z/OS

Distributed FileManager Guide and Reference

Release 1

SC26-7395-00

z/OS

Distributed FileManager Guide and Reference

Release 1

SC26-7395-00

Note

Before using this information and the product it supports, be sure to read the general information under "Notices" on page 139.

First Edition, March 2001

This edition applies to Version 1 Release 1 of z/OS™ (5694-A01) and to all subsequent releases and modifications until otherwise indicated in new editions.

Order publications through your IBM representative or the IBM branch office serving your locality. Publications are not stocked at the address below.

IBM welcomes your comments. A form for readers' comments may be provided at the back of this publication, or you may address your comments to the following address:

International Business Machines Corporation
RCF Processing, Department M86/050
5600 Cottle Road
San Jose, CA 95193-0001
United States of America

IBMLINK from US: STARPUBS at SJEVM5
IBMLINK from Canada: STARPUBS at TORIBM
IBM Mail Exchange: USIB3VVD at IBMMAIL
Internet: starpubs@us.ibm.com
World Wide Web: http://www.storage.ibm.com/software/sms/smshome.htm

If you would like a reply, be sure to include your name, address, telephone number, or FAX number.

Make sure to include the following in your comment or note:
- Title and order number of this book
- Page number or topic related to your comment

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

Contents

Figures

About This Book

This book tells you about Distributed FileManager/MVS (DFM/MVS) a Distributed Data Management Architecture (DDM) server implementation on MVS/ESA or z/OS systems. Distributed FileManager/MVS provides DDM client implementations on heterogeneous systems with remote access to MVS data. Applications can access MVS data independent of where the data is located in a distributed network. Using the Distributed FileManager/MVS server can significantly improve your capability to access MVS data from applications running on client systems.

This book introduces you to Distributed FileManager/MVS and how it works, what its data set requirements are, and how to customize MVS for Distributed FileManager/MVS support. It discusses operating the Distributed FileManager/MVS environment and working with DDM client implementations on OS/2, AIX, and OS/400 systems. It also discusses working with DataAgent through the remote DDM application of SMARTdata UTILITIES (SdU).

Who Should Read This Book

The audience for this book includes people who want to:

- Become acquainted with the features of Distributed FileManager/MVS, its benefits, and how it works.
- Customize MVS for Distributed FileManager/MVS, start up Distributed FileManager/MVS, or monitor and control Distributed FileManager/MVS conversations.
- Find reference information concerning Distributed FileManager (DFM).

What You Should Know Before Reading This Book

To use this book effectively, you should be familiar with the following:

- On MVS/ESA and z/OS systems
  - Characteristics of MVS data sets and access methods
  - Using the Storage Management Subsystem (SMS) to manage MVS data sets
  - Utilities that move or copy MVS data sets
  - Working with the Virtual Telecommunications Access Method (VTAM)
  - Using the Interactive Storage Management Facility (ISMF)
  - Working with Advanced Program-to-Program Communications (APPC)
- Distributed data processing
  - Fundamentals of DDM
  - System Network Architecture (SNA) LU 6.2 protocol for connecting applications
  - Client/server technology
- DDM source implementations (clients)
  - OS/2 2.0 (or higher level) operating system
    - Stream- and record-oriented file access
    - Application programming interfaces (APIs)
    - SMARTdata UTILITIES (SdU)
    - Communications Manager or Communications Manager/2
  - OS/400 operating system
    - File system

- OS/400 DDM
- OS/400 APPC
- OS/400 control language (CL)

What This Book Contains

"Chapter 1. Introduction to Distributed FileManager/MVS" on page 1 discusses the complex demands of today's distributed data processing environment. It introduces you to DFSMSdfp's client/server product, Distributed FileManager/MVS. It describes the features and benefits of Distributed FileManager/MVS and discusses how Distributed FileManager/MVS provides remote access to MVS data. Included are definitions of special terminology used in this book.

"Chapter 2. Accessing Data Sets with Distributed FileManager/MVS" on page 13 describes the data set access capabilities of DFM/MVS. It explains the MVS data set types; the DDM file models; and the record, stream, and directory access functions that are supported by DFM/MVS. It also includes information about wild cards, the VSAM reuse attribute, using PDSEs and PDSs, coded character sets, and DDM file attributes.

"Chapter 3. Customizing MVS for Distributed FileManager/MVS" on page 33 describes how to customize MVS to support the Distributed FileManager/MVS environment. Included are procedures for customizing APPC/MVS, VTAM, Distributed FileManager/MVS, automatic class selection (ACS) routines, and Resource Access Control Facility (RACF, a component of the SecureWay Security Server for z/OS).

"Chapter 4. Operating Distributed FileManager/MVS" on page 51 tells you how to start up, monitor, and control the Distributed FileManager/MVS environment.

"Appendix A. System Samples" on page 55 contains system SAMPLIB samples related to customizing the Distributed FileManager/MVS environment.

"Appendix K. DDM File Attributes" on page 129 contains a table of associated DDM file attributes by DDM file class.

Related Publications

The following publications are helpful for understanding the contents of this book.

Advanced Program-to-Program Communications

Publication Title	Order Number
z/OS MVS Planning: APPC/MVS Management	SA22-7599
Communications: Advanced Program-to-Program Communications Programmer's Guide	SC41-8189

Distributed Data Management Architecture (DDM)

This book discusses only existing DDM source (client) and DDM target (server) implementations. It does not offer information on developing your own custom DDM source implementation. To do this, you must be familiar with DDM and how to program using DDM commands. For more information, refer to the following publications:

Publication Title	Order Number
Distributed Data Management Architecture: General Information	GC21-9527
Distributed Data Management Architecture: Implementation Programmer's Guide	SC21-9529
Distributed Data Management Architecture: Reference	SC21-9526
Application System/400 Distributed Data Management Guide Version 2	SC41-9600
Programming: Control Language Reference	SC41-0030

SMARTdata UTILITIES (SdU) for OS/2 and AIX

Publication Title	Order Number
SMARTdata UTILITIES for OS/2 Bill of Forms	SBOF-6131
SMARTdata UTILITIES for OS/2: VSAM in a Distributed Environment (included in SBOF-6131 and SK2T-2176)	SC26-7063
SMARTdata UTILITIES for OS/2: Data Description and Conversion (included in SBOF-6131 and SK2T-2176)	SC26-7091
SMARTdata UTILITIES for AIX Bill of Forms	SBOF-6132
SMARTdata UTILITIES for AIX: VSAM in a Distributed Environment (included in SBOF-6132 and SK2T-2066)	SC26-7064
SMARTdata UTILITIES for AIX: Data Description and Conversion (included in SBOF-6132 and SK2T-2066)	SC26-7066
SMARTdata UTILITIES: Data Description and Conversion: A Data Language Reference (included in SBOF-6131, SBOF-6132, SK2T-2066, and SK2T-2176)	SC26-7092
SMARTdata UTILITIES: SMARTsort for OS/2 and AIX (included in SBOF-6131, SBOF-6132, SK2T-2066, and SK2T-2176)	SC26-7099
SMARTdata UTILITIES for Windows Distributed FileManager User's Guide	SC26-7134
SMARTdata UTILITIES VSAM Application Programming Interface Reference	SC26-7133

Virtual Telecommunications Access Method (VTAM)

Publication Title	Order Number
VTAM Network Implementation Guide	SC31-6434
VTAM Resource Definition Reference	SC31-6438
VTAM Resource Definition Samples	SC31-6414

Referenced Publications

In this book, references are made to the following publications:

Short Title	Publication Title	Order Number
Character Data Representation Architecture Overview	*Character Data Representation Architecture Overview*	GC09-2207
Character Data Representation Architecture Reference and Registry	*Character Data Representation Architecture Reference and Registry*	SC09-2190

Short Title	Publication Title	Order Number
DDM Architecture: General Information	*Distributed Data Management Architecture: General Information*	GC21-9527
z/OS DFSMS Access Method Services	*z/OS DFSMS Access Method Services*	SC26-7394
z/OS DFSMSdss Storage Administration Guide	*z/OS DFSMSdss Storage Administration Guide*	SC35-0423
z/OS DFSMS: Using Data Sets	*z/OS DFSMS: Using Data Sets*	SC26-7410
z/OS DFSMSdfp Utilities	*z/OS DFSMSdfp Utilities*	SC26-7414
z/OS MVS Planning: APPC/MVS Management	*z/OS MVS Planning: APPC/MVS Management*	SA22-7599
z/OS SecureWay Security Server RACF Security Administrator’s Guide	*z/OS SecureWay Security Server RACF Security Administrator’s Guide*	SA22-7683
VTAM Network Implementation Guide	*VTAM Network Implementation Guide*	SC31-6434
VTAM Resource Definition Reference	*VTAM Resource Definition Reference*	SC31-6438
VTAM Resource Definition Samples	*VTAM Resource Definition Samples*	SC31-6414

Accessing z/OS DFSMS® Books on the Internet

In addition to making softcopy books available on CD-ROM, IBM provides access to unlicensed z/OS softcopy books on the Internet. To find z/OS books on the Internet, first go to the z/OS home page: **http://www.ibm.com/servers/eservers/zseries/zos**

From this Web site, you can link directly to the z/OS softcopy books by selecting the Library icon. You can also link to IBM Direct to order hardcopy books.

Accessing Messages Using LookAt

LookAt is an online facility that allows you to look up explanations for z/OS messages and system abends.

Using LookAt to find information is faster than a conventional search because LookAt goes directly to the explanation.

LookAt can be accessed from the Internet or from a TSO command line.

You can use LookAt on the Internet at:

```
www.ibm.com/servers/eserver/zseries/zos/bkserv/lookat/lookat.html
```

To use LookAt as a TSO command, LookAt must be installed on your host system. You can obtain the LookAt code for TSO from the LookAt Web site by clicking on **News and Help** or from the *z/OS Collection*, SK3T-4269.

To find a message explanation from a TSO command line, simply enter: **lookat** *message-id* as in the following example:

```
lookat iec192i
```

This results in direct access to the message explanation for message IEC192I.

To find a message explanation from the LookAt Web site, simply enter the message ID. You can select the release if needed.

Note: Some messages have information in more than one book. For example, IEC192I has routing and descriptor codes listed in *z/OS MVS Routing and Descriptor Codes*. For such messages, LookAt prompts you to choose which book to open.

How to Send Your Comments

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

- Visit our home page at: http://ibm.com/s390/dfsms/

  There you will find the feedback page where you can enter and submit your comments
- Send your comments by e-mail to:
  - IBMLink™ from US: starpubs@us.ibm.com™
  - IBMLink from Canada: STARPUBS at TORIBM
  - IBM Mail Exchange: USIB3VVD at IBMMAIL
  - Internet: starpubs@us.ibm.com

  Be sure to include the name of the book, the part number of the book, version and product name, and if applicable, the specific location of the text you are commenting on (for example, a page number or a table number).
- Fill out one of the forms at the back of this book and return it by mail or by giving it to an IBM representative. If the form has been removed, address your comments to IBM Corporation, RCF Processing Department G26/050, 5600 Cottle Road, San Jose, California 95193-0001, U.S.A.

Chapter 1. Introduction to Distributed FileManager/MVS

Today's complex data processing environment often requires accessing data that is distributed among many different computer systems. Distributed FileManager/MVS (DFM/MVS) helps solve the problems of distributed data processing. It offers services for accessing and processing MVS™ data from remote computer systems as if the data were local to the remote systems.

The objective of this chapter is for you to understand the concepts of Distributed FileManager/MVS, its benefits, and how it works. To aid your understanding, this chapter includes the following topics:

- Introduction to distributed data processing
- DFSMSdfp distributed processing environment
- Introduction to the Distributed FileManager/MVS environment
- Scenarios for Distributed FileManager/MVS applications

Terminology Used in This Book

Before beginning the discussion, please acquaint yourself with the following terms:

associated DDM attributes
Associated DDM attributes are MVS data set attributes that are defined in DDM. Examples of associated DDM attributes are file size, lock options or end-of-file offset for byte-stream files. Associated DDM attributes are not necessarily exclusive to DDM, but can be common to other applications that access the same data sets.

data set/file
Data set and file are used interchangeably throughout this book. Both terms refer to a named collection of data that is treated as a single unit of data storage and retrieval.

Distributed Data Management Architecture
Distributed Data Management Architecture (DDM) offers a vocabulary and a set of rules for sharing and accessing data among like and unlike computer systems. DDM includes a set of standardized file models for keyed, relative record, sequential, and stream data. It allows users and applications to access data without concern for the location or format of the data.

Distributed FileManager/MVS
Distributed FileManager/MVS is an implementation of target (server) support as defined by DDM. DDM permits systems in an extended enterprise that have DDM source (client) capability to access file data on a DDM target MVS system. See definitions for *source*, *target*, and *extended enterprise*.

extended enterprise
An extended enterprise is a heterogeneous computing environment that often includes both centralized hosts and distributed workstations connected in a network. Gateways within the extended enterprise provide connections to local area networks (LANs). These LANs can serve any computing systems architecture.

local Local is your reference point when discussing such entities as platforms or applications. For example, when discussing network conversations from the reference point of an MVS platform, local refers to entities located on that

MVS system. Similarly, when discussing data access methods from the reference point of an MVS platform, local refers to those access methods. Contrast with *remote*.

partner
Refers to complementary information or function on a remote platform. For example, for Distributed FileManager/MVS to conduct a network conversation requires a local logical unit (LU) on the target MVS/ESA™ or z/OS® system and a partner LU on the source system.

platform
A computer system running a specific operating system connected in a network. For example, OS/2®, OS/400®, and MVS are different operating system platforms.

record-oriented file
File with a record-oriented structure that is accessed record by record. This file structure is typical of data sets on VM, MVS/ESA, z/OS, OS/390, and OS/400 systems. Contrast with *stream-oriented file*.

remote
Remote is relative to your reference point when discussing such entities as platforms or applications. For example, when discussing network conversations from the reference point of an MVS/ESA, z/OS, or OS/390 platform, remote refers to entities that access MVS data across a network. For example, an OS/2 application accessing local MVS data would be remote. Contrast with *local*.

source
Source is the term used in DDM to refer to the platform that originates a request for remote data. Source is also known as client. Source and client are used interchangeably within the scope of this document. Contrast with *target*.

stream-oriented file
File with a byte-oriented structure that is accessed as continuous streams of data bytes. This file structure is common in workstation environments. Contrast with *record-oriented file*.

target
Target is the term used in DDM to refer to the platform that fulfills a request for remote data. Target is also known as server. Target and server are used interchangeably within the scope of this document. Contrast with *source*.

Introduction to Distributed Data Processing

Topics included in this introduction are extended enterprise data access, requirements for accessing remote data in a network, and the client/server perspective.

Extended Enterprise Data Access

The extended enterprise environment depicted in Figure 1 on page 3 represents today's data processing installations. Such an environment often includes both centralized hosts and distributed workstations or hosts connected in a network. Gateways provide connections to LANs. These LANs can serve any computing systems architecture.

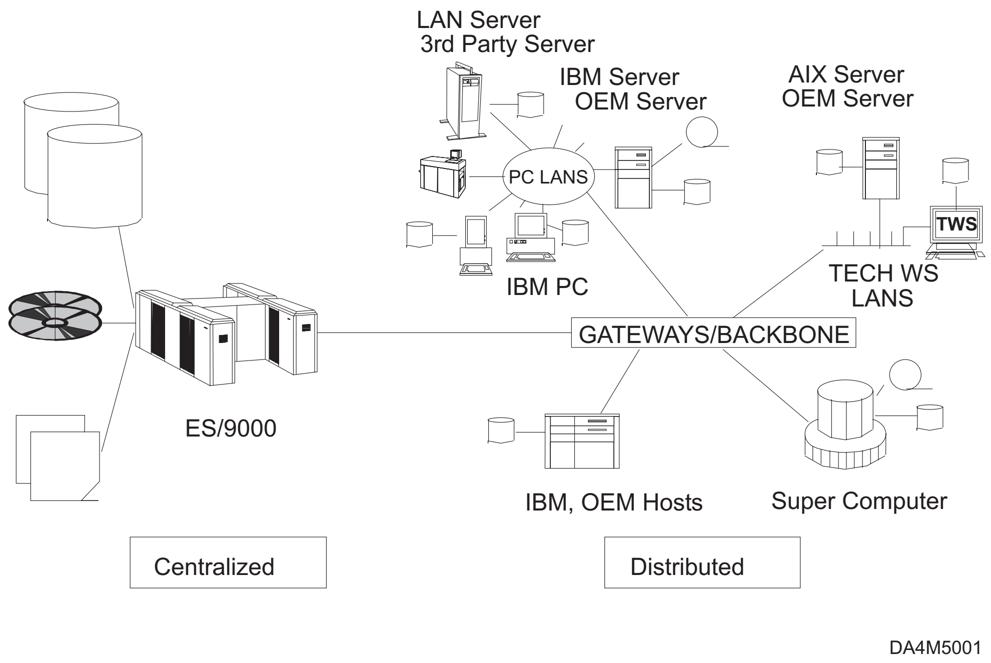


Figure 1. Extended Enterprise Environment

Most applications in an extended enterprise tend to share data to some degree. The trend toward sharing data will grow as workstations become more powerful, networks become more widespread, and applications are written that exploit these capabilities. The evolution of the distributed environment has created the following new requirements for accessing remote data.

Transparent Data Access

Data access should be transparent to applications regardless of the internal format and location of the data. In addition, the data that applications access must at all times be the latest copy. In this discussion, data access implies that applications on source systems can create, read, and write data on target MVS/ESA, z/OS, or OS/390 systems.

Sharing and Accessing Data

Data in an extended enterprise must be in a form that can be shared throughout an enterprise. Multiple workstations must have access to the same MVS data in which the MVS/ESA, z/OS, or OS/390 system provides data sharing and serializing at the data set level.

Avoiding Duplicate Data

In an extended enterprise, uncontrolled data duplication leads to storage management problems and wasting of storage resources. In contrast, controlled duplication for backups and migration is desirable and necessary. Downloading data to the local or LAN environment becomes unnecessary for applications with access to MVS data through Distributed FileManager/MVS services.

Portable Applications

As more and more computing is off-loaded from mainframe systems to workstations, applications should be readily portable to workstations without downloading data. You should be able to access and share data resident on MVS/ESA, z/OS, or OS/390 systems by running applications on workstations.

Applications developed on workstations before they are ported to MVS/ESA, z/OS, or OS/390 systems should also be able to access data on MVS/ESA, z/OS, or OS/390 systems without downloading it to the workstations or to the LAN servers supporting the workstations.

Transparent Applications

Sometimes pertinent data is spread out in an extended enterprise, some of it local to the workstation where the application is running, and the rest of it on a remote MVS/ESA, z/OS, or OS/390 system. If so, a transparent application that runs without modifications allows existing or new applications to access data wherever the data exists without unnecessary data movement. These applications frequently require both record- and stream-oriented data.

Client/Server Perspective

From an architectural point of view, the client or server can be a workstation, a central processor, a local processor or a departmental processor. Generally, a client is best described as a workstation. It is possible, however, for a large host system to be a client that requests data from a small computer such as a workstation.

Usually a server is a central processor, a local system or a departmental system. It is possible, however, for a workstation to be a server that provides data to a central processor. See Figure 2.

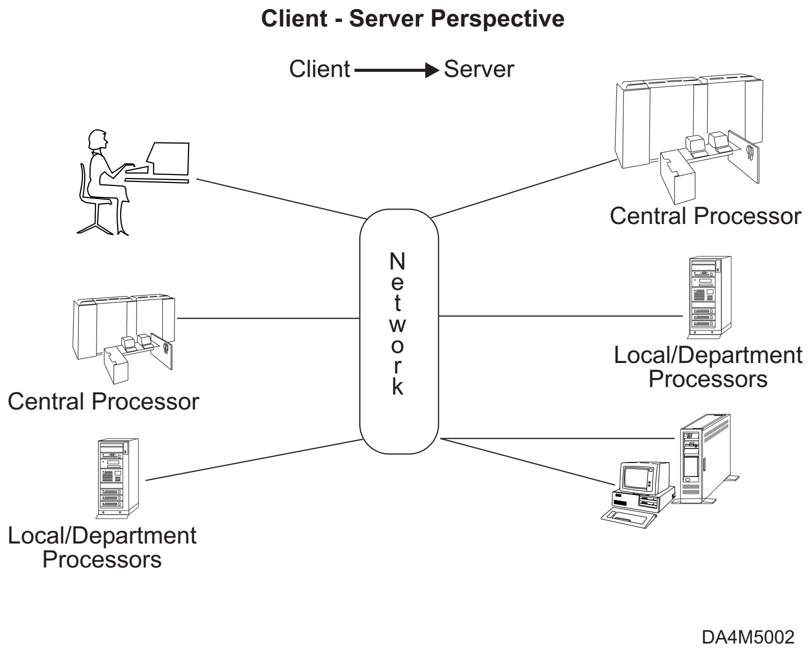

Client - Server Perspective
Client → Server
Central Processor
Network
Local/Department Processors
Central Processor
Local/Department Processors
DA4M5002

Figure 2. Client/Server Cooperative Processing

DFSMSdfp™ Distributed Data Processing Environment

A key objective of DFSMSdfp is to offer products that provide workstations with both record- and stream-oriented data access to MVS data. Workstations accessing MVS data must have the capability of creating, reading, and writing data to the MVS system-managed external storage. Distributed FileManager/MVS is a DFSMSdfp client/server product that enables remote clients in your network to access data on MVS/ESA, z/OS, or OS/390 systems.

Distributed FileManager/MVS

Distributed FileManager/MVS is a DDM server on an MVS/ESA, z/OS, or OS/390 system. DDM enables clients to share and access data on MVS/ESA, z/OS, or OS/390 servers regardless of where the data is located. The benefits of Distributed FileManager/MVS are:

- It provides applications and end-users with transparent access to MVS data from remote platforms
  - Supports both record- and stream-oriented data
  - Gives workstations access to MVS data as if the data were local
  - Allows you to use local commands; no need to use MVS commands
- It improves the productivity of application programmers
  - Can develop high-level language applications independent of data location
  - Eliminates upload and download procedures. Data access is in-place
  - Can share data with other workstations as well as with MVS batch jobs and Time Sharing Option (TSO) users
  - Allows creating, updating, deleting, and renaming of MVS data that is accessed in-place
- It capitalizes on strengths of centralized data storage
  - Offers backup and recovery support across an extended enterprise
  - Allows data to be shared throughout an extended enterprise
  - Ensures security and data integrity using normal MVS conventions
  - Provides latest storage and data management techniques for workstation data
- It leverages existing investments in data, applications, support skills, and storage capacity

Distributed FileManager/MVS uses APPC LU 6.2 protocol to establish network conversations with DDM clients. The conversations consist of DDM commands and messages. DDM is the common language between DDM clients and Distributed FileManager/MVS. DDM client support is currently available on OS/2 and OS/400 systems (see Figure 3 on page 6).

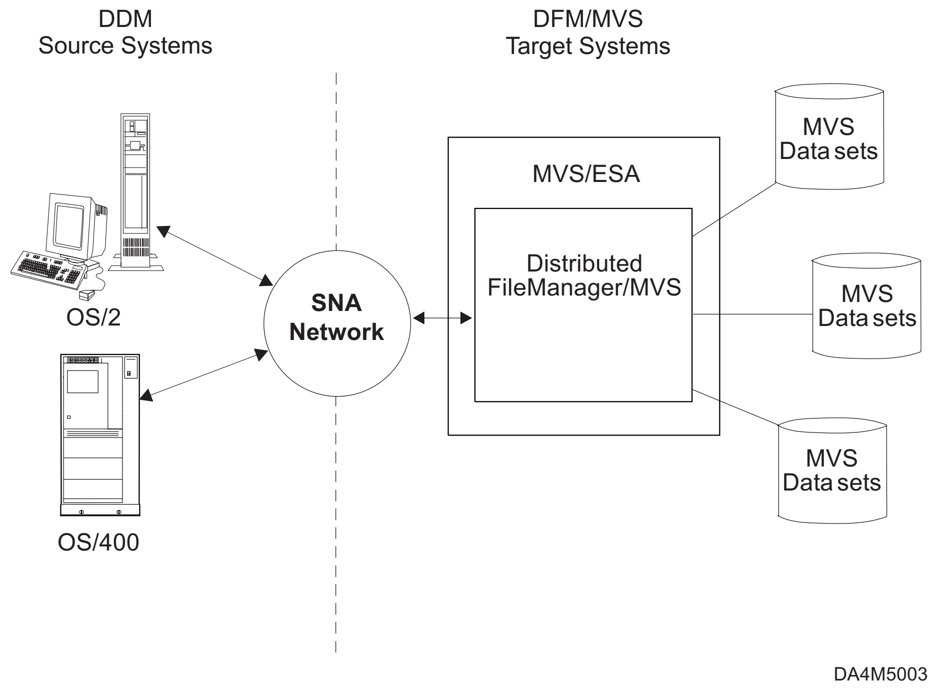


Figure 3. Example of DDM Source—DDM Target Relationships

Introduction to the Distributed FileManager Environment

This discussion includes the following topics:

- Components of the Distributed FileManager/MVS environment
- Platforms that support DDM implementations
- How Distributed FileManager/MVS works
- How Distributed FileManager/MVS DataAgent works

Components of the Distributed FileManager/MVS Environment

The Distributed FileManager/MVS environment requires DDM, DDM source systems, APPC/LU 6.2 protocol, and Resource Access Control Facility (RACF) or an equivalent product.

DDM

DDM implementations use DDM commands as their common language for processing remote data access. DDM provides a vocabulary and set of rules for sharing and accessing data among like and unlike computer platforms. It includes a set of standardized file models and access methods that allows users and applications to access remote data without needing to upload and download files.

Distributed FileManager/MVS is a DDM target implementation providing access and sharing of MVS files to DDM source implementations. DDM source systems use DDM commands to access and share data on DDM target systems. DDM defines the following terms concerning remote file access:

Source of the request
: Initiates requests for data that resides remotely on another system that has DDM target capability.

Target of the request
Processes requests for data initiated by a source system in the network.

For more information about DDM, see *Distributed Data Management Architecture: General Information*

DDM Source Systems

OS/400, OS/2, and Advanced Interactive Executive (AIX) DDM source systems exploit Distributed FileManager/MVS services. OS/400 supports both DDM source and DDM target capabilities. OS/2 and AIX® DDM source systems are available through SMARTdata UTILITIES (SdU) and provide local record file support and DDM source capability. The OS/2 DDM source implementation requires OS/2 2.0 (and higher-level) and SdU.

SdU supports record file access for applications written in high-level languages that include C, PL/I for OS/2, and IBM® Visual COBOL and Visual PL/I for OS/2 and AIX. These high-level languages use the record access feature in a transparent manner so an application can run from the workstation to access remote record data just by recompiling. It supports stream file access for applications written in C. With SdU for OS/2, stream access is also invoked for files or directories that are the target of OS/2 commands issued for the MVS drive.

APPC Communications Protocol

Distributed FileManager/MVS uses APPC LU 6.2 protocol to communicate with DDM source implementations in an extended enterprise network. APPC LU 6.2 protocol is defined by Systems Network Architecture (SNA). It allows systems to communicate on a peer-to-peer basis with other systems in a network.

