the correspondent address in a SWIFT message type. If the retyped address is exactly the same as the original address, it is accepted. If the original address has a branch code of XXX, and the retyped address is of length 8, and has the same bank, country, and location code as the original address, then it is accepted. Otherwise the retyped address is rejected.

Note: Subfield SWIHADDR in the DSLFDTT must have EDIT=1.

DSLBM06A MFS User Exit DSLMU009.

This user exit checks the length of a SWIFT message. When the message is too long to be sent to the SWIFT network an error message is issued.

Sample User Exit Program

DSLBN01A is a sample program for DSLNUSR user exit DSLNU003. This user exit shows how to make an expiry date check on the sign-on password.

Sample Nucleus Programs

DSLBN10A A sample central service that can be used to start or stop another nucleus program.

DSLBN11A A sample nucleus program that illustrates how to implement a timer-controlled monitor program. It periodically executes a **start function** command. This command can easily be replaced by other MERVA ESA operator commands.

Sample API Programs

The sample library contains sample programs that illustrate the use of DSLAPI functions. These programs have names of the form **DSLBA*nnx***, where ***nn*** is the sample program number, and ***x*** indicates the programming language:

A Assembler

B COBOL

C C/370

P PL/I

Two types of sample API programs are provided:

- Batch API programs, each of which is provided in a COBOL, PL/I, C/370™, and an Assembler version:

  DSLBA01x CMD MERVA command service

  DSLBA02x Export a SWIFT message from MERVA ESA

  DSLBA03x Import a SWIFT message from MERVA ESA

  DSLBA04x SAVE and REEN services

  DSLBA05x User File services.

- A transaction for automatic start, which is provided in a COBOL, PL/I, and C/370 version:

  DSLBA06x Queue Management services.

For more information, refer to the *MERVA for ESA Application Programming Interface Guide*.

Sample API Application for a CICS Online Environment

There is a sample API application written in COBOL and in PL/I in the sample library. For CICS screen display services, some BMS maps are provided. For more information, refer to the *MERVA for ESA Application Programming Interface Guide*.

Sample Scenarios for Using MERVA Link

These are described in *MERVA for ESA Customization Guide*.

Appendix E. Notices

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

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

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

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

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

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

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

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

IBM Deutschland
Informationssysteme GmbH
Department 3982
Pascalstrasse 100

70569 Stuttgart
Germany

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

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

The following paragraph does apply to the US only.

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

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:

General-use programming interface

Product-Sensitive Programming Interface and Associated Guidance Information...

End of General-use programming interface

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

- *High Level Assembler Language Reference*, SC26-4940
- *IMS/ESA Version 5 Administration Guide: Database Manager*, SC26-8012
- *IMS/ESA Version 5 Application Programming: Transaction Manager*, SC26-8017

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

E

F

G

H

S

T

U

W

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
System Programming Guide
Version 4 Release 1

Publication No. SH12-6366-01

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
SH12-6366-01

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

SH12-6366-01

Cut or Fold
Along Line

Program Number: 5648-B29

SH12-6366-01

Spine information:

MERVA for ESA

System Programming Guide

Version 4
Release 1