APPC LU 6.2 support on MVS/ESA, z/OS, and OS/390 systems is provided by APPC/MVS, a part of the base control program (BCP) of MVS/ESA, z/OS, and OS/390 operating systems and Virtual Telecommunications Access Method (VTAM). Distributed FileManager/MVS is conversant in APPC/MVS LU 6.2 protocols and commands. A DDM source implementation is conversant in APPC LU 6.2 protocols and commands.

APPC LU 6.2 on a DDM source system and APPC/MVS LU 6.2 on an MVS/ESA, z/OS, or OS/390 system enable conversations to take place between the DDM source and Distributed FileManager/MVS. These conversations carry DDM commands and messages involved in processing remote access to MVS data.

For more information about APPC, see *z/OS MVS Planning: APPC/MVS Management*.

RACF® Conversation Access Security

RACF, or an equivalent product, is used to control which source systems are authorized to initiate conversations with Distributed FileManager/MVS. By setting up RACF resource class profiles, you can define which user IDs or groups are authorized to access Distributed FileManager/MVS services. You can use RACF resource class profiles to define administrators with update authority to authorize access to Distributed FileManager/MVS.

Once a conversation is initiated, Distributed FileManager/MVS uses RACF services to control the actual data access as well. For more details, see “Using RACF to Control Access to the Distributed FileManager/MVS TP” on page 49.

Platforms That Support DDM Architecture Implementations

DDM source or target implementations are supported on the following IBM platforms:

Platform	Implementation
MVS/ESA	DDM target only
OS/2	DDM source only
AIX	DDM source only
OS/400	Both DDM source and DDM target
4680 Point-of-Sale	DDM target only
CICS®/DDM	DDM target only (MVS and VSE)

How Distributed FileManager/MVS Works

The objective of this discussion is to explain how Distributed FileManager/MVS generally works on an MVS/ESA, z/OS, or OS/390 system and how remote applications access MVS data using Distributed FileManager/MVS. Unless otherwise indicated, you can assume that the DDM source implementation is SMARTdata UTILITIES on an OS/2 2.0 (or higher-level).

Profile of the Distributed FileManager/MVS Environment

Distributed FileManager/MVS enables authorized users and applications on DDM source systems to access MVS data remotely. Applications executing on DDM source systems can access MVS data by exploiting the Distributed FileManager/MVS target support.

Distributed FileManager/MVS participates in APPC LU 6.2 conversations with DDM source implementations. The conversations are exchanges of DDM source commands and DDM target responses. APPC/MVS works with VTAM® to provide the logical connection on MVS for network conversations with source systems. VTAM manages the local logical unit (LU) that forms an LU 6.2 to LU 6.2 link with a partner LU on a remote system.

RACF, or an equivalent product, provides authorization services for controlling access to Distributed FileManager/MVS. Once a conversation is established, RACF also provides authorization services for controlling access to the MVS data.

Distributed FileManager/MVS provides access to MVS data to DDM source implementations as follows (see Figure 4 on page 9):

1. An LU 6.2 to LU 6.2 network link is established between a DDM source and VTAM and APPC/MVS on the DDM target MVS system.
2. The DDM source sends an LU 6.2 allocate request to initiate a conversation with DFM (Distributed FileManager/MVS).
3. The RACF authorizes the DDM source access to DFM.
4. The APPC/MVS scheduler (ASCH), in APPC conversation address space, initiates and schedules DFM (DFM started procedure address space).
5. The DFM, in the DFM central address space, processes the allocate request from the DDM source. It begins a network conversation with the DDM source exchanging DDM commands and messages.

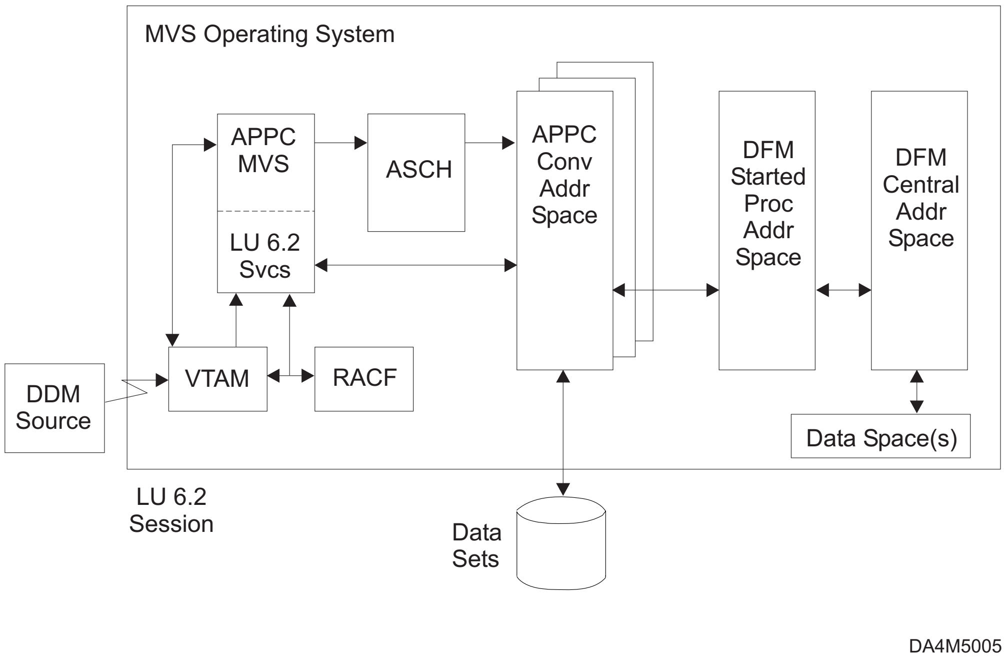


Figure 4. Distributed FileManager/MVS Processing Environment

How DDM Source Systems Communicate with Distributed FileManager/MVS

The communication relationship between a DDM source system and Distributed FileManager/MVS is shown in Figure 5.

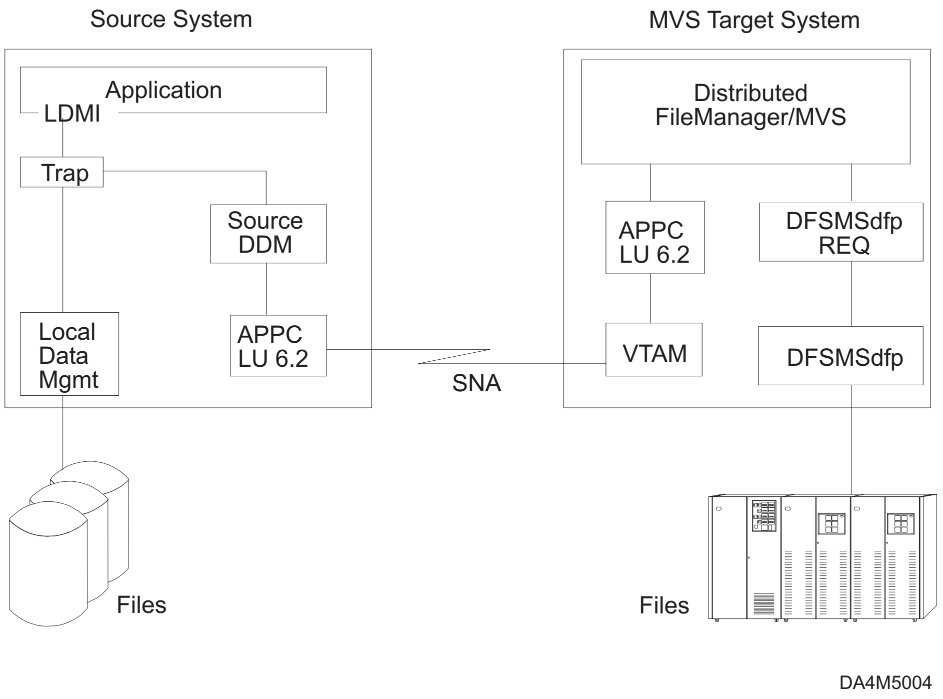


Figure 5. DDM Source System/MVS Target Communication Flow

The DDM source system accesses MVS data using Distributed FileManager/MVS as follows:

1. An application on the source system requests data.
2. The source LDMI (local data management interface) determines if the data is located locally or remotely.
3. If the data is remotely located, the request is turned over to the source DDM (DDM source).
4. An LU 6.2 to LU 6.2 network link is established between APPC LU 6.2 on the source system and VTAM and APPC LU 6.2 on the MVS system.
5. Source DDM sends an allocate request across the network link to Distributed FileManager/MVS.
6. If the source is authorized access, a network conversation begins between source DDM and target Distributed FileManager/MVS that processes remote data access to MVS data.

How Distributed FileManager/MVS DataAgent Works

The Distributed FileManager/MVS DataAgent function allows workstation users to invoke remote procedures that run as extensions or agents of DFM/MVS. This expands the capability of this mode of access by providing access to functions or data sets not ordinarily supported by Distributed FileManager/MVS and by allowing workstations greater control over processing on the MVS server.

Distributed FileManager/MVS DataAgent allows the workstation user of SdU to invoke user-written, IBM-written, or vendor-written Distributed FileManager/MVS DataAgent routines using:

- TSO commands, clists, or REXX execs
- Distributed FileManager/MVS DataAgent routines through the remote file access feature of SdU

The Distributed FileManager/MVS DataAgent is an extension to the Distributed FileManager component of DFSMSdfp and to the remote DDM application of SdU that provides the ability for remote callers to invoke the DFM DataAgent function to execute specified routines on MVS. The functions that may be performed using this facility include the execution of TSO and REXX commands as well as user-written programs. Samples are provided that show specific uses of this function. The DFM DataAgent enhancement represents a significant extension of the functionality of the remote DDM application beyond basic data access.

A sample DataAgent is provided to invoke TSO functions, such as TSO clists, REXX execs, or TSO commands. Another sample DataAgent is provided to invoke SORT.

The following is an example of how the DFM DataAgent expands the capability of the remote DDM application on MVS. Before the DataAgent, the remote application was able to create and delete files, and read, write, update, and delete data contained in files on MVS. With the DataAgent, this capability is broadened significantly by the ability to execute remote procedures on MVS. A DataAgent job could be invoked to preprocess data on MVS by retrieving it from the files or other repositories and place it in the file that the application will access. The DDM application could then process the data in the intermediate file. When the DDM application has finished, a second DataAgent could be invoked to take the data in the intermediate file and distribute the changes to the permanent files.

Scenarios for Distributed FileManager/MVS

Distributed FileManager/MVS offers distributed data processing solutions for a broad range of diverse applications. The following are a few scenarios of the many possibilities:

- Insurance industry
  - Customer-written PC applications can present insurance data to underwriters.
  - OS/2 DDM source systems can connect with Distributed FileManager/MVS on a target MVS system and remotely access and update insurance information.
- Chemical industry
  - Orders can be entered on an OS/400 system.

    Source DDM on an OS/400 can transmit orders to Distributed FileManager/MVS providing access to MVS data sets for centralized tracking.
  - An OS/400 contains personnel data for security guards at one establishment. Daily updates of personnel data can be retrieved from an MVS/ESA, z/OS, or OS/390 system using Distributed FileManager/MVS.
- Banking and finance industries

  Foreign currency transactions on a branch OS/400 can be transmitted to a central MVS/ESA, z/OS, or OS/390 system using Distributed FileManager/MVS.

Chapter 2. Accessing Data Sets with Distributed FileManager/MVS

This chapter describes Distributed FileManager/MVS (DFM/MVS) support for MVS data sets. It explains the types of data sets you can access and describes the record, stream, and directory access functions you can use. It also includes information about data set naming, data set usage, character sets, and file attributes.

Accessing MVS Data Sets

This section introduces the data set access capabilities of DFM/MVS, including:
- DFM/MVS data set requirements
- Data set types supported by DFM/MVS
- File models supported by DFM/MVS
- Default file attributes

Data Set Requirements

DFM/MVS has the following data set requirements:

- New data sets created using DFM/MVS should be SMS-managed, although DFM/MVS supports non-SMS-managed data set creation and access of existing non-SMS-managed data sets.

  Note: The creation of non-SMS-managed data sets is not recommended, because DFM cannot save attributes designed to improve performance or to enhance function. Non-SMS-managed data set creation by DFM/MVS should only be used during the transition period between DFM installation and the implementation of system-managed storage. Once this transition is complete the *UNIT* and *VOLUME* parameters should be removed from DFM00. Refer to “Tuning Distributed FileManager/MVS Startup Parameters in System PARMLIB” on page 43 for additional information on the tunable parameters in DFM00.
- All data sets accessed must be cataloged in an integrated catalog facility catalog.
- All data sets accessed must reside on direct access storage.
- All data sets must be one of the supported types in the next section.

Data Set Types Supported

DFM/MVS supports the following MVS data set types:

- Non-SMS-managed data sets
- Sequential access method (SAM) data sets
  - Basic sequential access method (BSAM) data sets
  - Queued sequential access method (QSAM) data sets
- Virtual Storage Access Method (VSAM) data sets
  - Entry-sequenced data sets (ESDSs)
  - Key-sequenced data sets (KSDSs)
  - Fixed-length relative record data sets (RRDSs)
  - Variable-length relative record data sets (VRRDSs)
  - VSAM alternate indexes to ESDSs or KSDSs
- Basic partitioned access method data sets
  - Partitioned data set extended (PDSE) members
  - Partitioned data set (PDS) members

- Read-only support for PDSE directories
- Read-only support for PDS directories

Data Set Types Not Supported

DFM/MVS does not support:
- VSAM linear data sets (LDSs)
- Generation data groups (GDGs) and generation data sets (GDSs)
- Basic direct access method (BDAM) data sets
- Indexed sequential access method (ISAM) data sets
- Sequential data striping data sets
- OpenEdition® MVS hierarchical file system (HFS) files
- Tape media

File Models Supported

The IBM Distributed Data Management (DDM) architecture helps client applications access server data by defining common data access rules that can be used between different kinds of systems. DFM/MVS supports a set of standardized DDM file models, that allow client applications to use the DDM architecture to access MVS data. One or more DDM file models can be used to access each supported MVS data set listed below:

Record Files

- The DDM direct file model can be used to create and access VSAM RRDSs and VRRDSs.
- The DDM keyed file model can be used to create and access VSAM KSDSs.
- The DDM sequential file model can be used to create and access SAM data sets, VSAM ESDSs, VSAM RRDSs and VRRDSs, PDSE members, and PDS members.
- DFM/MVS supports non-SMS-file creation.

Stream files

- The DDM stream file model can be used to create and access SAM data sets and PDSE members. In addition, DFM/MVS supports stream access to record files.
- DFM/MVS supports non-SMS-file creation.

DFM/MVS also supports the following DDM file models, that have no exact equivalent on MVS:

- A DDM alternate index file (AIF) can be used to create a VSAM alternate index and alternate index path, and to access VSAM data through an alternate index path.
- A DDM directory can be used to access a PDSE directory, a PDS directory, or a directory consisting of all target data sets selected by a source system.

Default File Attributes

Unless otherwise specified by a DDM record access application or changed through the appropriate workstation command, MVS files have the following default values:
- File hidden, system file, and file protect are set to FALSE.
- Get, insert, and modify capabilities are set to TRUE.

Default Delete Capability Attribute

The default delete capability attribute for the following categories is FALSE, unless you take action to change the delete capability attribute to TRUE:

- Files created by DFM/MVS, prior to DFSMS 1.2 with SPE UW90099, have a default delete capability of FALSE.
- All SMS-managed VSAM KSDSs, RRDSs, and VRRDSs, which were not created through DFM/MVS, have a default delete capability of FALSE.

  Prior to the small programming enhancement (SPE), the assumed default delete capability was TRUE.

Changing the Delete Capability Attribute

You cannot issue a DDMOpen with delete access intent against data sets that have a delete capability attribute of FALSE.

To change the delete capability attribute for a file, perform either of the following steps:

- Delete the file, and recreate it through a DDM record access application specifying DELCP in the DDMCreate command.
- Add a DDMSetFileInfo command to the application to set the delete capability as desired.

Distributed FileManager/MVS Access Functions

This section describes the DFM/MVS support for the following categories of files and access functions:

- Record files and record access functions
- Stream files and stream access functions
- Directories and directory access functions

Record Files and Record Access

DFM/MVS supports four DDM record file classes, two sets of DDM access methods for record files, and a complete range of DDM record access functions. The file classes, access methods, and access functions you can use depends on the type of MVS data set you want to access.

Record File Classes

The DDM file classes that you can use to create and access record files are the direct file, keyed file, sequential file, and alternate index file classes. The DDM file classes correspond to the DDM file models explained in “File Models Supported” on page 14.

DDM file classes can be used to access record files as follows:

- SAM data sets, VSAM ESDSs, PDSE members, and PDS members can be accessed through the sequential file class.

  Note: SAM data sets and PDSE members can also be accessed through the stream file class, as explained in “Stream Files and Stream Access” on page 17.
- VSAM RRDSs and VRRDSs can be accessed through the sequential file class or the direct file class.
- VSAM KSDSs can be accessed through the keyed file class.
- A VSAM alternate index and associated alternate index path can be accessed through the alternate index file class.

Access to Record Files

You can use DDM record access methods and DDM keyed access methods to access the following record files.

- Record access methods can be used with these record files: SAM data sets; VSAM ESDSs, RRDSs, and VRRDSs; PDSE members and PDS members.
- Keyed access methods can be used with these record files: VSAM KSDSs and DDM AIFs.
- Record and keyed access methods can *only* be used with record files and they cannot be used to access stream files. The stream access method can be used with stream files and with record files. Stream access to record files is described in “Stream Files and Stream Access” on page 17.

Record Access Functions

The record access functions that can be performed from a remote system, and the data set types that can use with the functions include the following. The list begins with the most restrictive functions that are limited to certain SMS-managed data sets and ends with the most widely available functions that work with any data set supported by DFM/MVS.

Modify Attributes

You can modify DDM attributes associated with the following record files:
- SMS-managed SAM data sets on disk
- SMS-managed VSAM ESDSs, KSDSs, RRDSs, or VRRDSs
- SMS-managed PDSE members

Create File

You can create the following record files:
- Non-SMS-managed data sets
- SMS-managed SAM data sets on disk
- SMS-managed VSAM ESDSs, KSDSs, RRDSs, or VRRDSs
- SMS-managed VSAM alternate indexes and alternate index paths
  - An alternate index and alternate index path are created for you when you create a DDM alternate index file (AIF).
  - The VSAM base data set must meet certain requirements, see “Using VSAM Data Sets” on page 21.
- SMS-managed PDSE members

  A PDSE will be created for you first, if it does not yet exist.
- PDS members, with these limitations:
  - You can only create PDS members if the PDS already exists. If a new data set is required, a PDSE will be created.
  - SMS-managed PDSs are recommended, but you can also create PDS members in a non-SMS-managed data set.
  - PDS members do not support DDM attributes.

Delete File

You can delete the following record files:
- SMS-managed SAM data sets on disk
- SMS-managed VSAM ESDSs, KSDSs, RRDSs, or VRRDSs

  If you delete a VSAM base data set with an alternate index, the alternate index will be deleted for you.
- SMS-managed VSAM alternate indexes and alternate index paths

  These are deleted for you when you delete DDM AIFs.
- SMS-managed PDSE members

Even if you delete the last member of a data set, the data set itself will not be deleted.

- PDS members (whether or not they are SMS-managed)
  Even if you delete the last member of a data set, the data set itself will not be deleted.

Clear File

You can clear the following record files, whether or not they are SMS-managed:
- SAM data sets on disk
- Reusable VSAM ESDSs, KSDSs, RRDSs, or VRRDSs
- PDSE members and PDS members

Other Access Functions

You can use read, write, and positioning functions with the following record files, and you can rename them or retrieve their DDM attributes:
- SAM data sets on disk
- VSAM ESDSs, KSDSs, RRDSs, or VRRDSs (reusable or nonreusable)
- DDM AIFs (and their associated VSAM base data sets)
  DDM attributes are retrieved from the VSAM alternate index or VSAM base data set, depending on the attribute.
- PDSE members and PDS members

Note: PDS members and non-SMS-managed data sets do not support their own DDM attributes. If you retrieve DDM attributes, you will receive default values.

Access Restrictions

When using record files, the following data set access restrictions apply:
- Alias names for PDSE and PDS members are not supported by DFM/MVS. Only the true names can be used to access a file. Load libraries cannot be handled properly due to loss of link edit attributes.
- When accessing multivolume data sets, backward processing and direct positioning is not supported, some forms of insert processing to the end of the file are not supported, and retrieval and update requests do not work if they span physical volumes.

  See “Appendix L. Application Programming Considerations” on page 131 for DDM record access restrictions for multivolume data sets.
- If a local MVS user updates a PDSE member that was created as a sequential file with associated DDM attributes, all the attributes will be lost. Because loss of attributes can cause data conversion and performance problems, local MVS users should avoid updating PDSE members that are accessed by DFM/MVS.
- DFM/MVS cannot create or access SAM data sets or PDSE members with fixed record lengths greater than 32,760 or variable record lengths greater than 32,756. DFM/MVS cannot create or access VSAM data sets with record lengths greater than 32,760.

Stream Files and Stream Access

DFM/MVS supports stream files in SAM data sets or PDSE members. It also supports stream access to record files in additional types of MVS data sets. This section only describes stream access to stream files and stream access to record files. For information about record files and record access, see “Record Files and Record Access” on page 15.

The stream files and access functions you can use from a remote system and the data set types you can use with them, include:

Stream Files

Two types of MVS data sets can be accessed using the DDM stream file class, or file model. You can create, rename, delete, modify DDM attributes, and retrieve DDM attributes for stream files in these data set types:

- SMS-managed SAM data sets on disk
- SMS-managed PDSE members

Stream access is provided on some workstation platforms (currently only OS/2) to allow commands to access remote data transparently. For example, the OS/2 EPM editor can be used to browse or update an MVS file in a transparent manner. In addition, the end user on the workstation can specify *TEXT* on the **DFMDRIVE ASSIGN** or **DFMDRIVE SETPARM** commands to activate stream data conversion and to influence the coded character set identifier (CCSID) used to tag newly created host files. See “Coded Character Set Identifiers” on page 24 for information on CCSID and “Data Conversion” on page 25 for information on the **DFMDRIVE ASSIGN** or **DFMDRIVE SETPARM** commands.

Stream files created while the *TEXT* option is in effect will be converted to the target system code page and tagged with the value specified by HOST_CCSID. If HOST_CCSID is omitted, they will be tagged with the CCSID of the target system. See “Coded Character Set Identifiers” on page 24 for information on supported CCSID code pages. Files for which *BINARY* is specified will be tagged with a CCSID of X’FFFF’.

Legacy data sets tagged with the wrong CCSID can be correctly tagged by running **IDCAMS ALTER**. Otherwise, they will have to be retrieved using the target server defined code page as the PC_CCSID or by using *BINARY* processing. For example, a text file that was stored as binary can be restored by retrieving the file from a drive assigned with the *BINARY* parameter and copying it to a drive with the *TEXT* parameter. The reverse is also true when restoring binary files that were stored as text files.

Files that are not tagged with a HOST_CCSID can be retrieved correctly by starting DFM on MVS with the CCSID parameter in SYS1.PARMLIB(DFM00) set to the CCSID of the file(s) to be retrieved. The DFM00 CCSID value applies to all untagged files for all client access. Alternatively, a HOST_CCSID can be specified on an exception basis by using the workstation HOST_CCSID parameter on the **DFMDRIVE ASSIGN/SETPARM** command. In either case, *TEXT* must be specified to trigger stream data conversion.

Stream Access

You can use the DDM stream access method to read stream data in the following files:

- Record files in nonreusable VSAM ESDSs, and
- Record files in VSAM KSDSs, RRDSs, or VRRDSs.

You can use the DDM stream access method to read, write, or clear stream data in the following files:

- Stream files in SAM data sets on disk
- Stream files in PDSE members
- Record files in SAM data sets on disk
- Record files in *Reusable* VSAM ESDS data sets
- Record files in PDSE members and PDS members

Access Restrictions

When using stream files, the following data set access restrictions apply:

- As a DFM/MVS user, you cannot use record file access methods to access stream files. However, as a local MVS user, you can use standard DFSMSdfp record access methods to access stream files.
- Alias names for PDSE and PDS members are not supported by DFM/MVS. Only the true names can be used to access a file.
- If a local MVS user updates a PDSE member that was created as a stream file with associated DDM attributes, the attribute extension cell containing those attributes is lost. As a result, the PDSE member loses its stream file properties and assumes a file class of sequential file. At the present time, there is no way to correct this situation.

Directories and Directory Access

DFM/MVS supports DDM directories and read-only directory access functions. You can read the directory entries themselves, and their associated DDM attributes, when you list these directories:

- Selected lists of target data sets
- PDSE directories and PDS directories

Note: For record files, the file size shows the number of records in the file. For stream files, the file size shows the number of bytes in the file. (If the file size is unknown to MVS, a size of 0 is shown.)

DFM/MVS does not support the MKDIR command for directories, nor does it support member names that do not comply with MVS PDSE or PDS naming restrictions. For example, a client file named AFILE could be copied to an MVS directory, but AFILE.TXT could not.

Renaming within an MVS directory is possible, but the full MVS path name must be given. For example, you could perform a RENAME with the following command:

```
RENAME IBMUSER.PDSE(A) IBMUSER.PDSE(B)
```

However, the following RENAME would not work:

```
CD IBMUSER.PDSE (or DFMDRIVE ASSIGN IBMUSER.PDSE)
RENAME A B
```

Selected Lists of Target Data Sets

From a remote system, you can use DFM/MVS to list various target data sets as follows:

- You can use a wild card to select a filtered list of target data sets, and you can view the list as if it were a directory. For example, ‘userid.*.PAY’ lists all the source requester’s PAY data sets.
- You can view lists of SAM, VSAM, PDSE, and PDS data sets. A directory can also include files with access restrictions (see “Access Restrictions” on page 20).

PDSE Directories and PDS Directories

- If you select a PDSE or PDS as a directory, you can select file names that have a wild card in the member name. For example, you can select the following file names:

  MEM*, MEM, or null

- If you do not select a directory, you can select file names that have a wild card in the data set name or member name, but not both. For example:
  A.*, A.B, A(MEM*), or A(MEM).
- PDSEs, PDSs, and PDS members do not support their own DDM attributes, so default attributes are displayed.

Access Restrictions

A directory can include hidden files, system files, migrated files, or unsupported files; alias names are not shown. When using directories the following access restrictions apply:

- Hidden or system files
  When you create files using DFM/MVS, you can mark them as hidden files or system files. Later, when you list the directory, you have the option of excluding either of these kinds of files from the list.
- Migrated files
  Directory lists will show default attribute values for migrated files, until they are recalled. When they are recalled, they will show their true attribute values.
- Unsupported files
  The directory lists all files with the names you selected, regardless of whether the data set type is supported by DFM/MVS. If a file is unsupported, it is listed with default attributes. However, you cannot use DFM/MVS to access the file itself.
- Alias names
  Alias names for PDSE and PDS members are not supported by DFM/MVS. Only the true names are shown when you use DFM/MVS to list a PDSE or PDS directory.

Data Set Naming

DDM source applications use a file name parameter to specify target data set names on MVS. If the source file names conform to MVS data set naming conventions, they can also be used as the target data set names. However, if you want to use source file names which cannot be used on MVS, you can implement a name mapping function on the source system.

For example, when you create a SAM data set from OS/2, you can also use the OS/2 naming convention on MVS (an 8-character file name plus a 3-character extension). However, when you create a PDSE member from OS/2, you might need a file name exit on OS/2 to map the OS/2 file names to the MVS data set names. If you are using SMARTdata UTILITIES for OS/2, the Distributed FileManager component provides a user exit that lets you write a file name mapping program.

Wild Cards

A wild card is a character that can be used to represent zero or more characters or qualifiers in a data set name. DFM/MVS supports the use of wild cards with commands that rename or delete files, or that retrieve directory information. For example, you can use wild cards with the DOS commands DELETE, ERASE, or DIR.

The following wild cards are supported by DFM/MVS:

% Represents one and only one character for which a match is not required. For example:
- If the data set name is AB%.XY, then ABA.XY and ABC.XY match.
- If the data set name is A.B(M%), then A.B(MR) and A.B(MS) match.

*	Represents zero or more characters for which a match is not required. For example: • If the data set name is A*.XY, then A.XY and ABC.XY match. • If the data set name is A.B(M*), then A.B(M) and A.B(MEMX) match.
*	Represents zero or more qualifiers for which a match is not required. For example: If the data set name is ABC.*.Z, then ABC.Z and ABC.X.Y.Z match.
*	Represents a member name for which a match is not required. For example: If the data set name is A.B(*), then A.B(MEMX) and A.B(MEMY) match.

Wild Card Restrictions

When using wild cards, the following restrictions apply:

- Wild cards cannot be used to process a group of PDSE or PDS data sets. For example, ABC*(DE) is not allowed. However, wild cards can be used to process a group of PDSE or PDS members, as shown above.
- Only one wild card can be used in each data set name. For example, ABC%E%.XY and AB*.C(E%) are not allowed.
- Wild cards cannot be used in the first character of the data set name. For example, %BCDE.XY and *ABC.D are not allowed.

Using VSAM Data Sets

This section explains how to use alternate indexes to VSAM data sets.

Alternate Index Files

DFM/MVS support for VSAM alternate indexes is provided by a DDM file model called an alternate index file (AIF). A DDM AIF provides access to a VSAM base data set (an ESDS or a KSDS), through a VSAM alternate index path. You can define multiple AIFs over a single VSAM base data set. DFM/MVS will create an AIF with RECSZ(4086 32600). The RECSZ parameter on the chosen DATACLAS will be ignored by IDCAMS.

You can choose a DDM AIF name with a maximum of 40 characters.

Base Data Sets

You can use DFM/MVS to create an alternate index over an ESDS or KSDS base data set. Before you can build an alternate index, the base data set must meet the following requirements:

- DFM/MVS requirements:
  – If the base was created using DFM/MVS, it must be a KSDS.
  – If it was created by a local MVS user, it can be an ESDS or a KSDS.
- VSAM requirements:
  – The base data set must contain records.
  – The base data set must be nonreusable. For more information, see “REUSE Attribute for VSAM Data Sets” on page 22.

Access Restriction

After you create an alternate index file or files, you will have more than one access path to the same VSAM base data set. At that point, you can access the base data set directly or through an alternate index path. However, to avoid locking conflicts, you are advised to use only one access path at a time.

DFM/MVS uses VSAM data definition name (ddname) sharing, VSAM SHAREOPTIONS, and the DFM/MVS lock manager to ensure data integrity. For simultaneous use of more than one access path at a time, consult *z/OS DFSMS: Using Data Sets* for additional information about VSAM sharing.

REUSE Attribute for VSAM Data Sets

This section explains how to use the REUSE attribute for VSAM data sets. VSAM data sets can be marked nonreusable or reusable. A nonreusable data set cannot be reopened as a new data set. A reusable data set can be used as a new data set each time it is opened, as if it were empty.

Nonreusable Attribute

A VSAM data set must be marked nonreusable before you can build an alternate index over it. This VSAM requirement applies whether or not DFM/MVS is used to build the index.

If DFM/MVS is used to build the index, it automatically changes the base ESDS or KSDS to nonreusable, before it creates the alternate index. It is also possible for an MVS user to mark a data set nonreusable with the IDCAMS ALTER command and the NOREUSE parameter.

Reusable Attribute

All VSAM data sets created by DFM/MVS are initially reusable. A VSAM data set must be marked reusable before you can use DFM/MVS to clear any data set or write to a VSAM ESDS with the stream access method.

You can mark a data set reusable using the IDCAMS ALTER command with the REUSE parameter.

Notes:

1. You must be a local MVS user to use the ALTER command. You cannot turn on the reusable attribute using DFM/MVS.
2. If you delete an alternate index file, you must use the ALTER command if you want to make the base reusable again. DFM/MVS does not automatically change the base ESDS or KSDS back to reusable.
3. These restrictions are implemented because it is possible for additional indexes to be defined on MVS, which are not known to DFM/MVS.

Figure 6 on page 23 is an example of an ALTER command that marks a data set reusable.

```
//ALTER    JOB   ...
//STEPA    EXEC  PGM=IDCAMS
//SYSPRINT DD    SYSOUT=A
//SYSIN    DD    *
           ALTER -
              VSAM.DFM.DATASET -
              REUSE
/*
```

Figure 6. ALTER Command that Marks a Data Set Reusable

For more information, see the ALTER command in *z/OS DFSMS Access Method Services*.

Using PDSE and PDS Data Sets

DFM/MVS supports both PDSEs and PDSs. PDSEs are recommended because they have more capabilities than PDSs.

Special PDSE and PDS Processing Considerations

You can only create PDS members as sequential record files using DFM. OS/2 commands create stream files and can be used to create new PDSE members, but cannot be used to create new PDS members.

A DIR command shows PDSEs and PDSs as directories. However, as discussed in "Directories and Directory Access" on page 19, DFM/MVS does not provide full directory support. Also, ambiguities might arise if a PDSE or PDS name matches a prefix name. For example, you may have a PDSE named IBMUSER.DATA and a sequential file named IBMUSER.DATA.SAMFILE.

Therefore, the following rules are provided to help you control PDSE and PDS member access. These examples assume that the user ID is IBMUSER. Note that rules 1 and 2 take precedence over any of the other rules.

1. A **DFMDRIVE ASSIGN** specifying a PDSE or PDS implies that all the subsequent file references for that drive will be to members until a change directory (CD) command is issued, in which case see rule 2 (except that RENAME requires the full MVS path name).
2. A change directory into a PDSE or PDS implies that the file names that follow will be members. For example, CD ″IBMUSER.MYDIR″ implies that a reference to file A will be to member A of PDSE or PDS ″IBMUSER.MYDIR″ (except that RENAME requires the full MVS path name).
3. Explicit usage of parentheses in a fully qualified name implies a member. For example, ″IBMUSER.MYDIR(A)″ refers to member A of the PDSE or PDS ″IBMUSER.MYDIR″.
4. A file name with a ″\″ preceding the last qualifier implies a member. For example, ″IBMUSER.A.B\C″ refers to member C of the PDSE or PDS ″IBMUSER.A.B″ (except that RENAME requires the full MVS path name with parentheses around the member name).
5. A file name with a ″.″ preceding the last qualifier implies a nonmember. For example, ″IBMUSER.A.B.C″ refers to a file named ″IBMUSER.A.B.C″.

Wildcard Processing Exceptions

Note: Wildcard processing does not necessarily follow these rules. For example, COPY S:IMBUSER.PDSE(A*) C:\MYDIR will copy using long filenames IBMUSER.PDSE(A...). Most likely you will want to first issue the command, CD IBMUSER.PDSE, and then the command, COPY A* C:\MYDIR. This will copy using the 1 to 8 character member names only.

Using PDSEs

Some of the advantages to using PDSEs are as follows:

- PDSEs support member-level DDM attributes, whereas member-level attributes do not exist for PDSs.
- PDSE members can contain stream files, whereas PDS members cannot. However, you can use stream access to PDS members that contain record files.
- PDSEs use dynamic space allocation and reclamation, whereas PDSs need to be compressed periodically with the IEBCOPY utility.
- PDSEs are always SMS-managed. PDSs are not necessarily SMS-managed.

Using PDSs

Some of the limitations of using PDSs are as follows:

- PDSs must be compressed periodically using the IEBCOPY utility.
  – Space used by PDS members that are replaced or deleted cannot be reused until the data set is compressed. The more you update a PDS, the more you need to compress it.
  – For more information on IEBCOPY, see *z/OS DFSMSdfp Utilities*.
- You can only create PDS members if the PDS already exists. If it does not exist, a PDSE and PDSE member will be created instead.
- PDS members do not support their own DDM attributes, so DDM default attributes are assumed.

Coded Character Set Identifiers

DFM/MVS supports a DDM attribute called the coded character set identifier (CCSID). The CCSID attribute specifies an identifier registered with the IBM Character Data Representation Architecture (CDRA) of an encoding scheme for coded character set data. The CCSID attribute is a 16-bit number identifying a specific set of encoding scheme identifier, character set identifiers, code page identifiers, and additional coding-related required information that uniquely identifies the coded graphic character representation used. For example, if a file has a CCSID of 437, it is in USA ASCII format. If it has a CCSID of 297, it is in the French EBCDIC format. The meaning of each CCSID is defined in the IBM CDRA. See *Character Data Representation Architecture Reference and Registry* and *Character Data Representation Architecture Overview* for additional information.

All single-byte code page conversions supported by CDRA are supported. DFM/MVS provides built-in support for data conversions between code pages 500 and 850. CDRA needs to be activated for code page conversions that are outside the DFM/MVS built-in support range. The special PC code page values of 0 and 65535 prevent stream data conversion. These values are not valid with the *TEXT* parameter because not providing the PC code page makes it impossible to determine the delimiters that text processing requires. The special host code page value of 65535 (or BINARY) prevents stream data conversion.

Setting the CCSID Attribute

You can set the CCSID attribute locally or remotely. Note that setting the CCSID attribute identifies the character set used by the file, it does not convert the file to that character set.

Setting the CCSID from a Remote System

When you create a new MVS data set from a remote system, DFM/MVS supports the assignment of a CCSID at the time of creation. Also, if an MVS data set already exists, DFM/MVS supports modification of the CCSID from a remote system. You can assign a CCSID to any SMS-managed data set supported by DFM/MVS, except a PDS.

Setting the CCSID from a Local System

If an MVS data set is SMS-managed (and not a PDS), a local system user can run the IDCAMS ALTER command to set or change the CCSID, without using DFM/MVS.

Figure 7 is an example of the command.

```
//ALTER    JOB   ...
//STEPA    EXEC  PGM=IDCAMS
//SYSPRINT DD    SYSOUT=A
//SYSIN    DD    *
           ALTER -
              USER1.DFM.DATASET -
              CCSID(X'01F4')
/*
```

Figure 7. IDCAMS ALTER Command

In Figure 7, the CCSID parameter sets the coded character set identifier to X'01F4'. For more information, see the ALTER command in *z/OS DFSMS Access Method Services*.

Data Conversion

Stream Files: DFM/MVS offers limited support of data conversion for stream files. DFM/MVS APAR OW16828 for DFSMS/MVS® 1.3 provides enhancements to end users who install the new Distributed FileManager for OS/2 (DFM/2) enhancements, as described in informational APAR II09011. These OS/2 end users can now retrieve MVS stream data and have it converted to the single-byte code page associated with their workstation.

Using the DFM/2 commands DFMDRIVE SETPARM and DFMDRIVE ASSIGN, the OS/2 end user can specify MVS target parameters that will trigger stream data conversion. The MVS target parameters are as follows:

BINARY
Specifies no stream data conversion. *BINARY* is the default.

TEXT
Specifies stream data conversion and tags new stream file with the workstation CCSID (PP_CCSID). The current workstation CCSID is automatically passed in by DFM/2, but may be overridden by the PC_CCSID parameter. When *TEXT* is specified, the following parameters are also valid:

- *CRLF* maintains record boundaries using carriage return and line feed as the delimiters. *CRLF* is the default.

- *NL* maintains record boundaries using the new line character as the delimiter.
- *LF* maintains record boundaries using the line feed character as the delimiter.
- *NOEOL* does not maintain record boundaries and treats any delimiters or padding as data.

PC_CCSID=*ddddd*
Specifies the workstation CCSID, *ddddd* is the decimal CCSID. The CCSID is ignored for *BINARY* processing and new files are tagged with a CCSID of X'FFFF' to indicate they are not converted. The PC_CCSID setting does not affect the retrieval of binary files, only the *TEXT* parameter triggers data conversion.

HOST_CCSID=*ddddd*
Specifies the CCSID used for stream files created on the target system or for legacy data sets with no explicit CCSID defined. *ddddd* is a decimal CCSID from 0 to 65535. A CCSID value of 65535 prevents steam data conversion. If omitted, text files are created or retrieved using the current CCSID from SYS1.PARMLIB(DFM00).

The *TEXT* parameter triggers stream data conversion when required and when the combination of CCSIDs is supported by CDRA/MVS.

If the file is not tagged with a CCSID and *TEXT* processing is specified by the workstation, legacy files not tagged with a specific CCSID will default to the CCSID as specified in SYS1.PARMLIB(DFM00).

The HOST_CCSID parameter is not used to override an explicit CCSID associated with a file. It is only used to tag new files or to access files that have no CCSID set.

Record Files

DFM/MVS does not provide data conversion services for record files. When DFM/MVS stores record files on MVS, the data is stored in the format sent by the source. If a target system application requires a different data format, data conversion can be done by a source application.

For example, if data is sent from an OS/2 source system in ASCII format, it requires conversion to EBCDIC before it can be read by a standard MVS application. This conversion from ASCII to EBCDIC can be done by the conversion utility provided by the Distributed FileManager component of SMARTdata UTILITIES.

Associated DDM Attributes

Associated DDM attributes are MVS data set attributes that are defined in DDM architecture. Examples of associated DDM attributes are file size, lock options, or end-of-file offset for byte-stream files. Associated DDM attributes are not exclusive to DDM, but can be common to other applications that access the same data sets.

DFM/MVS creates associated DDM attributes when it creates new data sets or changes the attributes of existing data sets. When copying, moving, or backing up data sets that have associated DDM attributes, it is important that you use recommended data moving utilities (see "Propagating DDM Attributes" on page 28).

The remainder of this section explains the applications and commands you can use to determine if an MVS data set has associated DDM attributes, and the utilities you can use to propagate associated DDM attributes.

Listing DDM Attributes

You can use one of the following tools to determine whether a data set has associated DDM attributes:

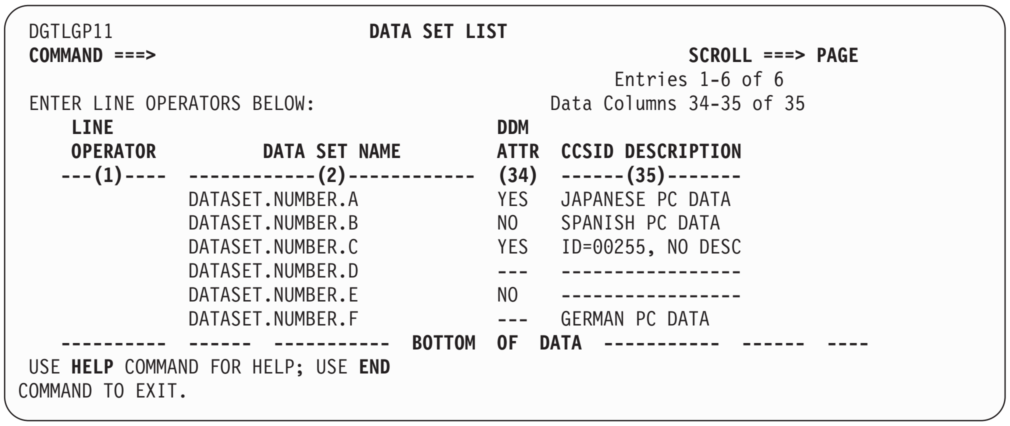
- ISMF data set list application
- IDCAMS DCOLLECT command
- IDCAMS LISTCAT command

However, they cannot determine which specific DDM attributes are associated with an MVS data set, nor the values of the DDM attributes (except for the CCSID attribute).

Using the ISMF Data Set List

With Interactive Storage Management Facility (ISMF), you can use the data set list application to determine whether SAM or VSAM data sets have associated DDM attributes, and the value of the CCSID attribute.

Using the FILTER, LIST, SORT, or VIEW command, select specified data sets and sort on the DDMATTR field in column 34 and the CCSID DESCRIPTION field in column 35. The DDMATTR field indicates whether or not a data set has DDM attributes and the CCSID DESCRIPTION field gives the value of the CCSID. Figure 8 is an example of the resulting output.



```
 DGTLGP11                       DATA SET LIST
 COMMAND ===>                                                    SCROLL ===> PAGE
                                                         Entries 1-6 of 6
 ENTER LINE OPERATORS BELOW:                          Data Columns 34-35 of 35
     LINE                                         DDM
     OPERATOR          DATA SET NAME              ATTR  CCSID DESCRIPTION
    ---(1)----  ------------(2)------------       (34)  ------(35)-------
                DATASET.NUMBER.A                  YES   JAPANESE PC DATA
                DATASET.NUMBER.B                  NO    SPANISH PC DATA
                DATASET.NUMBER.C                  YES   ID=00255, NO DESC
                DATASET.NUMBER.D                  ---   -----------------
                DATASET.NUMBER.E                  NO    -----------------
                DATASET.NUMBER.F                  ---   GERMAN PC DATA
    ----------  ------  -----------  BOTTOM  OF  DATA  -----------  ------  ----
 USE HELP COMMAND FOR HELP; USE END
COMMAND TO EXIT.
```

Figure 8. ISMF Data Set List Columns 34-35

For more details, see *z/OS DFSMS: Using the Interactive Storage Management Facility*.

Using the IDCAMS DCOLLECT Command

You can use the IDCAMS DCOLLECT command to determine if SMS-managed data sets have associated DDM attributes and the value of the CCSID attribute. In the DCOLLECT command output, the DCDDDMEX flag indicates if a data set has associated DDM attributes and the DCDCCSID field contains the value of the CCSID attribute.

Using the IDCAMS LISTCAT Command

You can use the IDCAMS LISTCAT command to determine if SAM or VSAM data sets have associated DDM attributes and the value of the CCSID attribute. Figure 9 on page 28 uses the LISTCAT command to generate a report on a data set named IBMUSER.DFMDATA:

```
//LISTCAT  JOB
//STEP     EXEC PGM=IDCAMS
//*************************************************************
//* PURPOSE: LIST A CATALOG AND A CLUSTER
//**************************************************************
//SYSPRINT   DD   SYSOUT=*
//AMSDUMP    DD   SYSOUT=*
//SYSIN      DD   *
    LISTCAT LVL(IBMUSER.DFMDATA) ALL
 /*
```

Figure 9. LISTCAT Command

Figure 10 shows the resulting output. The DDMEXIST field contains the value TEXT, indicating that associated DDM attributes exist. And the CCSID field contains the value X’01F4’, NLS EBCDIC STANDARD.

```
IDCAMS  SYSTEM SERVICES                                           TIME:08:10       12/02/92
PAGE   1
NONVSAM ------- IBMUSER.DFMDATA.TEST
     IN-CAT --- SYS1.ICFCAT.VSYS306
     HISTORY
       DATASET-OWNER-----(NULL)     CREATION--------1990.016
       RELEASE----------------2     EXPIRATION------0000.000
     SMSDATA
       STORAGECLASS -----NORMAL     MANAGEMENTCLASS--PRIMARY
       DATACLASS --------(NULL)     LBACKUP ---1992.296.0129
     VOLUMES
       VOLSER------------SYS309     DEVTYPE------X'3010200E'     FSEQN------------------0
     ASSOCIATIONS--------(NULL)
     ATTRIBUTES
       STRIPE-COUNT------(NULL)     CCSID------------X'01F4', NLS EBCDIC STANDARD
       DDMEXIST            TEXT
IDCAMS  SYSTEM SERVICES                                           TIME:08:10       12/02/92
PAGE   2
         THE NUMBER OF ENTRIES PROCESSED WAS:
                   AIX -------------------0
                   ALIAS -----------------0
                   CLUSTER ---------------0
                   DATA ------------------0
                   GDG -------------------0
                   INDEX -----------------0
                   NONVSAM ---------------1
                   PAGESPACE -------------0
                   PATH ------------------0
                   SPACE -----------------0
                   USERCATALOG -----------0
                   TAPELIBRARY -----------0
                   TAPEVOLUME ------------0
                   TOTAL -----------------1
         THE NUMBER OF PROTECTED ENTRIES SUPPRESSED WAS 0
IDC0001I FUNCTION COMPLETED, HIGHEST CONDITION CODE WAS 0
```

Figure 10. IDCAMS LISTCAT Output Showing DDMEXIST and CCSID Fields

Propagating DDM Attributes

To reliably propagate DDM attributes when moving their associated files, you must use recommended data movers.

For more information on IMPORT and EXPORT, see *z/OS DFSMS Access Method Services*. For more information on DFSMSdss, see *z/OS DFSMSdss Storage Administration Guide*. For more information on IEBCOPY, see *z/OS DFSMSdfp Utilities*.

SAM and VSAM Data Sets

You can use the IDCAMS IMPORT and EXPORT commands to copy or move SAM and VSAM data sets. You can use the DFSMSdss data mover to back up, retrieve, or migrate SAM and VSAM data sets. You must use the DFSMSdss data mover if the data sets are managed by DFSMShsm.

These data movers automatically propagate associated DDM attributes when moving data sets to other volumes or other systems. DDM attributes associated with SAM and VSAM data sets are not propagated in the following situations:

- If you move data sets to a system that does not support DFM/MVS
- If you use IDCAMS IMPORT with the INTOEMPTY parameter
- If you use the IDCAMS REPRO command

PDSE Members

You can use the IEBCOPY utility or the DFSMSdss data mover to copy, move, or back up PDSE members. With IEBCOPY you can create unloaded copies directly to tape or disk. You must use the DFSMSdss data mover if the data sets are managed by DFSMShsm. These data movers automatically propagate DDM attributes associated with PDSE members. DDM attributes associated with PDSE members are not propagated in the following situations:

- If you copy or move individual records from one member to another
- If the input data set does not completely replace the output data set
- If you move a PDSE member to a system that does not support DFM/MVS
- If you convert a PDSE to a PDS
- If you load an unloaded PDSE to a PDS
- If you copy or move a PDSE member to a PDS

If you move or copy a PDS member or a PDS data set to a PDSE, default DDM attributes will be assigned to the resulting PDSE members.

Accessing Data Using the DataAgent Parameters

This section describes accessing data using the DataAgent parameters. A DataAgent can only be started from a DDM application from a client workstation. You can use the DDMOpen function to get a filename suffix that can be used to start DFM DataAgent processing on MVS. The DDMClose function terminates DataAgent processing. It issues the DDM commands CLOSE and DELDCL, which actually terminates agent processing by invoking the exit with the DELDCL code point in the parameter list, if requested.

Using the DFM DataAgent Filename Suffix Parameters

The DFM DataAgent filename suffix parameters supported by MVS are the following:

Using the AGENT(agent_name<,procedure_parameter>)

This parameter specifies the name of the agent that is invoked when a file is declared (at DDMOpen) and, optionally, when the file declaration is deleted (at DDMClose). The agent_name specifies the name of a member that must exist in SYS1.PROCLIB. Parameters can optionally be provided for symbolic substitution in the PROCLIB member.

Allocation will run under the authorization assigned to started tasks. The agent (running under the user’s authorization) may have to use dynamic allocation to allocate files that cannot be allocated by started tasks.

Because agents begin as started tasks, unless DFM00 specifies RESTRICT_START(NO), the first 3 characters of the procedure (or agent) must be ″DFM.″

The maximum length of the agent name and its parameters is 107 bytes. Each parameter in the list of procedures is subject to the MVS limit of 44 bytes.

The agent runs synchronously. If the PROCLIB member has multiple steps, any file name changes or return code settings will be propagated to the later steps and will only be returned to DFM after the last step has executed.

Using the PARM(agent_parameter_list)

This parameter is used to pass parameters to the agent routine when it begins to execute in the DataAgent address space. Parameters are converted to upper case and concatenated to any parameters provided in the PROCLIB member.

The maximum length of the parameters in bytes is limited only by the space available to the filename suffix.

If PARM is specified, the PROCLIB member must contain the JCL statements as provided by the sample DFMX001 that specifies the DFMINIT parameter so as to run a DFM DataAgent address space initialization routine as the first program in the address space. This causes DFM/MVS to pass a supplementary run time parameter list to the DataAgent routine and allows the routine to return an error code and additional reason codes to DFM. The agent parameter list specified is concatenated with DFMINIT before the DataAgent routine invoked.

This parameter is ignored if AGENT is omitted.

Using the PGM(program_name)

This parameter specifies the name of the program (DataAgent routine) to be invoked by DFM after initialization. If omitted, the program invoked will default to the agent name requiring that you have identically named MVS load modules and PROCLIB members.

This parameter is ignored if AGENT is omitted.

Using the START(job_name<,job_parameters>)

This parameter specifies the name of the PROCLIB member representing a job or procedure to be started asynchronously. Unless DFM00 specifies RESTRICT_START(NO), the first 3 characters of the procedure or command must be ″DFM.″

Optional parameters can also be provided. The MVS limit for the total length of the job name and its parameters is 124 bytes.

DFM will verify that an address space for running the procedure was created, but will not verify that the procedure exists or that it ever completes successfully. That is, the started job runs asynchronously.

It is possible to run some existing PROCLIB members that may not have particular initialization requirements by using only the AGENT keyword. However, it is not recommended because return codes will not be passed back to DFM. It is expected that there will usually be a need for extended parameter passing. The AGENT parameter should be used in conjunction with the PARM and PGM parameters even if the PARM parameter is the null value of PARM() or the PGM name is the same as the agent name.

DFM/MVS imposes a limit of 256 bytes for the file name and file name suffix and for the total length of the parameters (AGENT, PARM, PC_CCISD, START, and so on) that can be passed.

As with the other filename suffix parameters, unidentified or misspelled keywords are ignored and the first (leftmost) is used in case of duplicate keywords.

Chapter 3. Customizing MVS for Distributed FileManager/MVS

This chapter is about customizing MVS for Distributed FileManager/MVS (DFM/MVS). It discusses how to enable Distributed FileManager/MVS to function in a network as a DDM target (server) providing remote access to MVS data sets for DDM source implementations (clients). Distributed FileManager/MVS does not support DDM source capability.

What Is In This Chapter?

Customizing MVS for Distributed FileManager/MVS includes several tasks. These tasks involve establishing APPC/MVS, VTAM, Distributed FileManager/MVS, and other system information so that Distributed FileManager/MVS can provide remote access to MVS data sets. The objective of this chapter is for you to understand the tasks involved and how they are interrelated.

Summary of Customizing Tasks

- APPC/MVS customizing tasks
  - Defining PARMLIB start parameters for APPC/MVS
  - Creating the APPC/MVS transaction program (TP) profile data set (if not already existent)
  - Adding Distributed FileManager/MVS TP profile information to the TP profile data set
  - Creating the APPC/MVS side information data set
  - Defining PARMLIB start parameters for the APPC/MVS scheduler
- VTAM customizing tasks
  - Defining the local LU to VTAMLST
  - Defining the logon mode table in VTAMLIB
  - Defining the local LU and logon mode on a partner system
- Distributed FileManager/MVS customizing tasks
  - Installing PARMLIB start parameters for Distributed FileManager/MVS
  - Activating the PROCLIB startup procedure for Distributed FileManager/MVS
  - Verifying program property table (PPT) entries for Distributed FileManager/MVS
- Setting up automatic class selection (ACS) routines
- Defining TP access security

Interrelationships of Customizing Tasks

Figure 11 on page 35 shows some of the interrelationships among tasks involved in customizing MVS for Distributed FileManager/MVS. Each numbered box in the figure represents a task. To simplify the diagram, only sample parameters that show relationships are shown. The lines and arrows between boxes show relationships between parameters, members, or data set names.

1 Adding the Distributed FileManager/MVS TP profile to the TP profile data set

2 Creating the APPC/MVS side information data

3 Defining a local LU and logon mode table to VTAMLST

4 Defining logon mode table to VTAMLIB

5 Starting up VTAMLST and VTAMLIB

6 Startup procedure for Distributed FileManager/MVS

7 Defining startup parameters for APPC/MVS

8 Defining startup parameters for the APPC/MVS transaction scheduler

9 Tunable startup parameters for Distributed FileManager/MVS

10 Defining partner OS/2 local LU and logon mode information

11 Operator command for starting up APPC/MVS

12 Operator command for starting up the APPC/MVS transaction scheduler

13 Operator command for starting up Distributed FileManager/MVS

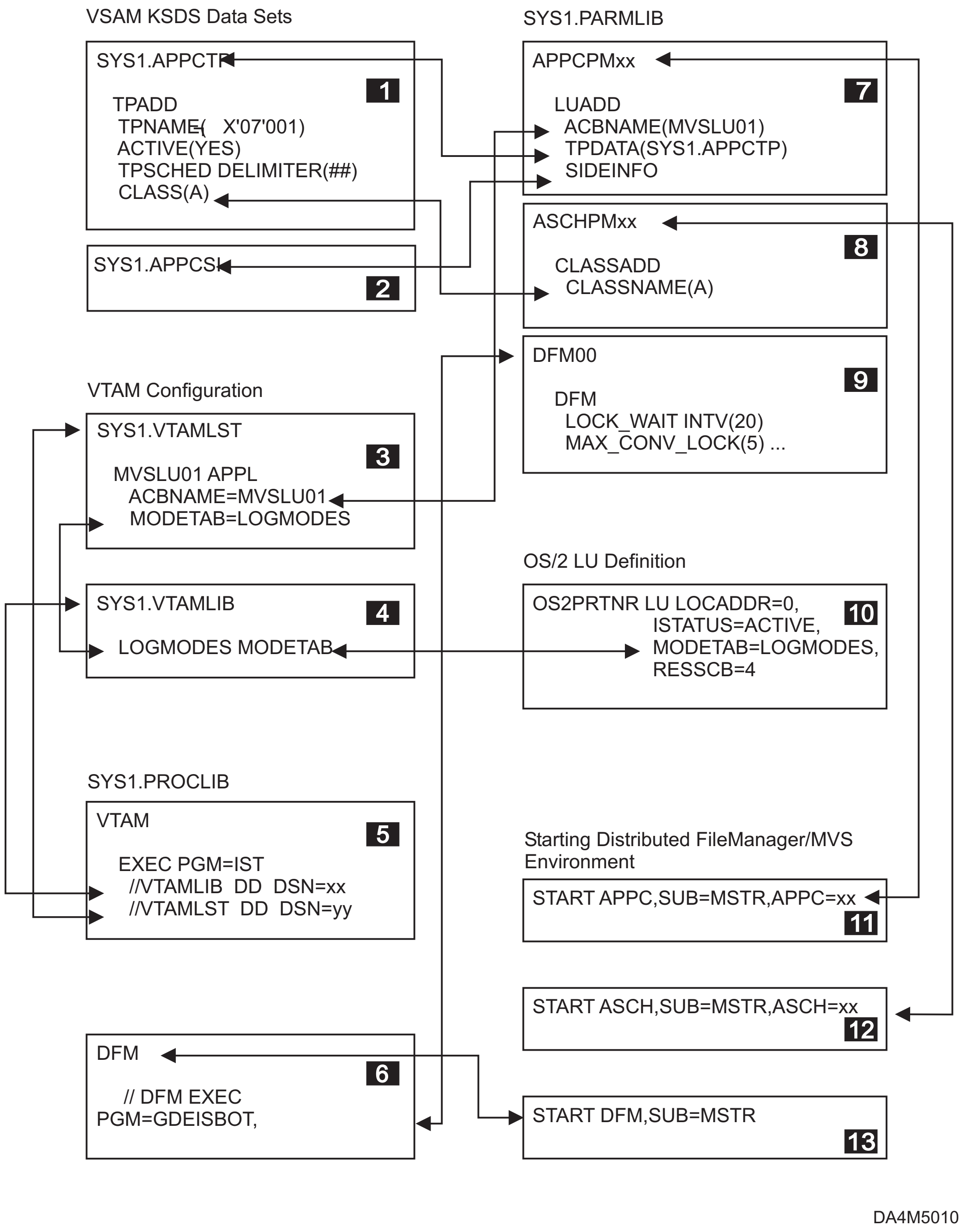


Figure 11. Interrelationships of Customizing Tasks for Distributed FileManager/MVS

APPC/MVS Customizing Tasks

APPC/MVS customizing tasks include:

- Defining PARMLIB start parameters for APPC/MVS
- Creating the APPC/MVS TP profile data set (if not already existent)
- Creating the Distributed FileManager/MVS TP profile
- Creating the APPC/MVS side information data set
- Defining PARMLIB start parameters for the APPC/MVS transaction scheduler

Defining PARMLIB Start Parameters for APPC/MVS

You define APPC/MVS start parameters in system PARMLIB member APPCPMxx (for example, SYS1.PARMLIB(APPCPMxx)). The APPC/MVS start parameters contain information for establishing and controlling APPC conversations on the system. They identify the local LU to be used for APPC conversations, and the TP profile and side information data sets to be used by APPC/MVS.

Distributed FileManager/MVS conversations flow over the LU defined as the base LU in APPCPMxx. The TP profile data set provides APPC/MVS with the Distributed FileManager/MVS TP profile information. The TP profile information enables Distributed FileManager/MVS to participate in APPC LU 6.2 conversations.

You can control APPC/MVS start parameters by using different versions of APPCPMxx. Each version can have different values for the start parameters. For example, one version can omit an LU name that is included in another version.

APPC/MVS is started by a system operator command. The operator command (which can be part of initial program load) identifies APPCPMxx (see “Starting Up APPC/MVS” on page 51). Both APPC/MVS and the APPC/MVS transaction scheduler (also started by operator command) must be active before Distributed FileManager/MVS LU 6.2 conversations can take place.

Using the APPC/MVS LUADD Definition

Use the APPC/MVS LUADD definition to define the start parameters in APPCPMxx. With LUADD parameters, you can add, modify, and delete LU information. You can also change the defined names for both the TP profile and the side information data sets.

Figure 12 is an example of the basic LUADD definition that should be included in APPCPMxx. This example can be found in system SAMPLIB member GDEAPPC (for example, SYS1.SAMPLIB(GDEAPPC)), or see “GDEAPPC” on page 55. Also see system SAMPLIB members APPCPMRX and APPCPMXX for more details.

```
LUADD
   ACBNAME(MVSLU01 )             ◄── LU name
   BASE
   TPDATA(SYS1.APPCTP)           ◄── APPC/MVS VSAM TP data set
   SIDEINFO DATASET(SYS1.APPCSI) ◄── APPC/MVS VSAM SI data set
```

Figure 12. Basic LUADD Definition

The parameters in Figure 12 do the following:

ACBNAME(MVSLU01)
: Defines the name of the LU as MVSLU01. The name specified must be the same as the LU name specified to VTAM.

BASE
: Indicates that the LU specified for ACBNAME (MVSLU01 in this case) is the base LU for APPC/MVS. The base LU is associated with the APPC/MVS transaction scheduler. APPC LU 6.2 conversations take place across the base LU.

TPDATA(SYS1.APPCTP)
: Defines the name of the TP profile data set as SYS1.APPCTP. The name specified must match the name of the APPC/MVS TP profile data set.

SIDEINFO DATASET(SYS1.APPCSI)
Defines the name of the VSAM KSDS side information data set as SYS1.APPCSI. The name specified must match the name of the APPC/MVS side information data set.

The LUADD SCHED parameter (omitted in the above example) specifies the name of the APPC/MVS transaction scheduler. If omitted, it defaults to the value SCHED(ASCH).

For more information on using the LUADD definition, see *z/OS MVS Planning: APPC/MVS Management*.

Creating the Distributed FileManager/MVS TP Profile

APPC/MVS enables Distributed FileManager/MVS to communicate across a computer network with DDM source implementations. To use APPC/MVS services, Distributed FileManager/MVS must be set up as an APPC/MVS TP. Every TP must have a TP profile contained in the APPC/MVS TP profile data set. The TP profile consists of scheduling and security information needed to run the TP.

Allocating a VSAM KSDS for the TP Profile

If you are not already using APPC/MVS, you must allocate a VSAM KSDS in the system library (for example, SYS1.APPCTP) where APPC/MVS TP profile information can be stored. The name of the VSAM KSDS data set must match the name defined in system PARMLIB member APPCPMxx. A sample of allocating the VSAM KSDS is in system SAMPLIB member ATBTPVSM (for example, SYS1.SAMPLIB(ATBTPVSM)).

For more information on allocating a VSAM KSDS, see *z/OS DFSMS Access Method Services*.

Adding the TP Profile to the VSAM KSDS

This discussion assumes that a VSAM KSDS already exists for TP profile information.

The APPC/MVS administration utility (ATBSDFMU) has commands for creating and modifying APPC/MVS TP profiles in the VSAM KSDS. You can use the TPADD command to add a TP profile. For a sample of using TPADD to add a variety of TP profiles, see system SAMPLIB member ATBUTIL (for example, SYS1.SAMPLIB(ATBUTIL)).

Figure 13 on page 38 shows a job step that adds a Distributed FileManager/MVS TP profile to SYS1.APPCTP using the TPADD command. This example can be found in system SAMPLIB member GDETPDEF (for example, SYS1.SAMPLIB(GDETPDEF) and in “GDETPDEF” on page 60.

```
//STEP     EXEC PGM=ATBSDFMU
//SYSPRINT DD   SYSOUT=*
//SYSSDOUT DD   SYSOUT=*
//SYSSDLIB DD   DSN=SYS1.APPCTP,DISP=SHR
//SYSIN    DD   DATA,DLM=XX
    TPDELETE
      TPNAME(ˆX'07'001)
    TPADD
      TPNAME(ˆX'07'001)                 <- TP key section
      ACTIVE(YES)                       <- TP attribute section
      TPSCHED_DELIMITER(##)             <------. TP scheduler section
        CLASS(A)                               |
        JCL_DELIMITER(ENDJCL)                  |
//GDEDFM JOB MSGCLASS=H,MSGLEVEL=(1,1),CLASS=A |
//GDEDFM    EXEC PGM=GDEISASB                  |
//* CHANGE THE STEPLIB STATEMENT AS REQUIRED IF YOUR INSTALLATION
//* DOESN'T HAVE THE LE RUNTIME DATA SET IN ITS LINK LIST.
//*STEPLIB  DD  DSN=SYS1.SCEERUN,DISP=SHR
//*CDRATRC DD DSN=SYS1.CDRATRC2,DISP=SHR  <- CDRA API TRACE OUTPUT
//*SYSOUT  DD DSN=SYS1.CDRAOUT2,DISP=SHR  <- C RUNTIME MESSAGES
//SYSOUT   DD DUMMY                       <- C RUNTIME MESSAGES (NO-OP)
ENDJCL                                         |
##                                        <------'
XX
```

Figure 13. TPADD Command Example

As Figure 13 shows, each TP profile contains three sections.

TP profile key section
Consists of a TP name and a TP level (because TP level is not included in this example, TP Level defaults to SYSTEM).

- The TP name for Distributed FileManager/MVS must be ˆX'07'001
- TP level identifies which entities RACF authorizes to access the Distributed FileManager/MVS TP. It can be one of the following:

Level	**Access**
SYSTEM	Any user can attach the TP. This is the default if no TP level is used.
GROUP	Any member of a predefined group of users can attach the TP.
USER	A single user can attach the TP.

TP attributes section
Consists of `ACTIVE(YES)`, which indicates that the TP status is active. If the status is set to `ACTIVE(NO)`, the TP cannot be scheduled.

TP scheduler section
Has the following information:

- Provides the JCL used to run the Distributed FileManager/MVS TP. This example shows sample JOB and EXEC statements.
- SCHED(A) indicates the APPC/MVS transaction scheduler for the Distributed FileManager/MVS TP.
- GDEDFM should have either no region size or a region size of `0K` to contain cached stream files.

- GDEISASB must be specified as the program to be executed on the EXEC statement.

  Note: The language environment (LE) is required to use CDRA. If LE is installed and is not in the link list, SYS1.PROCLIB(DFM) and SYS1.SAMPLIB(GDETPDEF) should be modified so their STEPLIB DD statements refer to the proper LE run time library. Refer to DFMREADM in SYS1.SAMPLIB for details. SYSOUT and CDRATRC files can be allocated as RECFM=FBA, LRECL=133, and DSORG=PS for use in diagnosing CDRA problems.

TP profile definition parameters not included in this example are set to default values. For more details about adding and modifying TP profile information, see *z/OS MVS Planning: APPC/MVS Management*.

Creating the APPC/MVS Side Information Data Set

APPC/MVS requires that a VSAM KSDS data set be allocated for the side information data set. The APPC/MVS side information data set contains translations of symbolic destination names used by TPs when they issue outbound allocate requests.

Because Distributed FileManager/MVS is a DDM target only and therefore does not issue outbound allocate requests, the side information data set does not need to contain any information. The name used for the SIDEINFO parameter in system PARMLIB member APPCPMxx must match the side information data set name.

Sample JCL for allocating the side information KSDS data set can be found in system SAMPLIB member ATBSIVSM (for example, SYS1.SAMPLIB(ATBSIVSM)). For information about creating side information, see *z/OS MVS Planning: APPC/MVS Management*.

Defining PARMLIB Start Parameters for the APPC/MVS Scheduler

The APPC/MVS transaction scheduler (ASCH) initiates and schedules TPs in response to inbound requests for conversations. Start parameter values in system PARMLIB member ASCHPMxx (for example, SYS1.PARMLIB(ASCHPMxx)) define and modify TP scheduling classes and other TP scheduling characteristics to ASCH.

ASCH is started by a system operator command. The operator command (can be part of initial program load) identifies the ASCHPMxx member (see “Starting Up the APPC/MVS Transaction Scheduler” on page 51). Both APPC/MVS (also started by operator command) and ASCH must be active before Distributed FileManager/MVS LU 6.2 conversations can take place.

You can use an APPC/MVS CLASSADD definition to define the ASCH start parameters in ASCHPMxx, as shown in the following example. This example can be found in system SAMPLIB member GDEASCH (for example, SYS1.SAMPLIB(GDEASCH)) or in “GDEASCH” on page 56.

```
CLASSADD CLASSNAME(A)
   MSGLIMIT(1000) MAX(10) MIN(1) RESPGOAL(1)
```

The parameters in this example do the following:

CLASSADD CLASSNAME(A)
Defines a class of transaction initiators to ASCH. A transaction initiator is an

entity, such as Distributed FileManager/MVS, that functions as an APPC/MVS TP. TPs in the same class should have similar characteristics such as run-time, priority, schedule-type, and security.

The CLASSNAME parameter defines the class name as A. It must match the class name used in the TP profile for Distributed FileManager/MVS.

MSGLIMIT(1000)
Defines 1000 as the maximum number of messages in the message log data set for TPs in this class.

MAX(10)
Defines 10 as the maximum number of transaction initiators allowed for this class.

MIN(1)
Defines 1 as the number of transaction initiators in this class that will be started when ASCH is started.

RESPGOAL(1)
Defines 1 second as the system response time goal for TPs running in this class.

For more information about defining ASCH and about scheduling TPs, see *z/OS MVS Planning: APPC/MVS Management*.

VTAM Customizing Tasks

Customizing VTAM for Distributed FileManager/MVS includes defining the local LU to VTAMLST, setting up the logon mode table in VTAMLIB, and establishing local LUs and logon mode definitions on partner systems.

Defining the Local LU to VTAMLST

When VTAM is initialized on an MVS system, local LUs are activated based on information in the system VTAMLST library. To define the local APPC/MVS LU to VTAM, use a VTAM application (APPL) definition in the VTAMLST library that is defined in the VTAM start procedure. The VTAM APPL definition:

- Names the local APPC/MVS LU
- Sets up defaults for the LU
- Specifies the name of the logon mode table that contains the logon mode used by the LU
- Defines security for the LU

Figure 14 on page 41 is an example of a VTAM APPL definition. This example can be found in system SAMPLIB member GDEAPDEF, for example, SYS1.SAMPLIB(GDEAPDEF), or see “GDEAPDEF” on page 55.

```
   MVSLU01 APPL   ACBNAME=MVSLU01,  ◄─── ACBNAME (also
APPC/MVS LUADD)
                  APPC=YES,
                  AUTOSES=0,
                  DDRAINL=NALLOW,
                  DMINWNL=5,
                  DMINWNR=5,
                  DRESPL=NALLOW,
                  DSESLIM=10,
                  LMDENT=19,
                  MODETAB=LOGMODES, ◄─── VTAM Logon Mode Table name
                  PARSESS=YES,
                  SECACPT=CONV,
                  SRBEXIT=YES,
                  VPACING=1
```

Figure 14. VTAM APPL Definition

The ACBNAME and MODETAB parameters in this example do the following:

ACBNAME=MVSLU01
Defines the local APPC/MVS LU name as MVSLU01. The LU name specified must match the local LU name defined in PARMLIB member APPCPMxx.

MODETAB=LOGMODES
Defines the name of the logon mode table as LOGMODES. This parameter is optional. Including it, however, allows you to make additional logon mode definitions known to VTAM. This is required if the logon mode name specified by the partner system is not supplied in the VTAM default logon mode table ISTINCLM.

The name for the logon mode table name must match the name of a defined in VTAMLIB.

For more information about VTAM APPL definitions, see *z/OS MVS Planning: APPC/MVS Management*.

Defining APPC/MVS Logon Mode Entry in VTAMLIB

A logon mode is a set of parameters and protocols that determines the communication characteristics of a VTAM session. Logon modes are entries in a logon mode table contained in the system VTAMLIB library.

APPC/MVS requires a logon mode entry in a logon mode table. The logon mode table containing the APPC/MVS logon mode entry must be assembled and linked into the VTAMLIB library defined by the VTAM start procedure. System SAMPLIB member ATBLJOB (for example, SYS1.SAMPLIB(ATBLJOB)) provides sample JCL for assembling and linking a logon mode table.

Figure 15 on page 42 is an example of a logon mode table containing several logon mode entries. This example can be found in system SAMPLIB member GDELOGMD (for example, SYS1.SAMPLIB(GDELOGMD)) or see “GDELOGMD” on page 58.

```
LOGMODES MODETAB   ◄─── VTAM APPL LOGMODE table name
         EJECT
***********************************************************************
         TITLE 'SNASVCMG'                                             *
***********************************************************************
*        LOGMODE TABLE ENTRY FOR RESOURCES CAPABLE OF ACTING         *
*               AS LU 6.2 DEVICES                                     *
*               REQUIRED FOR LU MANAGEMENT                            *
***********************************************************************
SNASVCMG MODEENT LOGMODE=SNASVCMG,FMPROF=X'13',TSPROF=X'07',      *
               PRIPROT=X'B0',SECPROT=X'B0',COMPROT=X'D0B1',       *
               RUSIZES=X'8585',ENCR=B'0000',                          *
               PSERVIC=X'060200000000000000000300'
***********************************************************************
         TITLE 'QPCSUPP '                                             *
***********************************************************************
*        LOGMODE TABLE ENTRY FOR RESOURCES CAPABLE OF ACTING         *
*               AS LU 6.2 DEVICES                                     *
*               REQUIRED FOR LU MANAGEMENT                            *
***********************************************************************
QPCSUPP  MODEENT LOGMODE=QPCSUPP,FMPROF=X'13',TSPROF=X'07',           *
               PRIPROT=X'B0',SECPROT=X'B0',COMPROT=X'D0B1',           *
               RUSIZES=X'8585',ENCR=B'0000',                          *
               PSERVIC=X'060200000000000000000300'
***********************************************************************
         TITLE 'APPCPCLM'
***********************************************************************
*        LOGMODE TABLE ENTRY FOR RESOURCES CAPABLE OF ACTING         *
*               AS LU 6.2 DEVICES                                     *
*               FOR PC TARGET                                         *
*   IN THIS EXAMPLE THE DEFAULT RU SIZE FOR OS/2 (1024) IS USED      *
***********************************************************************
APPCPCLM MODEENT LOGMODE=APPCPCLM,                                    *
               RUSIZES=X'8787',                                       *
               SRCVPAC=X'00',                                         *
               SSNDPAC=X'01'
***********************************************************************
         TITLE 'APPCHOST'
***********************************************************************
*        LOGMODE TABLE ENTRY FOR RESOURCES CAPABLE OF ACTING         *
*               AS LU 6.2 DEVICES                                     *
*               FOR HOST TARGET                                       *
*   IN THIS EXAMPLE RU SIZE OF 4096 IS USED                           *
***********************************************************************
APPCHOST MODEENT LOGMODE=APPCHOST,                                    *
               RUSIZES=X'8989',                                       *
               SRCVPAC=X'00',                                         *
               SSNDPAC=X'01'
         MODEEND
         END
```

Figure 15. Logon Mode Table

The name of the logon mode table defined in the VTAM APPL definition must match the name defined in VTAMLIB. In Figure 15, for example, the name of the logon mode table defined to VTAMLIB is LOGMODES.

For more information about defining the APPC/MVS logon mode, see *z/OS MVS Planning: APPC/MVS Management*.

Defining LU and Logon Mode on Partner Systems

For Distributed FileManager/MVS to conduct a network conversation with a DDM source implementation, each system in the conversation must know its partner's LU name and logon mode information. Establishing partner information involves the following steps:

1. You need to define on the partner (source) system the local LU, logon mode entry, and partner LU (the LU associated with Distributed FileManager/MVS). What conventions and utilities you use for defining this information depends on what platform the partner system runs. On an OS/2 system, for example, you use Communications Manager for specifying network information.

   The logon mode name on the partner system must match a logon mode table entry name defined to VTAM and associated with the VTAM APPL definition. This association can be explicit in the APPL MODETAB definition statement or implicit in the VTAM-supplied default logon mode table ISTINCLM.

   When the partner system sends an APPC allocate call to initiate a conversation with Distributed FileManager/MVS, it sends the name of a logon mode definition that must match a logon mode entry name defined to VTAM. If the partner system is an OS/2, you must use the QPCSUPP logon mode entry name.

2. On the target MVS system, you need to identify to VTAM the name of the partner LU. This name must match the local LU name that you have established on the partner system.

   For information about defining partner information to VTAM, see *VTAM Network Implementation Guide* , SC31-6434; *VTAM Resource Definition Samples* , SC31-6414; and *VTAM Resource Definition Samples* , SC31-6414.

Defining Partner Information on OS/2

The following is an example of defining a local LU and a VTAM logon mode table specification for a partner OS/2. This example can be found in system SAMPLIB member GDEPRTLU (for example, SYS1.SAMPLIB(GDEPRTLU)) or see "GDEPRTLU" on page 62.

```
OS2PRTNR LU  LOCADDR=0,
             ISTATUS=ACTIVE,
             MODETAB=LOGMODES   ◄─── VTAM Logon Mode Table name
             RESSCB=4
```

Distributed FileManager/MVS Customizing Tasks

Customizing Distributed FileManager/MVS includes installing and tuning Distributed FileManager/MVS startup parameters in system PARMLIB, activating the Distributed FileManager/MVS startup procedure in system PROCLIB, and verifying PPT entries for Distributed FileManager/MVS.

Tuning Distributed FileManager/MVS Startup Parameters in System PARMLIB

You can tune the startup parameters for Distributed FileManager/MVS to fit your installation's performance requirements. These parameters are contained in PARMLIB member DFM00 (for example, SYS1.PARMLIB(DFM00)). If DFM00 needs to be installed in PARMLIB on your system, you can copy system SAMPLIB member DFM00 (for example, SYS1.SAMPLIB(DFM00)). DFM00 can also be found in "DFM00" on page 57.

The parameters shown in Table 1 fall into categories related to either performance tuning or data set definition defaults.

Table 1. Tunable Parameters in DFM00

Parameter	Description	Default	Range
CLOSE_CHECK_INTV	Time interval (in seconds) to wait between searches of the Open PDSE queue for data sets to close	0	0—100
DEFER_CLOSE_TIME	Time interval (in seconds) to wait before closing a PDSE data set after a DDM close command has been processed	0	0—100
LOCK_RETRY	Number of lock conflict retries	3	1—100
LOCK_WAIT_INTV	File lock wait interval in seconds	20	1—100
MAX_AGENT_TSKS	Number of agents supported	5	1—100
MAX_CONV_LOCK	Maximum locks on a file per agent	5	1—100
SEND_BUFFER_THRESHOLD	Maximum number of buffers between APPC SEND verb completions	100	1—1,000
LOGICAL_CACHE	Cache limit for stream files	1,024	1—2,047
CCSID	Coded character set identifier (CCSID) for Distributed FileManager/MVS	0	0—65,535
PRIMARY	Data set space allocation in records (non-SMS only)	100	1—2GB
SECONDARY	Data set space allocation in records when PRIMARY space is exhausted (non-SMS only)	50	0—2GB
STREAM_LRECL	Logical record length for stream files	8,196	0—32,760
UNIT	Device type where non-SMS data sets are created, see VOLUME	SYSALLDA	N/A
VOLUME	DASD volume serial number for non-SMS data set creation	None	N/A
RESTRICT_START	Startup command in the PARMLIB DFM00 member	Yes	Yes or No

Parameters Related to Performance

CLOSE_CHECK_INTV and DEFER_CLOSE_TIME

These parameters offer a trade-off between concurrency and PDSE processing performance. If typical usage on your system tends to reaccess the same or other members of a PDSE, these parameters can be specified as nonzero values to leave PDSEs open longer. Then, when PDSE members are reaccessed, the overhead of closing and reopening data sets is eliminated. The trade-off is that the data sets might be unavailable to other remote or local users longer than necessary.

MAX_AGENT_TSKS

This parameter can be used as a control on Distributed FileManager/MVS resources. It determines the maximum number of concurrent remote user tasks that the target server will allow.

MAX_CONV_LOCK

This parameter establishes a limit on how many locks each agent can have. If you think of a lock as representing a system resource (in this case, a data set), then setting a maximum value for the number of locks that can be held establishes a limit on how much serially reusable resource a given agent can use at one time.

LOCK_RETRY and LOCK_WAIT_INTV

These parameters control how soon lock contentions are detected. In an interactive environment where you can choose how to handle “try again later” messages, you might want short wait intervals and few lock conflict retries. However, in a more batch-oriented environment, you might want the opposite to avoid terminating batch jobs just because a lock is temporarily unavailable.

SEND_BUFFER_THRESHOLD

For this parameter, the maximum number of buffers between APPC SEND verb completions should be fairly large to avoid irregularities in system response and to maximize concurrency. Specifying too large a value, however, could result in excess paging.

In some cases, this parameter can increase the overall auxiliary storage requirements of the system. As a general rule, you can determine the auxiliary storage increase by adding up the estimates for the following:

- The total size of the stream-oriented files that are likely to be accessed concurrently by a typical address space
- The space required for input buffers (up to the combined file size)
- The storage required for output buffers (SEND_BUFFER_THRESHOLD times 32k)

Take the resulting sum and multiply it by the number of concurrently running address spaces, then add 25% to allow for control block overhead and unused space at the end of some of the buffers.

LOGICAL_CACHE

This parameter allows you to limit the amount of virtual storage a DFM conversation can use for caching stream files. When the limit is reached, the current stream request is terminated. You can use this parameter to minimize the potential impact of DFM/MVS on MVS system performance.

Parameters Related to Data Set Definition

CCSID

Use this parameter to establish the default value for the CCSID associated with data sets that will be created by Distributed FileManager/MVS. This CCSID, unless over-ridden by the workstation, defines the code page in which stream files are stored when the workstation requests data conversion by specifying the TEXT option. In most cases, the value should be left as zero. Zero is a special value that causes the default CCSID to be inherited from a higher level in the hierarchy. Currently the only value that can be inherited is 500, EBCDIC International Latin-1. The inheritance occurs from DFM itself rather than from the operating system.

If the CCSID is not supported by Character Data Representation Architecture (CDRA), startup will end with message GDE006E indicating that the CCSID keyword has an incorrect value. The return code shown will be that defined for CDRA’s CDRGCTL function. If LE is not installed, message GDE006E will be issued for an invalid CCSID with a return code of X’FFFFFFFF’. DFM startup will indicate it is not started, but is actually started in a partial non-data conversion mode.

PRIMARY

This parameter defines the amount of space requested by a user for a data set when it is created. This parameter applied only to the creation of non-SMS data sets. The default primary space allocation is 100 records.

SECONDARY
This parameter defines the amount of additional space requested by the user for a data set when primary space is full. This parameter applied only to the creation of non-SMS data sets. The default secondary space allocation is 50 records.

STREAM_LRECL
This parameter provides a default value for block size if the logical record length is not provided by the SMS DATACLASS. If ACS routines or data classes are established so that all logical record length specifications are provided through SMS data classes, this parameter can be specified as 0. Larger values than 8196 can give better performance, but generally the most important consideration is whether the data will be shared with MVS applications.

If the data is shared, the needs of the MVS applications should determine the logical record length. For example, choosing a small value might allow easy editing or browsing of the file on MVS. If, however, the data will not be shared with MVS applications, the larger the logical record length the better the performance will tend to be.

UNIT This parameter defines the device type where non-SMS data sets are created, see VOLUME.

VOLUME
This parameter defines the DASD serial number for non-SMS data set creation.

When Distributed FileManager/MVS is installed, the VOLUME parameter must be activated if the installation does not use SMS or chooses not to establish ACS routines for the Distributed FileManager.

If SMS is not active and VOLUME is omitted, it is possible to create SAM data sets, but PDS data sets created will not have directory blocks assigned to them and a ″file damaged″ error will occur.

Parameters Related to DataAgent

RESTRICT_START
This parameter is the startup command in the PARMLIB DFM00 member. The default is YES.

Activating Distributed FileManager/MVS in System PROCLIB

Activating Distributed FileManager/MVS involves adding a startup procedure to a new system PROCLIB member called DFM, for example, SYS1.PROCLIB(DFM). Once DFM is added to PROCLIB, Distributed FileManager/MVS can be started by a system operator command. The operator command, which can be part of initial program load, identifies the DFM member (see “Starting Up Distributed FileManager/MVS” on page 51).

APPC/MVS and the APPC/MVS transaction scheduler (both are started by operator command) must be active before Distributed FileManager/MVS LU 6.2 conversations can take place.

Figure 16 on page 47 is an example of the contents of the DFM member.

```
//DFM     PROC  PARMS='NORMAL'
//*************************************************************
//*                                                           *
//*  DFSMS MVS/DFM START UP PROCEDURE                         *
//*                                                           *
//*************************************************************
//DFM      EXEC PGM=GDEISBOT,
//             PARM='&PARMS',
//             REGION=0K,
//             TIME=1440
//IEFPARM  DD   DSN=SYS1.PARMLIB,DISP=SHR
//* CHANGE THE STEPLIB STATEMENT AS REQUIRED IF YOUR INSTALLATION
//* DOESN'T HAVE THE LE RUNTIME DATA SET IN ITS LINK LIST.
//*STEPLIB  DD  DSN=SYS1.SCEERUN,DISP=SHR
//SYSPRINT DD   SYSOUT=*
//SYSUDUMP DD   SYSOUT=*
//*
//*  THE TWO FILES ASSOCIATED WITH THE DD STATEMENTS CDRATRC AND
//*  SYSOUT CAN BE USED TO DIAGNOSE DFM STARTUP PROBLEMS RELATED
//*  TO CDRA. (CDRA IS INVOKED DURING STARTUP FOR CERTAIN CCSID
//*  VALUES IN THE SYS1.PARMLIB MEMBER DFM00.)
//*
//*  YOU MUST ALLOCATE THE TWO FILES AS RECFM=FBA, LRECL=133,
//*  AND DSORG=PS BEFORE STARTING DFM WITH THE DD STATEMENTS
//*  ACTIVE.
//*
//*  NOTE THAT SYSOUT IS REQUIRED AND CDRATRC IS OPTIONAL
//*  WHEN USING CDRA AND THE DEFAULT INSTALLATION IS SET UP TO
//*  USE CDRA IF YOUR HOST CODE PAGE IS OTHER THAN 500.
//*
//*  CDRATRC  DD  DSN=SYS1.CDRATRC,DISP=SHR    CDRA API TRACING
//*  SYSOUT   DD  DSN=SYS1.CDRAOUT,DISP=SHR    C RUNTIME MESSAGES
//SYSOUT   DD  DUMMY <- DEFAULT = CDRA WITH RUNTIME MESSAGES DISCARDED
```

Figure 16. DFM Member Example

Note: LE is required to use CDRA, if LE is installed and is not in the link list, SYS1.PROCLIB(DFM) and SYS1.SAMPLIB(GDETPDEF) should be modified so their STEPLIB DD statements refer to the proper LE run time library. Refer to DFMREADM in SYS1.SAMPLIB for details.

Verifying PPT Entries for Distributed FileManager/MVS

To execute correctly, Distributed FileManager/MVS must have entries in the system program property table (PPT). These entries are automatically included in the base PPT for your installation (system LINKLIB member IEFSDPPT). If the need arises to override this base PPT, you can add the entries to system PARMLIB member SCHEDxx (for example, SYS1.PARMLIB(SCHEDxx)). For a sample of the entries, see “Appendix I. PPT Entries for Distributed FileManager/MVS” on page 101. PARMLIB(SHEDxx) members for these sample entries should not be created without prior discussion with your IBM service representative.

ACS Routines for Defining Distributed FileManager/MVS SMS Classes

ACS routines determine the SMS classes for data sets. For data sets to be classified as SMS-managed, they must be defined in a storage class. Distributed FileManager/MVS only permits remote creation of MVS data sets when the resultant data set is SMS-managed. If a request to create a data set would result in a non-SMS-managed data set, Distributed FileManager/MVS rejects the request.

Figure 17, Figure 18, and Figure 19 on page 49 are sample ACS routines for defining data, management, and storage classes for data sets created by Distributed FileManager/MVS.

```
/*                   DATACLAS ROUTINE                             */
/* DEFAULT DATACLASSES FOR DFM                                    */
/*                                                                */
IF &JOB = 'GDEDFM' AND &DATACLAS = '' THEN
  DO
    SELECT
      WHEN (&RECORG = 'KS') SET &DATACLAS = 'KS000000'
      WHEN (&RECORG = 'ES') SET &DATACLAS = 'ES000000'
      WHEN (&RECORG = 'RR') SET &DATACLAS = 'RR000000'
      WHEN (&DSORG = 'PS') SET &DATACLAS = 'PS000000'
      WHEN (&DSNTYPE = 'LIBRARY') SET &DATACLAS = 'LIB00000'
      OTHERWISE WRITE 'NOT A SUPPORTED DFM DATASET TYPE'
    END /* SELECT */
    /* DEBUGGING STATEMENT FOLLOWS.  REMOVE IT WHEN ROUTINE IS OK. */
    IF &DATACLAS ˆ= '' THEN
      WRITE 'DATACLAS SET TO '&DATACLAS' FOR DFM'
    EXIT CODE(0)
  END /* DO */
```

Figure 17. Data Class Routine

If the logical record length for stream files is specified as zero (STREAM_LRECL(0)) in the system PARMLIB member DFM00, you must select a data class providing a nonzero record length.

```
/*                 MGMTCLAS ROUTINE                             */
/* IF JOB IS DFM                                                */
/* SET MGMTCLAS TO DFMMGMT.                                     */
/*                                                              */
IF &JOB = 'GDEDFM' THEN
  DO
    SET &MGMTCLAS = 'DFMMGMT'
   /* DEBUGGING STATEMENT FOLLOWS.  REMOVE IT WHEN ROUTINE IS OK. */
    WRITE '&MGMTCLAS SET TO '&MGMTCLAS' FOR DFM'
    EXIT CODE(0)
  END  /* GDEDFM  */
/*                                                              */
```

Figure 18. Management Class Routine

```
 /*                  STORCLAS ROUTINE                                  */
 /* DEFAULT STORCLAS FOR DFM IS DFMCLASS.                              */
 /*                                                                    */
 /*                                                                    */
IF &JOB = 'GDEDFM' AND &STORCLAS = '' THEN
     DO
       SET &STORCLAS = 'DFMCLASS'
   /* DEBUGGING STATEMENT FOLLOWS.  REMOVE IT WHEN ROUTINE IS OK.   */
       WRITE 'STORCLAS SET TO '&STORCLAS' FOR DFM'
       EXIT CODE(0)
    END
 /*                                                                    */
/*                                                                     */
```

Figure 19. Storage Class Routine

Data sets without a storage class cannot be SMS-managed.

Establishing Distributed FileManager/MVS TP Access Security

You need to establish access security for the Distributed FileManager/MVS TP so that only authorized users and applications can remotely access it. To protect the Distributed FileManager/MVS TP, you can:

- Limit which LUs can enter your MVS system
- Ensure that inbound requests to initiate conversations with the Distributed FileManager/MVS TP contain security information such as user IDs and passwords
- Limit by user ID or group who can access the Distributed FileManager/MVS TP
- Limit the administrators who can define and update information in the Distributed FileManager/MVS TP profile

Using RACF to Control Access to the Distributed FileManager/MVS TP

You can use RACF (or an equivalent product) to control which user IDs or groups of user IDs are authorized to access Distributed FileManager/MVS. To accomplish this, you need the following information:

- Name of your Distributed FileManager/MVS TP profile
- User IDs that will be authorized EXECUTE access to your APPC/MVS TP
- User IDs that will be authorized as APPC/MVS administrators to read and update Distributed FileManager/MVS TP profile information

The RACF APPCTP resource class controls the use of the APPC/MVS TP. Profiles in this resource class define which user IDs can execute the APPC/MVS TP. The names of these profiles are in the form *dbtoken.level.tpname*, where

dbtoken
The database token associated with the Distributed FileManager/MVS TP profile (1 to 8 characters). The TP profile must have a database token, or else APPC/MVS cannot call RACF for TP access security. For more information about adding a database token, see *z/OS MVS Planning: APPC/MVS Management*.

level This is one of the following:

- The name of your system library (for example, SYS1), if the TP is available to all users who can access the LU
- A group ID, if the TP is available to a group
- A user ID, if the TP is available to just a specific user

tpname
: The name of the Distributed FileManager/MVS TP profile, which is always ˆX'07'001 (see "Adding the TP Profile to the VSAM KSDS" on page 37)

Defining the Distributed FileManager/MVS TP Profile to RACF

The following example defines to RACF the Distributed FileManager/MVS TP profile name (ˆX'07'001) in the RACF APPCTP class:

```
RDEFINE APPCTP TOKEN1.SYSTEM.ˆX'07'001 UACC(NONE)
```

Defining a TP Administrator to RACF

The following example defines to RACF the user ID ADMIN01 with update access to the Distributed FileManager/MVS TP profile:

```
PERMIT TOKEN1.SYSTEM.ˆX'07'001 CLASS (APPCTP) ID (ADMIN01) ACCESS(UPDATE)
```

Defining a User ID to RACF

The following example defines to RACF the user ID DFMUSER with authorization to execute the Distributed FileManager/MVS TP:

```
PERMIT TOKEN1.SYSTEM.ˆX'07'001 CLASS (APPCTP) ID (DFMUSER) ACCESS(EXECUTE)
```

Implementing RACF Access Protection for TP

To implement RACF protection as defined in the APPCTP profile, you must activate in RACF the APPCTP class and SETROPTS RACLIST for the class. For example:

```
SECTROPTS CLASSACT(APPCTP) RACLIST(APPCTP)
```

For more detailed information about using RACF to control Distributed FileManager/MVS TP access, see *z/OS MVS Planning: APPC/MVS Management*.

Chapter 4. Operating Distributed FileManager/MVS

This chapter is about operating Distributed FileManager/MVS on an MVS/ESA, z/OS, or OS/390 system. It covers procedures for starting up the Distributed FileManager/MVS environment and for monitoring and controlling the status of Distributed FileManager/MVS conversations.

For more information, see *z/OS MVS Planning: APPC/MVS Management*.

Starting Up the Distributed FileManager/MVS Environment

Starting up the Distributed FileManager/MVS environment requires that computer operations run procedures to start up APPC/MVS, the APPC/MVS transaction scheduler, and Distributed FileManager/MVS.

APPC/MVS and the APPC/MVS transaction scheduler must be started before starting Distributed FileManager/MVS. To automate these procedures at initial program load (IPL), you can add the startup commands to the system PARMLIB member COMMNDxx (for example, SYS1.PARMLIB(COMMNDxx)).

Starting Up APPC/MVS

The startup parameters for APPC/MVS are in system PARMLIB member APPCPMxx. These parameters define the local LU to be used for APPC/MVS. They associate the LU with an APPC/MVS transaction scheduler and the Distributed FileManager/MVS TP profile. See “Defining PARMLIB Start Parameters for APPC/MVS” on page 36 for more details.

The following command starts up APPC/MVS:

```
START APPC,SUB=MSTR,APPC=xx
 where xx is the unique APPCPMxx suffix
```

Starting Up the APPC/MVS Transaction Scheduler

The APPC/MVS transaction scheduler initiates and schedules TPs in response to inbound allocate requests. The system PARMLIB member ASCHPMxx controls the transaction scheduler for the Distributed FileManager/MVS TP. For more information about creating ASCHPMxx, see “Defining PARMLIB Start Parameters for the APPC/MVS Scheduler” on page 39.

The following command starts up the APPC/MVS transaction scheduler:

```
START ASCH,SUB=MSTR,ASCH=xx
 where xx is the unique ASCHPMxx suffix
```

Starting Up Distributed FileManager/MVS

System PROCLIB member DFM contains the procedure for starting up &mvsdfm (see “Activating Distributed FileManager/MVS in System PROCLIB” on page 46). DFM must be active prior to APPC/MVS initiating a conversation.

The following command starts up Distributed FileManager/MVS:

```
START DFM,SUB=MSTR
```

Triggering the Distributed FileManager/MVS DataAgent

A DFM/MVS DataAgent can only be triggered from an SdU application or a DDM application that is written to call the DataAgent from a client workstation. IBM provides sample DataAgent routines, DFMXAGNT, DFMQTSO, DFMXSORT, and DFMXTSO that you can execute. You can also use these routines as examples to help write your own DataAgent routines. Your SdU application or DDM application uses the DDMOpen function to trigger the DataAgent processing on MVS and it uses the DDMClose function to terminate the DataAgent processing.

Monitoring Status of Distributed FileManager/MVS Conversations

APPC/MVS provides the DISPLAY command for monitoring the status of APPC/MVS conversations. The DISPLAY APPC command gives status information about TPs and LUs. The DISPLAY ASCH command gives status information about APPC/MVS transaction schedulers. “Using the DISPLAY APPC Command” provides examples of using these commands.

Using the DISPLAY APPC Command

These are examples of using the DISPLAY APPC command.

Displaying TP Status Information

These examples use the DISPLAY APPC command to return selected TP status information about the following:

- Distributed FileManager/MVS TP:
```
DISPLAY APPC,TP,LIST,LTPN=ˆX'07'001
```
- TPs scheduled by ASCH:
```
DISPLAY APPC,LIST,SCHED=ASCH
```
- Distributed FileManager/MVS TP in a particular address space:
```
DISPLAY APPC,LIST,ASID=asid
 where asid is the hexadecimal address space identifier
```
- TPs activated by a specific user ID:
```
DISPLAY APPC,LIST,USERID=userid
```

Displaying LU Status Information

These examples use the DISPLAY APPC command to return selected LU status information about the following:

- A local LU:
```
DISPLAY APPC,LU,LIST,LLUN=lluname
 where lluname is the name of a local LU
```
- All LUs (includes detailed information about local and partner LUs):
```
DISPLAY APPC,LU,ALL
```

Using the DISPLAY ASCH Command

These examples use the DISPLAY ASCH command to return selected status information about ASCH:

- Summary of ASCH transaction scheduling information (includes summary of all APPC/MVS transaction scheduling activity):
```
DISPLAY ASCH,SUMMARY
```
- Distributed FileManager/MVS TP scheduling information:
```
DISPLAY ASCH,LIST,ASID=001E
 where 001E is the hexadecimal address space identifier for
 the Distributed FileManager/MVS TP
```

- TPs scheduled by a specific user ID:

```
DISPLAY ASCH,LIST,USERID=userid
```

Controlling Status of Distributed FileManager/MVS Conversations

This discussion covers the following ways for controlling the status of Distributed FileManager/MVS conversations:
- Deactivating the Distributed FileManager/MVS TP
- Stopping a local LU with the MVS SET™ command
- Stopping Distributed FileManager/MVS with the MVS CANCEL command
- Using the MVS FORCE command

Deactivating the Distributed FileManager/MVS TP

You can deactivate the Distributed FileManager/MVS TP by modifying its TP profile. Using the TPMODIFY command in the APPC/MVS administration utility (ATBSDFMU), you can stop the Distributed FileManager/MVS TP from being scheduled and stop new requests for the TP.

The TPMODIFY command lets you change the active status of the Distributed FileManager/MVS TP to `NO` in the TP profile data set. If the Distributed FileManager/MVS TP is running at the time, then the current and any queued requests are allowed to complete. No new requests, however, are allowed.

This is an example of using the TPMODIFY command to deactivate the Distributed FileManager/MVS TP:

```
TPMODIFY
 TPNAME(ˆX'07'001)
 SYSTEM
 ACTIVE(NO)
```

For more information, see *z/OS MVS Planning: APPC/MVS Management*.

Stopping a Local LU with the MVS SET Command

You can stop work from coming into a local LU by using the MVS SET command to delete an LU from the APPC/MVS configuration. Use this method to:
- Stop an LU that is not functioning properly (for example, because of a VTAM error)
- Stop TPs defined in a TP profile data set that uses one or more LUs
- Stop a TP scheduler

To stop an LU by using the SET command:
1. Code system PARMLIB member APPCPMxx with the command to delete the LU. For example, to delete an LU named MYLU, code PARMLIB member APPCPM1D as follows:

   ```
   LUDEL
    ACBNAME(MYLU)
   ```
2. After coding APPCPM1D, issue this command to stop the LU:

   ```
   SET APPC=1D
   ```

For more information about deleting LUs, see *z/OS MVS Planning: APPC/MVS Management*.

Stopping DFM/MVS with the MVS CANCEL Command

You can use the MVS CANCEL command to immediately stop the Distributed FileManager/MVS TP in a particular address space. It can also be used to immediately stop the Distributed FileManager/MVS startup procedures APPC/MVS, ASCH, and DFM. If the CANCEL command is not successful, you can try the FORCE command (see “Using the FORCE Command”).

Stopping the Distributed FileManager/MVS TP

This example stops both the Distributed FileManager/MVS TP and any associated APPC/MVS conversation:

1. First use this command to find out the jobname and address space identifier (ASID) for your Distributed FileManager/MVS TP:

   ```
   DISPLAY ASCH,ALL,LTPN=ˆX'07'001
    where ˆX'07'001 is the Distributed FileManager/MVS TP profile name
   ```

2. Suppose that the jobname is GDEDFM and the ASID is 0044. Use this information as shown to immediately stop the Distributed FileManager/MVS TP:

   ```
   CANCEL GDEDFM,A=0044
   ```

Stopping APPC, ASCH, and DFM

The following examples stop APPC, ASCH, or DFM:

```
CANCEL APPC
CANCEL ASCH
CANCEL DFM
```

You should consider the following before using the CANCEL command:

- Before cancelling DFM, first cancel all jobs servicing APPC/MVS conversations, for example JOBNAME GDEDFM. Otherwise the jobs will abend when they try to access Distributed FileManager/MVS resources.
- Each time DFM is cancelled, the system marks the address space in which it was running as nonreusable until the next IPL.
- Cancelling APPC, ASCH, or DFM immediately ends all TPs and scheduling activity for APPC/MVS, which could have serious repercussions.

Note: 13E abends can occur during CANCEL command processing. These abends are perfectly normal and do not interfere with the CANCEL command processing.

Using the FORCE Command

If the CANCEL command is not successful, you can try the MVS FORCE command to stop the Distributed FileManager/MVS TP. Using the FORCE command can, however, result in loss of resources until the system is re-IPLed.

In this example, the FORCE command is used to stop a Distributed FileManager/MVS TP with jobname GDEDFM and an ASID of 0044:

```
FORCE GDEDFM,A=0044
```

Appendix A. System Samples

This appendix documents:

- The system SAMPLIB and PROCLIB samples related to customizing MVS for the Distributed FileManager/MVS environment
- A sample of the PPT entries for Distributed FileManager/MVS
- The DFM DataAgent DFMACALL.C sample

System SAMPLIB Samples

The following are system SAMPLIB samples that are referred to in "Chapter 3. Customizing MVS for Distributed FileManager/MVS" on page 33.

GDEAPPC

System SAMPLIB member GDEAPPC, for example, SYS1.SAMPLIB(GDEAPPC), contains the sample shown in Figure 20 of the APPC/MVS start parameters.

```
/* START OF SPECIFICATIONS *******************************************/
/*                                                                   */
/*01* MEMBER-NAME: GDEAPPC                                           */
/*                                                                   */
/*02* DESCRIPTIVE-NAME: DFSMS/MVS DISTRIBUTED FILEMANAGER SAMPLE TO  */
/*                      DEFINE A LOCAL LU TO APPC/MVS                */
/*                                                                   */
/*01* DISCLAIMER =                                                   */
/*                                                                   */
/*   THIS SAMPLE SOURCE IS PROVIDED FOR TUTORIAL PURPOSES ONLY. A    */
/*   COMPLETE HANDLING OF ERROR CONDITIONS HAS NOT BEEN SHOWN OR     */
/*   ATTEMPTED, AND THIS SOURCE HAS NOT BEEN SUBMITTED TO FORMAL IBM */
/*   TESTING. THIS SOURCE IS DISTRIBUTED ON AN 'AS IS' BASIS         */
/*   WITHOUT ANY WARRANTIES EITHER EXPRESSED OR IMPLIED.             */
/*                                                                   */
/*                                                                   */
/*01* FUNCTION:                                                      */
/*     THIS SAMPLE  MEMBER DEFINES AN LU TO APPC, ALONG WITH A VSAM  */
/*     DATASET FOR TP PROFILES AND A SECOND ONE FOR SIDE INFORMATION */
/*                                                                   */
/*                                                                   */
/*01* DISTRIBUTION LIBRARY: ASAMPLIB                                 */
/*                                                                   */
/*01* CHANGE-ACTIVITY:                                               */
/*                                                                   */
/* FLAG LINEITEM  FMID    DATE  ID  COMMENT                          */
/*  $L0=GDEAPPC  HDZ11B0 931009 DFSMS 1.2.0 DISTRIBUTED FILEMANAGER  */
/*                           SAMPLE TO ADD A LOCAL LU TO APPC/MVS    */
/*                                                                   */
/*                                                                   */
/*********************************************************************/
LUADD ACBNAME(MVSLU01) BASE TPDATA(SYS1.APPCTP)
SIDEINFO DATASET(SYS1.APPCSI)
```

Figure 20. APPC/MVS Start Parameters

GDEAPDEF

System SAMPLIB member GDEAPDEF, for example, SYS1.SAMPLIB(GDEAPDEF), contains the sample shown in Figure 21 on page 56 of a VTAM APPL definition in VTAMLST.

```
*/* START OF SPECIFICATIONS ********************************************
*                                                                      *
*01* MEMBER-NAME: GDEAPDEF                                             *
*                                                                      *
*02* DESCRIPTIVE-NAME: SAMPLE VTAM APPL STATEMENT FOR APPC/MVS         *
*                      NECESSARY TO RUN DFSMS DISTRIBUTED FILEMANAGER *
*                                                                      *
* 01* DISCLAIMER =                                                     *
*                                                                      *
*    THIS SAMPLE SOURCE IS PROVIDED FOR TUTORIAL PURPOSES ONLY. A     *
*    COMPLETE HANDLING OF ERROR CONDITIONS HAS NOT BEEN SHOWN OR      *
*    ATTEMPTED, AND THIS SOURCE HAS NOT BEEN SUBMITTED TO FORMAL IBM  *
*    TESTING. THIS SOURCE IS DISTRIBUTED ON AN 'AS IS' BASIS          *
*    WITHOUT ANY WARRANTIES EITHER EXPRESSED OR IMPLIED.              *
*                                                                      *
*                                                                      *
*01* FUNCTION: THIS APPL STATEMENT IDENTIFIES APPC/MVS AS A VTAM      *
*              APPLICATION, WITH ONE ACB DEFINED FOR LU MVSLU01.      *
*                                                                      *
*                                                                      *
*01* DISTRIBUTION LIBRARY: ASAMPLIB                                    *
*                                                                      *
*01* CHANGE-ACTIVITY:                                                  *
*                                                                      *
*  FLAG LINEITEM  FMID    DATE  ID  COMMENT                            *
*   $L0=GDEAPDEF HDZ11B0 931009 DFSMS 1.2.0 DISTRIBUTED FILEMANAGER   *
*                           SAMPLE VTAM APPL DEFINITION               *
*                                                                      *
************************************************************************
MVSLU01 APPL   ACBNAME=MVSLU01,                                        C
               APPC=YES,                                               C
               AUTOSES=0,                                              C
               DDRAINL=NALLOW,                                         C
               DMINWNL=5,                                              C
               DMINWNR=5,                                              C
               DRESPL=NALLOW,                                          C
               DSESLIM=10,                                             C
               LMDENT=19,                                              C
               MODETAB=LOGMODES,                                       C
               PARSESS=YES,                                            C
               SECACPT=CONV,                                           C
               SRBEXIT=YES,                                            C
               VPACING=1
```

Figure 21. VTAM APPL Definition in VTAMLST

GDEASCH

System SAMPLIB member GDEASCH, for example, SYS1.SAMPLIB(GDEASCH), contains the sample shown in Figure 22 on page 57 of start parameters for the APPC/MVS scheduler (ASCH).

```
/** START OF SPECIFICATIONS ******************************************/
/*                                                                   */
/*01*MEMBER-NAME: GDEASCH                                            */
/*                                                                   */
/*02* DESCRIPTIVE-NAME: SAMPLE ASCH START PARAMETER STATEMENTS       */
/*                      NECESSARY TO RUN DFSMS DISTRIBUTED FILEMANAGER*/
/*                                                                   */
/*01* DISCLAIMER =                                                   */
/*                                                                   */
/*   THIS SAMPLE SOURCE IS PROVIDED FOR TUTORIAL PURPOSES ONLY. A    */
/*   COMPLETE HANDLING OF ERROR CONDITIONS HAS NOT BEEN SHOWN OR     */
/*   ATTEMPTED, AND THIS SOURCE HAS NOT BEEN SUBMITTED TO FORMAL IBM */
/*   TESTING. THIS SOURCE IS DISTRIBUTED ON AN 'AS IS' BASIS         */
/*   WITHOUT ANY WARRANTIES EITHER EXPRESSED OR IMPLIED.             */
/*                                                                   */
/*01* FUNCTION:                                                      */
/*     THIS PARMLIB MEMBER SETS UP A SCHEDULER CLASS.                */
/*                                                                   */
/*01* DISTRIBUTION LIBRARY: ASAMPLIB                                 */
/*                                                                   */
/*01* CHANGE-ACTIVITY:                                               */
/*                                                                   */
/*FLAG LINEITEM   FMID    DATE  ID  COMMENT                          */
/* $L0=GDEASCH   HDZ11B0 931009 DFSMS 1.2.0 DISTRIBUTED FILEMANAGER  */
/*                        SAMPLE ASCH START PARAMETERS               */
/*                                                                   */
/*********************************************************************/
CLASSADD CLASSNAME(A)
         MSGLIMIT(1000) MAX(10) MIN(1) RESPGOAL(1)
```

Figure 22. ASCH Start Parameter Statements to Run DFSMS/DFM

DFM00

System SAMPLIB member DFM00, for example, SYS1.SAMPLIB(DFM00), contains the sample shown in Figure 23 on page 58 of the startup parameters for Distributed FileManager/MVS.

```
DFM CCSID(0)
    CLOSE_CHECK_INTV(0)
    DEFER_CLOSE_TIME(0)
    LOCK_RETRY(3)
    LOCK_WAIT_INTV(20)
    LOGICAL_CACHE(1024)
    MAX_AGENT_TSKS(5)
    MAX_CONV_LOCK(5)
    RESTRICT_START(YES)
    STREAM_LRECL(8196)
    SEND_BUFFER_THRESHOLD(100)
/* Uncomment the next lines to provide allocation defaults if       */
/* non-SMS files are to be created by DFM.  Remove or comment them  */
/* out again once SMS allocation is in use.                         */
    /*  PRIMARY(100) SECONDARY(50) */
    /*  UNIT(SYSALLDA)             */
    /*  VOLUME(xxxxxx)             */
```

Figure 23. Startup Parameters for Distributed FileManager/MVS

GDELOGMD

System SAMPLIB member GDELOGMD, for example, SYS1.SAMPLIB(GDELOGMD), contains the sample shown in Figure 24 on page 59 of a VTAM logon mode table that contains the logon mode entry for Distributed FileManager/MVS.

```
*** START OF SPECIFICATIONS ******************************************
*                                                                    *
*01* MEMBER-NAME: GDELOGMD                                           *
*                                                                    *
*02* DESCRIPTIVE-NAME: SAMPLE VTAM LOGMODE TABLE                      *
*                                                                    *
*                                                                    *
* 01* DISCLAIMER =                                                   *
*                                                                    *
*    THIS SAMPLE SOURCE IS PROVIDED FOR TUTORIAL PURPOSES ONLY. A    *
*    COMPLETE HANDLING OF ERROR CONDITIONS HAS NOT BEEN SHOWN OR     *
*    ATTEMPTED, AND THIS SOURCE HAS NOT BEEN SUBMITTED TO FORMAL IBM *
*    TESTING. THIS SOURCE IS DISTRIBUTED ON AN 'AS IS' BASIS         *
*    WITHOUT ANY WARRANTIES EITHER EXPRESSED OR IMPLIED.             *
*                                                                    *
*01* FUNCTION:                                                       *
*      THIS TABLE IS AN EXAMPLE OF A VTAM LOGMODE TABLE NECESSARY    *
*      TO BE INSTALLED ON MVS HOST TO RUN DFSMS DISTRIBUTED          *
*      FILEMANAGER.                                                  *
*                                                                    *
*01* DISTRIBUTION LIBRARY: ASAMPLIB                                  *
*                                                                    *
*01* CHANGE-ACTIVITY:                                                *
*                                                                    *
*  FLAG LINEITEM  FMID    DATE  ID    COMMENT                        *
*   $L0=GDELOGMD HDZ11B0 931009 DFSMS 1.2.0 DISTRIBUTED FILEMANAGER  *
*                        SAMPLE VTAM LOGON MODE TABLE.               *
*                                                                    *
**********************************************************************
*                                                                    *
LOGMODES MODETAB
         EJECT
**********************************************************************
         TITLE 'SNASVCMG'
**********************************************************************
*        LOGMODE TABLE ENTRY FOR RESOURCES CAPABLE OF ACTING         *
*               AS LU 6.2 DEVICES                                    *
*               REQUIRED FOR LU MANAGEMENT                           *
**********************************************************************
SNASVCMG MODEENT LOGMODE=SNASVCMG,FMPROF=X'13',TSPROF=X'07',         *
               PRIPROT=X'B0',SECPROT=X'B0',COMPROT=X'D0B1',          *
               RUSIZES=X'8585',ENCR=B'0000',                         *
               PSERVIC=X'060200000000000000000300'
**********************************************************************
```

Figure 24. VTAM Logon Mode Table (Part 1 of 2)

```
         TITLE 'QPCSUPP'
**********************************************************************
*        LOGMODE TABLE ENTRY FOR RESOURCES CAPABLE OF ACTING         *
*               AS LU 6.2 DEVICES                                    *
*               REQUIRED FOR LU MANAGEMENT                           *
**********************************************************************
QPCSUPP  MODEENT LOGMODE=QPCSUPP,FMPROF=X'13',TSPROF=X'07',          *
               PRIPROT=X'B0',SECPROT=X'B0',COMPROT=X'D0B1',          *
               RUSIZES=X'8585',ENCR=B'0000',                         *
               PSERVIC=X'060200000000000000000300'
**********************************************************************
         TITLE 'APPCPCLM'
**********************************************************************
*        LOGMODE TABLE ENTRY FOR RESOURCES CAPABLE OF ACTING         *
*               AS LU 6.2 DEVICES                                    *
*               FOR PC TARGET                                        *
*   IN THIS EXAMPLE THE DEFAULT RU SIZE FOR OS/2 (1024) IS USED      *
**********************************************************************
APPCPCLM MODEENT LOGMODE=APPCPCLM,                                   *
               RUSIZES=X'8787',                                      *
               SRCVPAC=X'00',                                        *
               SSNDPAC=X'01'
**********************************************************************
         TITLE 'APPCHOST'
**********************************************************************
*        LOGMODE TABLE ENTRY FOR RESOURCES CAPABLE OF ACTING         *
*               AS LU 6.2 DEVICES                                    *
*               FOR HOST TARGET                                      *
*   IN THIS EXAMPLE RU SIZE OF 4096 IS USED                          *
**********************************************************************
APPCHOST MODEENT LOGMODE=APPCHOST,                                   *
               RUSIZES=X'8989',                                      *
               SRCVPAC=X'00',                                        *
               SSNDPAC=X'01'
         MODEEND
         END
```

Figure 24. VTAM Logon Mode Table (Part 2 of 2)

GDETPDEF

System SAMPLIB member GDETPDEF, for example, SYS1.SAMPLIB(GDETPDEF), contains the sample shown in Figure 25 on page 61 of adding the Distributed FileManager/MVS TP profile to the APPC/MVS TP profile data set.

Note: The GDEDFM job in GDETPDEF should have either no region size or a region size of 0K to contain cached stream files. LE is required to use CDRA, if LE is installed and is not in the link list, SYS1.PROCLIB(DFM) and SYS1.SAMPLIB(GDETPDEF) should be modified so their STEPLIB DD statements refer to the proper LE run time library. Refer to DFMREADM in SYS1.SAMPLIB for details. SYSOUT and CDRATRC files can be allocated as RECFM=FBA, LRECL=133, and DSORG=PS for use in diagnosing CDRA problems.

```
//******************************************************************
//* PROPRIETARY V2 STATEMENT
//* LICENSED MATERIALS - PROPERTY OF IBM
//* 5695-DF1 (C) COPYRIGHT 1994,1995  IBM CORP.
//* END PROPRIETARY V2 STATEMENT
//*
//******************************************************************
//IBMUSER1  JOB 'GDETPDEF',NOTIFY=IBMUSER,MSGCLASS=H
//******************************************************************
//*
//* GDETPDEF - MVS/APPC setup for DFM: TP definition utility
//*
//* This job invokes the APPC/MVS administration utility to add
//* the TP profile to the APPC/MVS data set.
//*
//* It consists of a single job step that adds a MVS/DFM TP
//* to SYS1.APPCTP.
//*
//* Modify the above job statement as required and,
//* optionally, make the following modifications
//* to the job itself:
//*
//*   change 'SYS1.APPCTP' to another name if required by
//*               your installation
//*   change the DFMJOB card to one suitable for your installation
//* Note that you can alter the DD statements CDRATRC and SYSOUT
//* as needed to obtain CDRA API trace output and C runtime messages.
//******************************************************************
```

Figure 25. MVS/APPC Setup for DFM: TP Definition Utility (Part 1 of 2)

```
//STEP     EXEC PGM=ATBSDFMU
//SYSPRINT DD   SYSOUT=*
//SYSSDOUT DD   SYSOUT=*
//SYSSDLIB DD   DSN=SYS1.APPCTP,DISP=SHR
//SYSIN    DD   DATA,DLM=XX
    TPDELETE
      TPNAME(ˆX'07'001)
    TPADD
      TPNAME(ˆX'07'001)
      ACTIVE(YES)
      TPSCHED_DELIMITER(##)
        CLASS(A)
        JCL_DELIMITER(ENDJCL)
//GDEDFM JOB MSGCLASS=H,MSGLEVEL=(1,1),CLASS=A
//GDEDFM  EXEC PGM=GDEISASB
//* CHANGE THE STEPLIB STATEMENT AS REQUIRED IF YOUR INSTALLATION
//* DOES NOT HAVE THE LE RUNTIME DATA SET IN ITS LINK LIST.
//*STEPLIB  DD  DSN=SYS1.SCEERUN,DISP=SHR
//*CDRATRC DD DSN=SYS1.CDRATRC2,DISP=SHR  <- CDRA API TRACE OUTPUT
//*SYSOUT  DD DSN=SYS1.CDRAOUT2,DISP=SHR  <- C RUNTIME MESSAGES
//SYSOUT   DD DUMMY                       <- C RUNTIME MESSAGES (NO-OP)
ENDJCL
##
XX
```

Figure 25. MVS/APPC Setup for DFM: TP Definition Utility (Part 2 of 2)

GDEPRTLU

System SAMPLIB member GDEPRTLU, for example, SYS1.SAMPLIB(GDEPRTLU), contains the sample shown in Figure 26 on page 63 of a partner definition for an OS/2 system.

```
*/* START OF SPECIFICATIONS ******************************************
*                                                                    *
*01* MEMBER-NAME: GDEPRTLU                                           *
*                                                                    *
*02* DESCRIPTIVE-NAME: SAMPLE VTAM PARTNER LU DEFINITION NECESSARY    *
*                                TO RUN DFSMS DISTRIBUTED FILEMANAGER *
*                                                                    *
* 01* DISCLAIMER =                                                   *
*                                                                    *
*    THIS SAMPLE SOURCE IS PROVIDED FOR TUTORIAL PURPOSES ONLY. A    *
*    COMPLETE HANDLING OF ERROR CONDITIONS HAS NOT BEEN SHOWN OR     *
*    ATTEMPTED, AND THIS SOURCE HAS NOT BEEN SUBMITTED TO FORMAL IBM *
*    TESTING. THIS SOURCE IS DISTRIBUTED ON AN 'AS IS' BASIS         *
*    WITHOUT ANY WARRANTIES EITHER EXPRESSED OR IMPLIED.             *
*                                                                    *
*01* FUNCTION: THIS LU STATEMENT IDENTIFIES THE PARTNER LU           *
*                                                                    *
*01* DISTRIBUTION LIBRARY: ASAMPLIB                                  *
*                                                                    *
*01* CHANGE-ACTIVITY:                                                *
*                                                                    *
*  FLAG LINEITEM  FMID    DATE  ID  COMMENT                          *
*   $L0=GDEPRTLU HDZ11B0 931009 DFSMS 1.2.0 DISTRIBUTED FILEMANAGER  *
*                          SAMPLE PARTNER LU DEFINITION              *
*                                                                    *
**********************************************************************
OS2PRTNR  LU   LOCADDR=0,                                            *
               ISTATUS=ACTIVE,                                       *
               MODETAB=LOGMODES,                                     *
               RESSCB=4
```

Figure 26. VTAM Partner LU Definition to Run DFSMS/DFM

Appendix B. DFMX0001

System SAMPLIB member DFMX0001, for example, SYS1.SAMPLIB(DFMX0001), contains the sample shown in Figure 27, showing how to set up a procedure for starting the DFM DataAgent.

```
//DFMX0001 JOB ,MSGCLASS=Z
//DFMX0001 PROC DFMINIT=  <,optional_procedural_parameters>
//DFMAGENT EXEC PGM=&DFMINIT     <,optional_program_parameters>
//*
//* This procedure is a sample showing how to set up a procedure
//* for starting a DFM DataAgent.
//*
//* DFM DataAgent processing requires a procedure (whose name is
//* the same as the agent name).  The procedure has to run DFMINIT
//* and DFMINIT will call a DataAgent routine.  The DataAgent routine
//* will default to the same name as the agent (or procedure) name
//* but can be the name of any executable program suitable for
//* running as a key 8 job step.
//*
//* For example, you could run this DataAgent routine with no
//* further setup by issuing the following SdU sample command
//* from a workstation:
//*      dfmacall agent x:filename dfmx0001 pgm iefbr14
//*
//* DFMINIT is DFM's DataAgent routing module and should not
//* be changed.  Other DD statements and symbolic
//* substitutions can be added as needed by the DataAgent
//* program itself.
//*
//* This example is intended to discard the output by routing
//* it to MSGCLASS of Z.  You should modify it as appropriate
//* for your installation.  If you do not purge output, the
//* operator will have to periodically have to issue the $ps
//* command.
//*
//* Add additional DD statements as required by your program.
//*
//SYSPRINT  DD  SYSOUT=*
//SYSUDUMP  DD  SYSOUT=*
//DFMX0001 PEND
//GO       EXEC DFMX0001
```

Figure 27. Starting the DFM DataAgent

Appendix C. DFMXAGNT

System SAMPLIB member DFMXAGNT, for example, SYS1.SAMPLIB(DFMXAGNT) shown in Figure 28, contains the DFM DataAgent sample routine.

```
/*********************************************************************
* PROPRIETARY V3 STATEMENT                                           *
* LICENSED MATERIALS - PROPERTY OF IBM                               *
* 5695-DF1                                                           *
* (C) COPYRIGHT 1997  IBM CORP.                                      *
* END PROPRIETARY V3 STATEMENT                                       *
*********************************************************************/

/*********************************************************************
*                                                                    *
*   $MOD(DFMXAGNT) COMP(5695-DF120)                                  *
*                                                                    *
*    MODULE NAME: DFMXAGNT                                     *
*                                                                    *
*    DESCRIPTION: DFM DataAgent Sample Routine                 *
*                                                                    *
*    STATUS: Version 1 Release 4.0 (DFSMS)                     *
*                                                                    *
*    COPYRIGHT: See the copyright statement on the previous page.*
*                                                                    *
*    FUNCTION: This module illustrates how a DFM DataAgent can       *
*      be written in C.  It gets control at file declaration time    *
*      and sets the reason code so that it will also get control     *
*      when the file declaration is deleted.  Its sole function is   *
*      to issue printf statements displaying the parameters if its   *
*      internal debug flag is set.                                   *
*                                                                    *
*      You can use the C compiler and your installation's linkedit   *
*      JCL to build the DFMXAGNT executable code on the mainframe.   *
*                                                                    *
*      You can then copy DFMX0001 to produce a proclib               *
*      member (proclib member name is also referred to as the        *
*      DFM DataAgent name).   You could then invoke the DataAgent    *
*      by means of the DFMACALL sample application provided by       *
*      SdU.  (An analogous procedure is given by the installation    *
*      sample DFMXSRTI but in this case you might call the load      *
*      module DFMXAGNT and copy DMFX0001 to build PROCLIB(DFMXAGNT)  *
*      so that "dfmacall agent x:filename dfmxagent" could be used   *
*      to invoke it from the workstation.)                           *
*                                                                    *
*      An installation sample is not provided in order to            *
*      demonstrate that the manual process as outlined above is      *
*      straight-forward and because this sample would probably not   *
*      be used without significant changes anyway.                   *
*                                                                    *
```

Figure 28. DFM DataAgent Sample Routine (Part 1 of 5)

```
*    PROCESSING:                                               *
*                                                                    *
*      LOGIC:                                                  *
*        Refer to block comments in the code.                        *
*                                                                    *
*      ERROR PROCESSING:                                       *
*        Issue a printf and then return with register 15 set to a    *
*        non-zero value and with the reason code in the extended     *
*        parameter list set to a unique value.                       *
*                                                                    *
*    NOTES:                                                    *
*                                                                    *
*      PATCH SPACE: None                                             *
*      XAX CONSIDERATIONS: AMODE(31) RMODE(ANY) ENV(PRI)             *
*      DEPENDENCIES: The Language Environment and the C runtime      *
*        library must be installed.                                  *
*      RESTRICTIONS: None                                            *
*      REGISTER CONVENTIONS: Standard conventions--refer to C        *
*        compiler documentation.                                     *
*      SERIALIZATION: No serialization techniques are used by this   *
*        module.                                                     *
*                                                                    *
*    MODULE TYPE: Procedure                                          *
*      PROCESSOR: C                                                  *
*      ATTRIBUTES:                                                   *
*        TYPE:            Reentrant                                  *
*        PRIMARY ASID:    Caller's ASID                             *
*        SECONDARY ASID:  Same as primary                            *
*        HOME ASID:       Same as primary                            *
*        MODE:            Task                                       *
*        KEY:             8                                          *
*        STATE:           Problem program                            *
*        LOCATION:        Link library                               *
*                                                                    *
*    ENTRY POINT: main                                               *
*                                                                    *
*      PURPOSE: Show that a DataAgent routine can be written in C.   *
*      LINKAGE: Called by Distributed FileManager.                   *
*      INPUT/OUTPUT: Refer to the DFM Guide and Reference for the    *
*           parameter list format.                                   *
*                                                                    *
*    MESSAGES: Refer to printf statements.                           *
*                                                                    *
*    EXIT NORMAL:                                                    *
*      RETURN CODE: Register 15 = 0                                  *
*        REASON CODE: Not applicable                                 *
*         MESSAGE ID: None:                                          *
```

Figure 28. DFM DataAgent Sample Routine (Part 2 of 5)

```
*                                                                    *
*    EXIT ERRORS:                                                    *
*      RETURN CODE: Register 15 = 8                                  *
*        REASON CODE: Unique values set in the extended parameter    *
*           list                                                     *
*         MESSAGE ID: See printf statements.                         *
*                                                                    *
*    EXTERNAL REFERENCES: None                                       *
*                                                                    *
*    CHANGE ACTIVITY:                                                *
*   $L0=DFSMS14,HDZ11D0,960628,SJPLMMR: DFM DataAgent initial code   *
*********************************************************************/
#pragma   csect (static, "DFMXAGN")
#include  <stdio.h>
#include  <stdlib.h>
#include  <stddef.h>
#include  <ctype.h>
#include  <string.h>

#pragma   csect (code, "DFMXAGNT")


int main(argc, argv)
  int argc;                  /* count of input parameters           */
  char  **argv;              /* input parameters                    */
{
  int i;                     /* loop counter                        */
  char dsname??(55??);       /* data set name area                  */
  int debug = 1;             /* debug flag                          */

  /*****************************************************************/
  /* Define additional parameter for DataAgent special processing   */
  /*****************************************************************/
  _Packed struct extra_parms {
    short int extra_parms_len;     /* length of extra parms         */
    /***************************************************************/
    /* Basic section of the extra parameter structure              */
    /***************************************************************/
    unsigned short int reserved;   /* reserved field                */
    unsigned short int command_cp; /* command code point            */
    unsigned short int object_cp;  /* object code point             */
    unsigned short int ofn_len;    /* original filename length      */
               char ofn??(54??);   /* original filename             */
    unsigned short int cfn_len;    /* current/modified filename len */
               char cfn??(54??);   /* current/modified filename     */
    signed long int reason_code1;  /* main reason code              */
    signed long int reason_code2;  /* secondary reason code         */
```

Figure 28. DFM DataAgent Sample Routine (Part 3 of 5)

```
  /***************************************************************/
  /* Any additions to the extra parameter structure in           */
  /* future DFSMS releases would go here.                        */
  /***************************************************************/
} ep_area;                /* extra parameters for DataAgent      */
/* Define extra parameter instance                               */
struct extra_parms  *p_extra;

/*****************************************************************/
/* Begin DataAgent routine processing.                           */
/*****************************************************************/
if (debug) {
  /***************************************************************/
  /* Display input parameters for debugging purposes.            */
  /***************************************************************/
  printf ("DFMXAGNT: DataAgent routine entered.\n");
  if (argc > 0) {
    /*************************************************************/
    /* Display standard parameters and the program name.         */
    /*************************************************************/
    printf("  \n");
    printf ("Parameters passed were the following:\n");
    {
    for (i = 1; i < argc; i++)
      printf ("   %s\n",argv??(i??));
    }
    /*************************************************************/
    /* DataAgent has access to an extra parameter list in addition*/
    /* to the standard format MVS parameter list. This parameter  */
    /* list is defined by the structure agent_parms.              */
    /*************************************************************/
    /* Locate the extra parameters.                               */
    /*************************************************************/

    DFMXLPRM("DFMXAGNT",&p_extra);
    if (p_extra == NULL) {
      /***********************************************************/
      /* No parameters--something went wrong.                    */
      /***********************************************************/
      printf ("DFMXAGNT: No DataAgent parameters!!\n");
      return 8;                /* exit with error                */
    }
```

Figure 28. DFM DataAgent Sample Routine (Part 4 of 5)

```
      if (p_extra->extra_parms_len >= sizeof(ep_area)) {
        /**********************************************************/
        /* Print the basic section of the extra parameter list.     */
        /**********************************************************/
        printf ("   \n");
        printf ("Extra DataAgent parameters were:\n");
        printf ("   DDM command code point: %X\n",p_extra->command_cp);
        printf ("   DDM object code point: %X\n",p_extra->object_cp);
        printf ("   DDM current filename length: %d\n",
                                  p_extra->cfn_len);
        printf ("   DDM current filename: %s\n",
                 strncpy(dsname, p_extra->cfn, p_extra->cfn_len));
        printf ("   DDM original filename length: %d\n",
                                  p_extra->ofn_len);
        printf ("   DDM original filename: %s\n",
                 strncpy(dsname, p_extra->ofn, p_extra->ofn_len));
        /**********************************************************/
        /* Force recall of the exit (for DELDCL, etc.)              */
        /**********************************************************/
        p_extra->reason_code1 = -1;
      }                       /* End, basic section exists          */
    }                         /* End, parameters exist              */
    else {
      /************************************************************/
      /* No parameters--something went wrong.                       */
      /************************************************************/
      printf ("DFMXAGNT: No parameters?\n");
      p_extra->reason_code1 = 2;  /* set reason code                */
      return 8;                   /* exit with error                */
    }                         /* End of missing parameters          */
  }                           /* End of input parameter display     */
  /**************************************************************/
  /* Any further DataAgent routine processing would go here.      */
  /**************************************************************/
  return 0;
}
```

Figure 28. DFM DataAgent Sample Routine (Part 5 of 5)

Appendix D. DFMXSORT

System SAMPLIB member DFMXSORT, for example, SYS1.SAMPLIB(DFMXSORT) shown in Figure 29, shows how a DFM DataAgent can be written in assembler language to invoke SORT.

```
          TITLE 'DFMXSORT - DFM DataAgent Sort Sample'
*/*********************************************************************
*/*PROPRIETARY V3 STATEMENT                                           *
*/*LICENSED MATERIALS - PROPERTY OF IBM                               *
*/*5695-DF1                                                           *
*/*(C) COPYRIGHT 1997  IBM CORP.                                      *
*/*END PROPRIETARY V3 STATEMENT                                       *
*/*********************************************************************
*/*********************************************************************
*/*                                                                   *
*/* $MOD(DFMXSORT) COMP(5695-DF120)                                   *
*/*                                                                   *
*/*  MODULE NAME: DFMXSORT                                            *
*/*                                                                   *
*/*  DESCRIPTION: DFM DataAgent Sample Routine (SORT)                 *
*/*                                                                   *
*/*  STATUS: Version 1 Release 4.0 (DFSMS)                            *
*/*                                                                   *
*/*  COPYRIGHT: See copyright statement on previous page              *
*/*                                                                   *
*/*  FUNCTION: This module illustrates how a DFM DataAgent can        *
*/*    be written in assembler language to invoke SORT.   It is      *
*/*    implemented with a basic function that assumes the input      *
*/*    filename matches the SORTIN DD statement.  It changes the     *
*/*    filename from SORTIN to SORTOUT so that any retrieval through *
*/*    DFM will retrieve the sorted data rather than the original.   *
*/*                                                                   *
*/*    Code commentary is provided to show how the                    *
*/*    function could be extended to use the input filename for      *
*/*    dynamic allocation and to derive an output filename.  The      *
*/*    commentary also discusses how this DataAgent could request     *
*/*    that it be called again at file close time to access and       *
*/*    possibly write the output file’s data to the original file.   *
*/*                                                                   *
*/*  PROCESSING:                                                      *
*/*                                                                   *
*/*    LOGIC:                                                         *
*/*      Refer to block comments in the code.                         *
*/*                                                                   *
*/*    ERROR PROCESSING:                                              *
*/*      Issue a WTO and then return with register 15 set to a        *
*/*      non-zero value and with the reason code set to a unique      *
*/*      value.  SORT error messages will be in the JOBLOG.           *
*/*                                                                   *
```

Figure 29. DFM DataAgent Sort Sample (Part 1 of 7)

```
*/*  NOTES:                                                         *
*/*                                                                 *
*/*    PATCH SPACE: None                                            *
*/*    XAX CONSIDERATIONS: AMODE(31) RMODE(ANY) ENV(PRI)            *
*/*    DEPENDENCIES: None                                           *
*/*    RESTRICTIONS: None                                           *
*/*    REGISTER CONVENTIONS: Standard                               *
*/*    SERIALIZATION: No serialization techniques are used by this  *
*/*      module.                                                    *
*/*                                                                 *
*/*  MODULE TYPE: Procedure                                         *
*/*    PROCESSOR: z/OS Assembler                                  *
*/*    ATTRIBUTES:                                                  *
*/*      TYPE:            Not reentrant                             *
*/*      PRIMARY ASID:    Caller's ASID                             *
*/*      SECONDARY ASID:  Same as primary                           *
*/*      HOME ASID:       Same as primary                           *
*/*      MODE:            Task                                      *
*/*      KEY:             8                                         *
*/*      STATE:           Problem program, non-APF-authorized       *
*/*      LOCATION:        Link library                              *
*/*                                                                 *
*/*  ENTRY POINT: DFMXSORT                                          *
*/*                                                                 *
*/*    PURPOSE: Show that a DataAgent routine can invoke SORT.      *
*/*    LINKAGE: Called by Distributed FileManager.                  *
*/*    INPUT: Refer to the DFM Guide and Reference for a general    *
*/*       description of the parameter list format.                 *
*/*                                                                 *
*/*  MESSAGES: Refer to WTO statements.                             *
*/*                                                                 *
*/*  EXIT NORMAL:                                                   *
*/*                                                                 *
*/*    RETURN CODE: Register 15 = 0                                 *
*/*      REASON CODE: Not applicable                                *
*/*       MESSAGE ID: None:                                         *
*/*                                                                 *
*/*  EXIT ERRORS:                                                   *
*/*                                                                 *
*/*    RETURN CODE: Register 15 = 8                                 *
*/*      REASON CODE: Unique values set in the extended parameter   *
*/*         list                                                    *
*/*       MESSAGE ID: See WTO statements.                           *
*/*                                                                 *
*/*  EXTERNAL REFERENCES: None                                      *
*/*                                                                 *
*/*  CHANGE ACTIVITY:                                               *
*/* $L0=DFSMS14,HDZ11D0,960628,SJPLMMR: DFM DataAgent initial code   *
*/*******************************************************************
DFMXSORT AMODE 31
DFMXSORT RMODE ANY
DFMXSORT CSECT
```

Figure 29. DFM DataAgent Sort Sample (Part 2 of 7)

```
*********************************************************************
*   This program is used as a DataAgent routine to get control       *
*   when a workstation’s SdU application opens a remote MVS file     *
*   with a filename suffix specifying "agent(dfmxsort)".  The file is *
*   assumed to be in the format of the sample file created earlier.   *
*   The input file is sorted and the filename is then changed to the  *
*   filename of the sort output file.  This results in the SdU        *
*   application’s retrieving a sorted subset of the records as if     *
*   they came from the original file.                                 *
*                                                                     *
*   An alternative invocation method is provided by SdU through       *
*   the DFMACALL sample application.  Refer to it for details.        *
*                                                                     *
*   The reason code can be set to -1 to force entry at the file’s     *
*   delete declaration time when adding extended function.            *
*********************************************************************
R0       EQU   0
R1       EQU   1
R2       EQU   2
R3       EQU   3
R4       EQU   4
R5       EQU   5
R6       EQU   6
R7       EQU   7
R8       EQU   8
R9       EQU   9
R10      EQU   10
RTN      EQU   10
R11      EQU   11
R12      EQU   12
R13      EQU   13
R14      EQU   14
R15      EQU   15
         SAVE  (14,12)       SAVE REGISTERS
         BALR  12,0          BRANCH AND LINK REG.
         USING *,12          USE REG 12
         ST    13,SAVEAREA+4 SAVE BACKWARD POINTER
         LA    14,SAVEAREA   SET FORWARD PT.ER IN CALLER SAVE AREA
         ST    14,8(13)
         LR    13,14         SET OUR SAVE AREA
*------------------------------------------------------------------*
*      Determine whether we are defining or deleting the file      *
*      declaration (i.e. previous to OPEN or after CLOSE).         *
*------------------------------------------------------------------*
         LR    R3,R1         Save original parameter pointers
         SR    R15,R15       Clear error code
         USING INPARMS,R3    Address of MVS parameter list
         L     R4,EXTPARMP   Point to DFM DataAgent parameters
         USING EXTPARMS,R4   Base of DFM DataAgent parameters
         CLC   EXTOBJCP,FILNAM  Is file being processed?
         BNE   EXIT          No, exit
```

Figure 29. DFM DataAgent Sort Sample (Part 3 of 7)

```
*-------------------------------------------------------------------*
*      Called for a file—see whether declaration or delete       *
*      declaration.                                                 *
*-------------------------------------------------------------------*
         CLC   EXTCMDCP,DCLFIL  Is file being declared?
         BE    DODCLFIL      Yes, process DCLFIL.
         CLC   EXTCMDCP,DELDCL  Is file declaration being deleted?
         BE    DODELDCL      Yes, process DELDCL.
*-------------------------------------------------------------------*
*      Unknown command type                                         *
*-------------------------------------------------------------------*
         WTO   'DFMXSORT: Unknown command code.'
         LA    R15,12
         B     EXIT
*
*-------------------------------------------------------------------*
*      DCLFIL Processing                                            *
*-------------------------------------------------------------------*
DODCLFIL EQU   *
*-------------------------------------------------------------------*
* Enhanced function:                                                *
*    Set sort input filename from input filename.                   *
*    Set sort output filename to input filename.                    *
*    If output filename is greater than 40 characters then do.      *
*      Locate last component of filename.                           *
*      If last component of filename is < 4 then                    *
*        Locate last 2 components of filename.                      *
*      If last component(s) are equal to ".SRT" then                *
*        Replace last component(s) with ".SR2".                     *
*      Else                                                         *
*        Replace last component(s) with ".SRT".                     *
*    end.                                                           *
*    Else                                                           *
*      Append ".SRT" to output filename.                            *
*    Allocate sort input as DISP=SHR.                               *
*    Allocate sort output as DISP=(NEW,CATLG).                      *
*-------------------------------------------------------------------*
*-------------------------------------------------------------------*
*      Invoke DFSORT with 31-bit parameter list                     *
*-------------------------------------------------------------------*
         LR    R1,R3              Fetch address of std parm list
         MVC   EXTPARMP,=F'-1'    End of list
         LINK  EP=SORT            Invoke DFSORT
         LTR   R15,R15            Check for SORT failure
         BNZ   SORTERR            Branch if error
*
*-------------------------------------------------------------------*
* Enhanced function:                                                *
*    Set modified filename generated for output file allocation.    *
*    Set reason code to -1 to force recall for DELDCL.              *
*-------------------------------------------------------------------*
*
```

Figure 29. DFM DataAgent Sort Sample (Part 4 of 7)

```
*-----------------------------------------------------------------*
*    Return modified filename and filename length to DFM.         *
*-----------------------------------------------------------------*
         LH    R2,EXTOFNLN         Get original (input) name length
         CH    R2,MAXIFNLN         Ensure maximum is not exceeded
         BH    OFNISOK             Branch—file name is left alone
         CH    R2,MINIFNLN         Ensure minimum is met
         BL    OFNISOK             Branch—file name is left alone
         LA    R1,EXTOFN           Point to beginning of orig fn
         AR    R1,R2               Point to end
         LH    R15,MINIFNLN        Get length of trigger in name
         SR    R1,R15              Backup to where trigger appears
         BCTR  R15,0               Decrement for execute
         EX    R15,COMPNAME        Compare last part of name
         BNE   OFNISOK             Branch—no trigger at name end
*    Original file name meets qualifications—modify file name.
         LA    R3,EXTMFN           Set pointer to modified file name.
         LR    R1,R2               Set length of filename.
         SH    R1,MINIFNLN         Backup to trigger (=root end)
         BCTR  R1,0                Decrement for execute
         EX    R1,MOVENAME         Move input name beginning to output
         LA    R1,1(R1)            Restore length
         AR    R3,R1               Point to end of root
         LH    R15,MODIFLEN        Get length of modifier
         BCTR  R15,0               Decrement for execute
         EX    R15,CHANGENM        Change name to root + modifier
         SH    R2,MINIFNLN         Decrement trigger length
         AH    R2,MODIFLEN         Add modifier length
         STH   R2,EXTMFNLN         Set output name length
OFNISOK  EQU   *                   Here if changed/no change to do
         SR    R15,R15             Exit with no error
         B     EXIT                Exit DataAgent DCLFIL routine
COMPNAME CLC   0(0,R1),TRIGGER     Compare name to trigger string
MOVENAME MVC   EXTMFN(0),EXTOFN    Move original name to modified name
CHANGENM MVC   0(0,R3),MODIFIER    Move modifier string to end of name
*
*-----------------------------------------------------------------*
*      SORT error occurred—refer to JOBLOG for details.          *
*-----------------------------------------------------------------*
SORTERR  EQU   *
         LR    R2,R15              Save SORT code
         ST    R0,EXTRSNC2         Save SORT reason code
         WTO   'DFMXSORT: SORT failure.'
         LR    R15,R2              Exit with error
         B     EXIT                Exit DataAgent routine
*
```

Figure 29. DFM DataAgent Sort Sample (Part 5 of 7)

```
*-------------------------------------------------------------------*
*      DELDCL Processing                                            *
*-------------------------------------------------------------------*
DODELDCL EQU   *
         WTO   'DFMXSORT: Can delete IBMUSER.DFMXSORT.SORTOUT now.'
*-------------------------------------------------------------------*
* Enhanced function:                                                *
*   Copy the changes to the permanent file.                         *
*-------------------------------------------------------------------*
         SR    R15,R15             Exit with no error
         B     EXIT                Exit with return code from Sort
*-------------------------------------------------------------------*
*      Exit                                                         *
*-------------------------------------------------------------------*
EXIT     EQU   *
         L     13,4(,13)     GET RETURN ADDRESS
         RETURN (14,12),RC=(15) RESTORE REGS,FLAG SAVEAREA,SET RC
SAVEAREA DC    18F'00'
         LTORG
*
*-------------------------------------------------------------------*
*    Define input parameters.                                       *
*-------------------------------------------------------------------*
*    In this case, the standard format MVS parameter list should be*
*    a halfword length field followed by a DFSORT extended          *
*    parameter list.  For example, Sort by ascending characters in *
*    columns 17-22 is:                                              *
*         SORT FIELDS=(17,6,CH,A)                                   *
*-------------------------------------------------------------------*
*
*-------------------------------------------------------------------*
*    Constants for use with the DFM DataAgent extended parameters. *
*-------------------------------------------------------------------*
DCLFIL   DC    X'102C'       Declare file command code point
DELDCL   DC    X'102D'       Delete declare file command code point
DRCNAM   DC    X'1165'       Directory is being declared
FILNAM   DC    X'110E'       File is being declared
*-------------------------------------------------------------------*
*    Local Constants                                                *
*-------------------------------------------------------------------*
MAXIFNLN DC    H'53'         Maximum allowing for trigger->modifier
MINIFNLN DC    H'6'          Minimum orig name len (= len of trigger)
TRIGGER  DC    CL6'SORTIN'   Trigger in input filename
MODIFLEN DC    H'7'          Length of modifier
MODIFIER DC    CL7'SORTOUT'  Modifier for output filename
```

Figure 29. DFM DataAgent Sort Sample (Part 6 of 7)

```
*-------------------------------------------------------------------*
*    Parameter list pointers.                                       *
*-------------------------------------------------------------------*
INPARMS  DSECT
STDPARMP DS    A(0)          Ptr to standard format MVS parameter list
EXTPARMP DS    A(0)          Ptr to extended DFM parameter list
*
*-------------------------------------------------------------------*
*    Standard MVS parameter list for SORT usage.                    *
*-------------------------------------------------------------------*
STDPARMS DSECT
STDPARML DS    H             Length of parameters
STDPARMC DS    CL256         Standard parameter string
*
*
*-------------------------------------------------------------------*
*    Extended parameter list unique to DFM.                         *
*-------------------------------------------------------------------*
EXTPARMS DSECT
EXTPARML DS    H             Length of parameters
         DS    H             Reserved
EXTCMDCP DS    H             Command code point
EXTOBJCP DS    H             Object code point
EXTOFNLN DS    H             Original filename length
EXTOFN   DS    CL54          Original filename
EXTMFNLN DS    H             Modified filename length
EXTMFN   DS    CL54          Modified filename
EXTRSNC1 DS    F             Reason code 1
EXTRSNC2 DS    F             Reason code 2
*
         END   DFMXSORT
```

Figure 29. DFM DataAgent Sort Sample (Part 7 of 7)

Appendix E. DFMXSRTI

System SAMPLIB member DFMXSRTI, for example, SYS1.SAMPLIB(DFMXSRTI) shown in Figure 30, shows how to install the DFMXSORT DataAgent routine.

```
//DFMXSRTI   JOB  ,'DFMXSORT SETUP',MSGLEVEL=(1,1),MSGCLASS=A,
//             USER=IBMUSER,PASSWORD=IBM,REGION=1M
//*******************************************************************/
//*PROPRIETARY V3 STATEMENT
//*LICENSED MATERIALS - PROPERTY OF IBM
//*5695-DF1
//*(C) COPYRIGHT 1997  IBM CORP.
//*END PROPRIETARY V3 STATEMENT
//*******************************************************************/
//* Sample installation for DFMXSORT DataAgent routine.  Modify the
//* job statement, etc. as appropriate for your installation.
//*******************************************************************/
//* This sample uses &PREFIX to generate input and output filenames.
//* For example, &PREFIX.SORTIN will be used for input and
//* &PREFIX.SORTOUT will be used for output.
//*
//* A sample input file, IBMUSER.DFMXSORT.SORTIN, is also provided.
//*
//* It could be cloned with other filenames and sort parameters
//* or could be generalized to use dynamic allocation for the
//* input and output files.  Pseudocode is included to illustrate
//* the types of changes that would be required.  If changes are
//* made, then you must remove DD statements for SORTIN and
//* SORTOUT from the cataloged procedure.
//*
//* Once this installation job has run, logon to a workstation with
//* the prerequisite level of SdU and with an APPC connection to the
//* mainframe and issue the following command:
//*    dfmacall agent x:ibmuser.dfmxsort.sortin
//*                'dfmxsort,prefix=ibmuser.dfmxsort'
//*                parm 'sort fields=(17,6,ch,a)'
//*
//* Refer to DFMACALL for details.
//*
//*******************************************************************/
```

Figure 30. Sample Installation for the DFMXSORT DataAgent Routine (Part 1 of 4)

```
//*******************************************************************/
//* Clean up test data sets.  Remove this step from production version.
//*******************************************************************/
//CLEANUP        EXEC PGM=IDCAMS
//SYSPRINT            DD SYSOUT=*
//SYSIN               DD *
    DELETE ( -
             IBMUSER.DFMXSORT.*   -
            ) NONVSAM PURGE SCRATCH
    IF LASTCC = 8 THEN DO
      SET LASTCC = 0
      SET MAXCC = 0
    END
/*


//*******************************************************************/
//*            Generate SORTIN and allocate SORTOUT                 */
//*******************************************************************/
//STEP1    EXEC PGM=IEBGENER
//SYSUT2   DD
DSN=IBMUSER.DFMXSORT.SORTIN,DISP=(NEW,CATLG),UNIT=SYSDA,
//             DCB=(LRECL=80,BLKSIZE=0,RECFM=FB),
//             SPACE=(TRK,(1,1))
//SORTOUT  DD  DSN=IBMUSER.DFMXSORT.SORTOUT,DISP=(NEW,CATLG),
//
UNIT=SYSDA,DCB=(LRECL=80,BLKSIZE=0,RECFM=FB),
//             SPACE=(TRK,(1,1))
//*  Create records with a 6 character sort field in column 17
//*  and with a flag in column 40 that is used to omit (=1) or
//*  include the record.  The program itself contains the parameters
//*  for the sort (ascending on columns 17-22) and the control file
//*  contains the definition of what records to include/omit.
//SYSUT1   DD   DATA
RECORD NUMBER = 000030, OMIT FLAG IS = 1   .............................
RECORD NUMBER = 000888, OMIT FLAG IS = 0   .............................
RECORD NUMBER = 000887, OMIT FLAG IS = 1   .............................
RECORD NUMBER = 099999, OMIT FLAG IS = 0   .............................
RECORD NUMBER = 100000, OMIT FLAG IS = 1   .............................
RECORD NUMBER = 100001, OMIT FLAG IS = 0   .............................
RECORD NUMBER = 111111, OMIT FLAG IS = 1   .............................
RECORD NUMBER = 111110, OMIT FLAG IS = 1   .............................
RECORD NUMBER = 211111, OMIT FLAG IS = 0   .............................
RECORD NUMBER = 999999, OMIT FLAG IS = 1   .............................
RECORD NUMBER = 000000, OMIT FLAG IS = 0   .............................
/*
//SYSPRINT DD   SYSOUT=*
//SYSIN    DD   DUMMY
```

Figure 30. Sample Installation for the DFMXSORT DataAgent Routine (Part 2 of 4)

```
//*********************************************************************
//* Assemble the DataAgent Sample DFMXSORT
//*********************************************************************
//ASM01   EXEC PGM=IEV90,PARM='OBJECT,NODECK'
//SYSPRINT DD  SYSOUT=*
//SYSLIB   DD  DSN=SYS1.MACLIB,DISP=SHR
//SYSUT1   DD  UNIT=SYSDA,SPACE=(1700,(600,100))
//SYSUT2   DD  UNIT=SYSDA,SPACE=(1700,(600,100))
//SYSUT3   DD  UNIT=SYSDA,SPACE=(1700,(600,100))
//SYSLIN   DD  DSN=&&DFMXSORT,DISP=(,PASS),
//             UNIT=SYSDA,SPACE=(TRK,(2,2,2))
//SYSIN    DD  DSN=SYS1.SAMPLIB(DFMXSORT),DISP=SHR
//********************************************************************
//* Link Edit DataAgent Routine DFMXSORT                             *
//********************************************************************
//LINK1   EXEC
PGM=IEWL,PARM='XREF,LET,LIST,AMODE=31,RMODE=ANY'
//SYSPRINT DD  SYSOUT=*
//OBJ      DD  DSN=&&DFMXSORT,DISP=(OLD,DELETE)
//SYSLMOD  DD  DSN=SYS1.LINKLIB(DFMXSORT),DISP=SHR
//SYSUT1   DD  UNIT=SYSDA,SPACE=(TRK,(80,10))
//SYSLIN   DD  *
          INCLUDE OBJ
          ENTRY DFMXSORT
          NAME  DFMXSORT(R)
/*
//*********************************************************************
//* Build Agent JCL in SYS1.PROCLIB
//*********************************************************************
//STEP1    EXEC PGM=IEBGENER
//SYSUT2   DD   DISP=SHR,DSN=SYS1.PROCLIB(DFMXSORT)
//SYSUT1   DD   DATA
//DFMXSORT JOB  ,MSGCLASS=A
//*********************************************************************
//* Set appropriate msgclass above for debug vs production
//*********************************************************************
//DFMXSORT PROC DFMINIT=,PREFIX=IBMUSER.DFMXSORT
//DFMAGENT EXEC PGM=&DFMINIT
//*********************************************************************
//* Run DFMXSORT DataAgent Sample
//*
//* Sort input comes from IBMUSER.DFMXSORT.SORTIN by default.
//* Sorted data goes into IBMUSER.DFMXSORT.SORTOUT by default.
//* You should modify the names, space allocation, etc. as appropriate
//* for your installation.
```

Figure 30. Sample Installation for the DFMXSORT DataAgent Routine (Part 3 of 4)

```
//*********************************************************************
//* Add STEPLIB statements as appropriate for your installation.
//*STEPLIB  DD  DSN=...,DISP=SHR
//*         DD  DSN=SYS1.LINKLIB,DISP=SHR
//SYSOUT   DD  SYSOUT=*
//SORTIN   DD  DSN=&PREFIX..SORTIN,DISP=SHR
//SORTOUT  DD  DSN=&PREFIX..SORTOUT,DISP=SHR
//SORTCNTL DD  DSN=SYS1.PROCLIB(DFMXSORI),DISP=SHR
//DFMXSORT PEND
//GO       EXEC DFMXSORT
/*
//SYSPRINT DD   SYSOUT=*
//SYSIN    DD   DUMMY
//*********************************************************************
//* Build omit control statement in SYS1.PROCLIB
//*********************************************************************
//STEP2    EXEC PGM=IEBGENER
//SYSUT2   DD   DISP=SHR,DSN=SYS1.PROCLIB(DFMXSORI)
//SYSUT1   DD   DATA
* The following omit statement will ensure that only the
* records without a 1 in column 40 appear in the sortout data set
 OMIT COND=(40,4,CH,EQ,C'1')
/*
//SYSPRINT DD   SYSOUT=*
//SYSIN    DD   DUMMY
```

Figure 30. Sample Installation for the DFMXSORT DataAgent Routine (Part 4 of 4)

Appendix F. DFMQTSO

System SAMPLIB member DFMQTSO, for example, SYS1.SAMPLIB(DFMQTSO) shown in Figure 31, shows how a DFM DataAgent can be written in assembler language to invoke TSO. The DFMQTSO routine links to the IKJTSOEV function.

```
          TITLE 'DFMQTSO - DFM DataAgent TSO Sample'
*/*********************************************************************
*/*PROPRIETARY V3 STATEMENT                                           *
*/*LICENSED MATERIALS - PROPERTY OF IBM                               *
*/*5695-DF1                                                           *
*/*(C) COPYRIGHT 1997  IBM CORP.                                      *
*/*END PROPRIETARY V3 STATEMENT                                       *
*/*********************************************************************
*/*********************************************************************
*/*                                                                   *
*/* $MOD(DFMQTSO) COMP(5695-DF120)                                    *
*/*                                                                   *
*/*  MODULE NAME: DFMQTSO (Quick TSO-Input in PARM)                   *
*/*                                                                   *
*/*  DESCRIPTION: DFM DataAgent Sample Routine (TSO)                  *
*/*                                                                   *
*/*  STATUS: Version 1 Release 4.0 (DFSMS)                            *
*/*                                                                   *
*/*  COPYRIGHT: See copyright statement on previous page              *
*/*                                                                   *
*/*  FUNCTION: This module illustrates how a DFM DataAgent can        *
*/*    be written in assembler language to invoke TSO.                *
*/*                                                                   *
*/*    SdU provides a sample application, DFMACALL, that can be       *
*/*    to invoke this sample.  Refer to it for details.               *
*/*                                                                   *
*/*    Refer to 'TSO Extensions for MVS: Programming Services' for    *
*/*    information about using the TSO environment service used in    *
*/*    this example.                                                  *
*/*                                                                   *
*/*  PROCESSING:                                                      *
*/*                                                                   *
*/*    LOGIC:                                                         *
*/*      Refer to block comments in the code.                         *
*/*                                                                   *
*/*    ERROR PROCESSING:                                              *
*/*      Issue a WTO and then return with register 15 set to a        *
*/*      non-zero value and with the reason code set to a unique      *
*/*      value.                                                       *
*/*                                                                   *
```

Figure 31. DFM DataAgent Sample Routine (TSO) (Part 1 of 7)

```
*/*  NOTES:                                                         *
*/*                                                                 *
*/*    PATCH SPACE: None                                            *
*/*    XAX CONSIDERATIONS: AMODE(31) RMODE(ANY) ENV(PRI)            *
*/*    DEPENDENCIES: None                                           *
*/*    RESTRICTIONS: None                                           *
*/*    REGISTER CONVENTIONS: Standard                               *
*/*    SERIALIZATION: No serialization techniques are used by this  *
*/*      module.                                                    *
*/*                                                                 *
*/*  MODULE TYPE: Procedure                                         *
*/*    PROCESSOR: z/OS Assembler                                    *
*/*    ATTRIBUTES:                                                  *
*/*      TYPE:            Not reentrant                             *
*/*      PRIMARY ASID:    Caller's ASID                             *
*/*      SECONDARY ASID:  Same as primary                           *
*/*      HOME ASID:       Same as primary                           *
*/*      MODE:            Task                                      *
*/*      KEY:             8 (Current task TCBPKF=jobstep task TCBPKF)*
*/*      STATE:           Problem program, non-APF-authorized       *
*/*      LOCATION:        Link library                              *
*/*                                                                 *
*/*  ENTRY POINT: DFMQTSO                                           *
*/*                                                                 *
*/*    PURPOSE: Show that a DataAgent routine can invoke TSO.       *
*/*    LINKAGE: Called by Distributed FileManager.                  *
*/*    INPUT: Refer to the DFM Guide and Reference for a general    *
*/*       description of the parameter list format.  Refer to       *
*/*       DFMACALL documentation for command line invocation from   *
*/*       SdU.                                                      *
*/*                                                                 *
*/*  MESSAGES: Refer to WTO statements.                             *
*/*                                                                 *
*/*  EXIT NORMAL:                                                   *
*/*                                                                 *
*/*    RETURN CODE: Register 15 = 0                                 *
*/*      REASON CODE: Not applicable                                *
*/*       MESSAGE ID: None:                                         *
*/*                                                                 *
*/*  EXIT ERRORS:                                                   *
*/*                                                                 *
*/*    RETURN CODE: Register 15 = non-zero                          *
*/*      REASON CODE: Unique values set in the extended parameter   *
*/*         list                                                    *
*/*       MESSAGE ID: See WTO statements.                           *
*/*                                                                 *
*/*  EXTERNAL REFERENCES: None                                      *
*/*                                                                 *
*/*  CHANGE ACTIVITY:                                               *
*/* $L0=DFSMS14,HDZ11D0,960628,SJPLMMR: DFM DataAgent initial code   *
*/******************************************************************
```

Figure 31. DFM DataAgent Sample Routine (TSO) (Part 2 of 7)

```
DFMQTSO  CSECT
DFMQTSO  AMODE 31
DFMQTSO  RMODE ANY
         STM   R14,R12,12(R13)
         BALR  R12,0
         USING *,R12
         ST    R13,SAVEAREA+4
         LA    R11,SAVEAREA
         ST    R11,8(,R13)
         LA    R13,SAVEAREA
         LR    R3,R1          Save original parameter pointers
         SR    R15,R15        Clear error code
         USING INPARMS,R3     Address of MVS parameter list
         L     R4,STDPARMP    Point to DFM DataAgent parameters
         USING STDPARMS,R4    Base of DFM DataAgent parameters
         L     R5,EXTPARMP    Point to DFM DataAgent parameters
         USING EXTPARMS,R5    Base of DFM DataAgent parameters
*------------------------------------------------------------------*
*      Called for a file or directory.  See whether declaration    *
*      is being created or deleted.                                *
*------------------------------------------------------------------*
         CLC   EXTCMDCP,DCLFIL  Is file being declared?
         BE    DODCLFIL       Yes, process DCLFIL.
         CLC   EXTCMDCP,DELDCL  Is file declaration being deleted?
         BE    DODELDCL       Yes, process DELDCL.
*------------------------------------------------------------------*
*      Unknown command type                                        *
*------------------------------------------------------------------*
         WTO   'DFMQTSO: Unknown command code.'
         LA    R15,16          Exit without trying to set reason code
         B     EXIT
*
*------------------------------------------------------------------*
*      DCLFIL Processing (DCLFIL, or Declare File, is a DDM,       *
*      Distributed Data Management, command issued when a remote   *
*      file or directory is about to be opened)                    *
*------------------------------------------------------------------*
DODCLFIL EQU   *
         WTO   'DFMQTSO: Declaring a file.'
*********************************************************************
* CALTSOEV - CALL THE TSO/E ENVIRONMENT SERVICE TO ESTABLISH A TSO/E
*            ENVIRONMENT IN THIS PROGRAM'S ADDRESS SPACE.
*   PARM1 IS RESERVED
*   PARM2 IS A FULLWORD THAT WILL CONTAIN THE RETURN CODE FROM IKJTSOEV
*   PARM3 IS A FULLWORD THAT WILL CONTAIN THE REASON CODE ON RETURN
*         FROM IKJTSOEV.
*   PARM4 IS A FULLWORD THAT WILL CONTAIN THE ABEND CODE, IF AN ABEND
*         OCCURS DURING TSO/E ENVIRONMENT SERVICE PROCESSING.
*   PARM5 IS A FULLWORD THAT WILL CONTAIN THE ADDRESS OF THE CPPL.
*********************************************************************
CALTSOEV DS    0H
         XC    PARM1,PARM1
         LINK  EP=IKJTSOEV,ERRET=LE,PARAM=(PARM1,PARM2,PARM3,PARM4,PARM*
               5),VL=1
```

Figure 31. DFM DataAgent Sample Routine (TSO) (Part 3 of 7)

```
***********************************************************************
* CHKEVRC - CHECK THE RETURN CODE FROM IKJTSOEV
***********************************************************************
CHKEVRC   DS     0H
          L      R2,PARM2
          LTR    R2,R2
          BNZ    BADEVRC
***********************************************************************
* TSO Environment established—process the input file.
***********************************************************************
***********************************************************************
* CALLTSR - Call IKJEFTSR to process the input file or the parameter
*           list—depending on whether a parameter list is present.
*           The output from the commands will go to the SYSTSPRT file.
***********************************************************************
CALLTSR  DS    0H
         LH    R2,STDPARML         Get length of input parameters
         LTR   R2,R2               If zero then use the input file
         BZ    USEINFIL
         ST    R2,BUFLEN           Set buffer length
         BCTR  R2,0
         EX    R2,COPYCMD          Copy command to parameter area
         L     R15,CVTPTR
         L     R15,CVTTVT(,R15)
         L     R15,TSVTASF-TSVT(,R15)
         CALL  (15),(FLAGS,CMDBUFF,BUFLEN,RETCODE,RSNCODE,ABNDCODE),VL
***********************************************************************
* DOALL - At this point, process the return values from
*         IKJEFTSR and the invoked functions.
***********************************************************************
DOALL    DS    0H
         LTR   R15,R15
         BZ    EXIT              Exit if no error
         C     R15,=F'4'         Did CLIST or REXX exec fail?
         BNE   SAVERC            No, just ensure RC is saved
         MVC   RSNCODE,RETCODE   Use CLIST/REXX RC as reason code
SAVERC   ST    R15,RETCODE       Ensure a retcode is set
         WTO   'DFMQTSO: IKJEFTSR failed.'
         L     R15,RETCODE       Set return code
         MVC   EXTRSNC1,RSNCODE  Set reason code 1
         MVC   EXTRSNC2,ABNDCODE Set reason code 2 to ABEND code
         B     EXIT
COPYCMD  MVC   CMDBUFF(0),STDPARMC
*
```

Figure 31. DFM DataAgent Sample Routine (TSO) (Part 4 of 7)

```
*-------------------------------------------------------------------*
*      SYSTSIN Processing                                           *
*-------------------------------------------------------------------*
USEINFIL EQU   *
         WTO   'DFMQTSO: SYSTSIN input not supported. Use DFMXTSO.'
         LA    R15,8
         B     EXIT
*
*-------------------------------------------------------------------*
*      DELDCL Processing (DELDCL, or Delete Declaration, is a DDM, *
*      Distributed Data Management, command issued after a remote  *
*      file or directory has been closed)                           *
*-------------------------------------------------------------------*
DODELDCL EQU   *
         WTO   'DFMQTSO: Deleting file declaration.'
         SR    R15,R15             Exit with no error
         B     EXIT                Exit with return code from Sort
***********************************************************************
* LE      - Branch here if LINK failed.  The ABEND code will be in
*           register 1 but there will be no reason code.  Set the
*           return code to 8 and use the ABEND code as the reason code.
***********************************************************************
*
LE       DS    0H
         ST    R1,EXTRSNC1         Save the ABEND code
         WTO   'DFMQTSO: LINK to IKJTSOEV failed.'
         LA    R15,8
         B     EXIT
*
***********************************************************************
* BADEVRC - Branch here if IKJTSOEV returned a non-zero return code.
*           If the program branches here, it will exit with an error.
*           In the DFM diagnostics, the error data will be as follows:
*              RETURN CODE   - THE RETURN CODE FROM IKJTSOEV
*              REASON CODE 1 - THE REASON CODE FROM IKJTSOEV
*              REASON CODE 2 - THE ABEND  CODE FROM IKJTSOEV
***********************************************************************
BADEVRC  DS    0H
         WTO   'DFMQTSO: IKJTSOEV error occurred.'
         L     R15,PARM2
         MVC   EXTRSNC1,PARM3
         MVC   EXTRSNC2,PARM4
         B     EXIT
```

Figure 31. DFM DataAgent Sample Routine (TSO) (Part 5 of 7)

```
***********************************************************************
* EXIT - RETURN TO CALLING PROGRAM
***********************************************************************
EXIT     DS    0H
         L     R13,4(,R13)
         RETURN (14,12),RC=(15) RESTORE REGS,FLAG SAVEAREA,SET RC
*
* REGISTER EQUATES
R1       EQU   1
R2       EQU   2
R3       EQU   3
R4       EQU   4
R5       EQU   5
R11      EQU   11
R12      EQU   12
R13      EQU   13
R14      EQU   14
R15      EQU   15
* PARAMETERS USED TO INVOKE THE TSO/E ENVIRONMENT SERVICE
PARM1    DS    F                       RESERVED FIELD
PARM2    DS    F                       RETURN CODE FIELD
PARM3    DS    F                       REASON CODE FIELD
PARM4    DS    F                       FUNCTION ABEND CODE
PARM5    DS    F                       CPPL ADDRESS
* PARAMETERS USED TO INVOKE THE TSO SERVICE FACILITY
FLAGS    DS    0F                   FULLWORD OF FLAGS
RESFLAGS DC    H'0001'              ESTABLISH UNAUTHORIZED ENVIRONMENT
ABFLAGS  DC    X'01'                PRODUCE A DUMP IF FUNCTION ABENDS
FNCFLAGS DC    X'01'                INVOKE TSO/E CMD, REXX EXEC, CLIST
CMDBUFF  DS    256C                 COMMAND BUFFER
BUFLEN   DS    F                    LENGTH OF COMMAND BUFFER
RETCODE  DS    F                    FUNCTION RETURN CODE
RSNCODE  DS    F                    FUNCTION REASON CODE
ABNDCODE DS    F                    FUNCTION ABEND CODE
CVTPTR   EQU   16                   THESE 2 PARMS ARE USED TO GET
CVTTVT   EQU   X'9C'                ADDR OF THE TSO SERVICE FACILITY
* SAVEAREA AND OTHER PROGRAM STORAGE
SAVEAREA DS   18F
* TSVT MAPPING MACRO (USED TO GET THE ADDRESS OF TSO SERVICE FACILITY)
         IKJTSVT
DFMQTSO  CSECT
         LTORG
*
*-------------------------------------------------------------------*
*    Define input parameters.                                       *
*-------------------------------------------------------------------*
*    The standard format MVS parameter list is a halfword length   *
*    field followed by a parameter list.  In this case the         *
*    parameter list is a TSO command.                              *
*-------------------------------------------------------------------*
*
```

Figure 31. DFM DataAgent Sample Routine (TSO) (Part 6 of 7)

```
*-----------------------------------------------------------------*
*    Constants for use with the DFM DataAgent extended parameters. *
*-----------------------------------------------------------------*
DCLFIL   DC    X'102C'       Declare file command code point
DELDCL   DC    X'102D'       Delete declare file command code point
DRCNAM   DC    X'1165'       Directory is being declared
FILNAM   DC    X'110E'       File is being declared
*-----------------------------------------------------------------*
*    Parameter list pointers.                                     *
*-----------------------------------------------------------------*
INPARMS  DSECT
STDPARMP DS    A(0)          Ptr to standard format MVS parameter list
EXTPARMP DS    A(0)          Ptr to extended DFM parameter list
*
*-----------------------------------------------------------------*
*    Standard MVS parameter list for SORT usage.                  *
*-----------------------------------------------------------------*
STDPARMS DSECT
STDPARML DS    H             Length of parameters
STDPARMC DS    CL256         Standard parameter string
*
*
*-----------------------------------------------------------------*
*    Extended parameter list unique to DFM.                       *
*-----------------------------------------------------------------*
EXTPARMS DSECT
EXTPARML DS    H             Length of parameters
         DS    H             Reserved
EXTCMDCP DS    H             Command code point
EXTOBJCP DS    H             Object code point
EXTOFNLN DS    H             Original filename length
EXTOFN   DS    CL54          Original filename
EXTMFNLN DS    H             Modified filename length
EXTMFN   DS    CL54          Modified filename
EXTRSNC1 DS    F             Reason code 1
EXTRSNC2 DS    F             Reason code 2
*
          END    DFMQTSO
```

Figure 31. DFM DataAgent Sample Routine (TSO) (Part 7 of 7)

Appendix G. DFMXTSOI

System SAMPLIB member DFMXTSOI, for example, SYS1.SAMPLIB(DFMXTSOI) shown in Figure 32, shows how to install the two DataAgent routines: DFMXTSO and DFMQTSO.

```
//DFMXTSOI   JOB  ,'DFMXTSO SETUP',MSGLEVEL=(1,1),MSGCLASS=A,
//             USER=IBMUSER,PASSWORD=IBM,REGION=1M
//*******************************************************************/
//*PROPRIETARY V3 STATEMENT
//*LICENSED MATERIALS - PROPERTY OF IBM
//*5695-DF1
//*(C) COPYRIGHT 1997  IBM CORP.
//*END PROPRIETARY V3 STATEMENT
//*******************************************************************/
//* Install two TSO DataAgent routines, DFMXTSO and DFMQTSO by
//* installing two procedures DFMXTSO and DFMQTSO and by producing
//* a load module for DFMQTSO.  No executable is produced for
//* procedure (or DataAgent) DFMXTSO because it uses the standard
//* TSO batch program, IKJEFT01.
//*
//* DFMXTSO is full function in that it is intended to be used with
//* IKJEFT01 which accepts input from SYSTSIN.  DFMQTSO (or quick
//* TSO) only accepts input from the PARM() parameter.
//*
//* Once this installation job has run, DFMACALL TSO or DFMACALL QTSO
//* can be run from a workstation that has SdU installed and that has
//* an APPC connection to the mainframe.  Refer to DFMACALL for
//* details.  (Note that you might want to set up some typical
//* SYSTSIN files and preallocate several SYSTSPRT files for the
//* anticipated users.)
//*
```

Figure 32. Sample Installation for the DFMXTSO and DFMQTSO DataAgent Routines (Part 1 of 7)

```
//*****************************************************************/
//*  General setup                                                */
//*****************************************************************/
//*****************************************************************/
//* Clean up old SYSTSIN and SYSTSPRT files.                      */
//*   (Remove or modify if copying this to production JCL.)       */
//*****************************************************************/
//CLEANUP        EXEC PGM=IDCAMS
//SYSPRINT            DD SYSOUT=*
//SYSIN               DD *
    DELETE ( -
             IBMUSER.DFMXTSO.SYSTSIN    -
            ) NONVSAM PURGE SCRATCH
    DELETE ( -
             IBMUSER.DFMQTSO.SYSTSPRT   -
            ) NONVSAM PURGE SCRATCH
    DELETE ( -
             IBMUSER.DFMXTSO.SYSTSPRT   -
            ) NONVSAM PURGE SCRATCH
    IF LASTCC = 8 THEN
      DO
        SET LASTCC = 0
        SET MAXCC = 0
      END
```

Figure 32. Sample Installation for the DFMXTSO and DFMQTSO DataAgent Routines (Part 2 of 7)

```
/*
//*****************************************************************/
//*   Allocate output files                                       */
//*   (Remove or modify if copying this to production JCL.)       */
//*****************************************************************/
//ALLOCATE EXEC PGM=IEFBR14
//ALLOC1  DD  DSN=IBMUSER.DFMQTSO.SYSTSPRT,DISP=(NEW,CATLG),UNIT=SYSDA,
//            DCB=(DSORG=PS,LRECL=121,BLKSIZE=0,RECFM=FBA),
//            SPACE=(TRK,(15,5))
//ALLOC2  DD  DSN=IBMUSER.DFMXTSO.SYSTSPRT,DISP=(NEW,CATLG),UNIT=SYSDA,
//            DCB=(DSORG=PS,LRECL=121,BLKSIZE=0,RECFM=FBA),
//            SPACE=(TRK,(15,5))
//*****************************************************************/
//*   Generate sample input file for full function DFMXTSO        */
//*   (Remove or modify if copying this to production JCL.)       */
//*****************************************************************/
//GENINPUT EXEC PGM=IEBGENER
//SYSUT2   DD  DSN=IBMUSER.DFMXTSO.SYSTSIN,DISP=(NEW,CATLG),UNIT=SYSDA,
//             DCB=(LRECL=80,BLKSIZE=0,RECFM=FB),
//             SPACE=(TRK,(1,1))
//SYSUT1   DD   DATA
     /* This is a sample input file for SYSTSIN.             */
     LISTC
/*
//SYSPRINT DD   SYSOUT=*
//SYSIN    DD   DUMMY
//*
```

Figure 32. Sample Installation for the DFMXTSO and DFMQTSO DataAgent Routines (Part 3 of 7)

```
//*******************************************************************/
//* 1. Setup for the DFMXTSO DataAgent procedure.   Modify the job
//* statement, etc. as appropriate for your installation.
//*******************************************************************/
//* This setup job installs a sample procedure, DFMXTSO, to allow
//* workstations to invoke TSO.
//*
//* The procedure is intended to be used with PGM(IKJEFT01) and will
//* accept input from both the PARM field (processed first) and
//* from userID.DFMXTSO.SYSTSIN.
//*
//* It uses userID.DFMXTSO.SYSTSPRT to contain the TSO output.
//*
//* Note that SYSTSPRT allocation is DISP=SHR.
//* This causes SYSTSPRT to be reset each invocation
//* of TSO so the output produced by a file declaration
//* will be overlain by a subsequent delete file declaration.
//* Therefore, end users or applications will have to use the
//* SYSTSPRT file contents before closing the file replaces it.
//* The SYSTSPRT allocation could also be changed to DISP=MOD to
//* cause appending of output.
//*
//* You must create the SYSTSPRT or SYSTSIN files that
//* will be needed before invoking the exit.  If parameters are
//* to be passed through the PARM field only, SYSTSIN could
//* be changed to DD DUMMY or the input file could be cleared.
//*
//* The exit must be invoked with at least the following
//* parameters:   ...,AGENT(DFMXTSO,U=userID),PARM(...)
//*
//* (DFM/MVS will automatically provide the U=userID parameter)
//*
//*******************************************************************/
```

Figure 32. Sample Installation for the DFMXTSO and DFMQTSO DataAgent Routines (Part 4 of 7)

```
//*******************************************************************
//* Build Agent JCL in SYS1.PROCLIB
//*******************************************************************
//GENPROC1 EXEC PGM=IEBGENER
//SYSPRINT DD   SYSOUT=*
//SYSIN    DD   DUMMY
//SYSUT2   DD   DISP=SHR,DSN=SYS1.PROCLIB(DFMXTSO)
//SYSUT1   DD   DATA
//DFMXTSO JOB   ,MSGCLASS=A
//*******************************************************************
//* Set appropriate msgclass above for debug vs production
//* Note that DFM will provide the DFMINIT parameter and the
//* U parameter (high-level name qualifier or userID).
//*******************************************************************
//DFMXTSO PROC DFMINIT=,U=
//DFMAGENT EXEC PGM=&DFMINIT.,
//         PERFORM=2,
//         REGION=5000K,
//         DYNAMNBR=20
//*******************************************************************
//* Run DFMXTSO DataAgent Sample
//*******************************************************************
//* Add STEPLIB statements as appropriate for your installation.
//* Note that if IKJEFT01 was installed into LPALIB, a STEPLIB will
//* be required because DFM's DataAgent processing uses the BLDL
//* function.
//*STEPLIB  DD  DSN=...,DISP=SHR
//*         DD  DSN=SYS1.LINKLIB,DISP=SHR
//*         DD  DSN=SYS1.LPALIB,DISP=SHR   <-- See note above
//*
//* TSO/E input comes from the SYSTSIN file (as well as from PARM).
//*
//* TSO/E output goes to the SYSTSPRT file.
//*
//* Sample using a generic CLIST...
//*SYSPROC  DD   DSN=&U..CLIST.CLIST,DISP=SHR
//SYSPROC  DD   DSN=SYS1.CLIST,DISP=SHR
//SYSTSIN  DD   DSN=&U..DFMXTSO.SYSTSIN,DISP=SHR
//SYSTSPRT DD   DSN=&U..DFMXTSO.SYSTSPRT,DISP=SHR
//SYSOUT   DD   SYSOUT=*
//DFMXTSO PEND
//GO       EXEC DFMXTSO
/*
```

Figure 32. Sample Installation for the DFMXTSO and DFMQTSO DataAgent Routines (Part 5 of 7)

```
//******************************************************************/
//* 2. Setup for the DFMQTSO DataAgent procedure.   Modify the job
//* control statements, etc. as appropriate for your installation.
//******************************************************************/
//* This setup job installs a sample procedure, DFMQTSO, to allow
//* workstations to invoke program DFMQTSO which will, in turn, invoke
//* IKJTSOEV to establish the TSO environment and then call TSO to
//* process the input from the PARM field.  It ignores file
//* userID.DFMXTSO.SYSTSIN.
//*
//* Like DFMXTSO, it uses userID.DFMXTSO.SYSTSPRT to contain
//* the TSO output.
//*
//* The exit must be invoked with at least the following
//* parameters:   ...,AGENT(DFMQTSO,U=userID),PARM(...)
//*
//* (DFM/MVS will automatically provide the U=userID parameter)
//*
//******************************************************************/
//*******************************************************************
//* Assemble the Quick TSO Sample DataAgent Routine, DFMQTSO
//*******************************************************************
//ASM01   EXEC PGM=IEV90,PARM='OBJECT,NODECK'
//SYSPRINT DD  SYSOUT=*
//SYSLIB   DD  DSN=SYS1.MACLIB,DISP=SHR
//SYSUT1   DD  UNIT=SYSDA,SPACE=(1700,(600,100))
//SYSUT2   DD  UNIT=SYSDA,SPACE=(1700,(600,100))
//SYSUT3   DD  UNIT=SYSDA,SPACE=(1700,(600,100))
//SYSLIN   DD  DSN=&&DFMQTSO,DISP=(,PASS),
//             UNIT=SYSDA,SPACE=(TRK,(2,2,2))
//SYSIN    DD  DSN=SYS1.SAMPLIB(DFMQTSO),DISP=SHR
//******************************************************************
//* Link Edit DataAgent Routine DFMQTSO                            *
//******************************************************************
//LINK1   EXEC PGM=IEWL,PARM='XREF,LET,LIST,AMODE=31,RMODE=ANY'
//SYSPRINT DD  SYSOUT=*
//OBJ      DD  DSN=&&DFMQTSO,DISP=(OLD,DELETE)
//SYSLMOD  DD  DSN=SYS1.LINKLIB(DFMQTSO),DISP=SHR
//SYSUT1   DD  UNIT=SYSDA,SPACE=(TRK,(80,10))
//SYSLIN   DD  *
          INCLUDE OBJ
          ENTRY DFMQTSO
          NAME  DFMQTSO(R)
/*
```

Figure 32. Sample Installation for the DFMXTSO and DFMQTSO DataAgent Routines (Part 6 of 7)

```
//*********************************************************************
//* Build DFMQTSO Agent JCL in SYS1.PROCLIB
//*********************************************************************
//GENPROC2 EXEC PGM=IEBGENER
//SYSPRINT DD   SYSOUT=*
//SYSIN    DD   DUMMY
//SYSUT2   DD   DISP=SHR,DSN=SYS1.PROCLIB(DFMQTSO)
//SYSUT1   DD   DATA
//DFMQTSO JOB   ,MSGCLASS=A
//*********************************************************************
//* Set appropriate msgclass above for debug vs production
//* Note that DFM will provide the DFMINIT parameter and the
//* U parameter (high-level name qualifier or userID).
//*********************************************************************
//DFMQTSO PROC DFMINIT=,U=
//DFMAGENT EXEC PGM=&DFMINIT.,
//         PERFORM=2,
//         REGION=5000K,
//         DYNAMNBR=20
//*********************************************************************
//* Run DFMQTSO DataAgent Sample
//*********************************************************************
//* Add STEPLIB statements as appropriate for your installation.
//*STEPLIB  DD  DSN=...,DISP=SHR
//*         DD  DSN=SYS1.LINKLIB,DISP=SHR
//*
//* Sample shown using a global CLIST library.
//*SYSPROC  DD   DSN=&U..CLIST.CLIST,DISP=SHR
//SYSPROC  DD   DSN=SYS1.CLIST,DISP=SHR
//*
//* TSO/E output goes to the SYSTSPRT file.
//*
//SYSTSPRT DD   DSN=&U..DFMQTSO.SYSTSPRT,DISP=SHR
//*
//* Note that the program DFMQTSO does not use SYSTSIN for input
//* so a dummy SYSTSIN DD statement is provided.
//*
//SYSTSIN  DD   DUMMY
//SYSOUT   DD   SYSOUT=*
//DFMQTSO PEND
//GO       EXEC DFMQTSO
/*
```

Figure 32. Sample Installation for the DFMXTSO and DFMQTSO DataAgent Routines (Part 7 of 7)

Appendix H. System PROCLIB Member DFM

System PROCLIB member DFM, for example, SYS1.PROCLIB(DFM), contains the sample shown in Figure 33 of the startup procedures for Distributed FileManager/MVS. See “Activating Distributed FileManager/MVS in System PROCLIB” on page 46.

Note: LE is required to use CDRA, if LE is installed and is not in the link list, SYS1.PROCLIB(DFM) and SYS1.SAMPLIB(GDETPDEF) should be modified so their STEPLIB DD statements refer to the proper LE runtime library. Refer to DFMREADM in SYS1.SAMPLIB for details.

```
//DFM     PROC  PARMS='NORMAL'
//***************************************************************
//*                                                             *
//*  DFSMS MVS/DFM START UP PROCEDURE                           *
//*                                                             *
//***************************************************************
//DFM      EXEC PGM=GDEISBOT,
//             PARM='&PARMS&sssq;,
//             REGION=0K,
//             TIME=1440
//IEFPARM  DD   DSN=SYS1.PARMLIB,DISP=SHR
//* CHANGE THE STEPLIB STATEMENT AS REQUIRED IF YOUR INSTALLATION
//* DOES NOT HAVE THE LE RUNTIME DATA SET IN ITS LINK LIST.
//*STEPLIB  DD  DSN=SYS1.SCEERUN,DISP=SHR
//SYSPRINT DD   SYSOUT=*
//SYSUDUMP DD   SYSOUT=*
//*
//*  THE TWO FILES ASSOCIATED WITH THE DD STATEMENTS CDRATRC AND
//*  SYSOUT CAN BE USED TO DIAGNOSE DFM STARTUP PROBLEMS RELATED
//*  TO CDRA. (CDRA IS INVOKED DURING STARTUP FOR CERTAIN CCSID
//*  VALUES IN THE SYS1.PARMLIB MEMBER DFM00.)
//*
//*  YOU MUST ALLOCATE THE TWO FILES AS RECFM=FBA, LRECL=133,
//*  AND DSORG=PS BEFORE STARTING DFM WITH THE DD STATEMENTS
//*  ACTIVE.
//*
//*  NOTE THAT SYSOUT IS REQUIRED AND CDRATRC IS OPTIONAL
//*  WHEN USING CDRA AND THE DEFAULT INSTALLATION IS SET UP TO
//*  USE CDRA IF YOUR HOST CODE PAGE IS OTHER THAN 500.
//*
//*  CDRATRC  DD  DSN=SYS1.CDRATRC,DISP=SHR    CDRA API TRACING
//*  SYSOUT   DD  DSN=SYS1.CDRAOUT,DISP=SHR    C RUNTIME MESSAGES
//SYSOUT   DD  DUMMY <- DEFAULT = CDRA WITH RUNTIME MESSAGES DISCARDED
```

Figure 33. DFSMS MVS/DFM Startup Procedure

Appendix I. PPT Entries for Distributed FileManager/MVS

Figure 34 shows the PPT entries for Distributed FileManager/MVS (see "Verifying PPT Entries for Distributed FileManager/MVS" on page 47).

```
PPT PGMNAME(GDEISASB)  /* DFM CONVERSATION ADDRESS SPACE    */
          CANCEL       /* PROGRAM CAN BE CANCELLED          */
          KEY(5)       /* PROTECT KEY IS 5                  */
          SWAP         /* PROGRAM IS SWAPPABLE              */
          NOPRIV       /* PROGRAM IS NON PRIVILEGED         */
          DSI          /* REQUIRED DATASET INTEGRITY        */
          SYST         /* PROGRAM IS A SYSTEM TASK          */
          PASS         /* CANNOT BYPASS PASSWORD PROTECTION */
          AFF(NONE)    /* NO CPU AFFINITY                   */
          NOPREF       /* NO PREFERRED STORAGE FRAMES       */

PPT PGMNAME(GDEISBOT)  /* DFM SYSTEM INITIALIZATION         */
          CANCEL       /* PROGRAM CAN BE CANCELLED          */
          KEY(5)       /* PROTECT KEY IS 5                  */
          NOSWAP       /* PROGRAM IS NON SWAPPABLE          */
          NOPRIV       /* PROGRAM IS NON PRIVILEGED         */
          DSI          /* REQUIRED DATASET INTEGRITY        */
          SYST         /* PROGRAM IS A SYSTEM TASK          */
          PASS         /* CANNOT BYPASS PASSWORD PROTECTION */
          AFF(NONE)    /* NO CPU AFFINITY                   */
          NOPREF       /* NO PREFERRED STORAGE FRAMES       */

PPT PGMNAME(GDEICASB) /* DFM CENTRAL ADDRESS SPACE          */
          CANCEL    /* PROGRAM CAN BE CANCELLED             */
          KEY(5)    /* PROTECT KEY ASSIGNED IS FIVE         */
          NOSWAP    /* PROGRAM IS NON SWAPPABLE             */
          NOPRIV    /* PROGRAM IS NON PRIVILEGE             */
          DSI       /* REQUIRES DATA SET INTEGRITY          */
          SYST      /* PROGRAM IS A SYSTEM TASK             */
          PASS      /* CANNOT BYPASS PASSWORD PROTECTION    */
          AFF(NONE) /* NO CPU AFFINITY                      */
          NOPREF    /* NO PREFERRED STORAGE FRAMES          */
```

Figure 34. PPT Entries for Distributed FileManager/MVS

Appendix J. DFMACALL.C

The DataAgent sample in Figure 35 demonstrates the ability to invoke DFM DataAgent functions from C applications on workstations running SmartData Utilities (SdU) on OS/2. The sample may need to be modified for your application and your platform. If modified, rename and compile it on a workstation using the header files distributed with SmartData Utilities (SdU). This sample is not included in SYS1.SAMPLIB.

```
/***************************************************************************
****************************  DFMACALL.C  *********************************
***************************************************************************
*
*                          DFM DataAgent Sample
*
* Module Name: DFMACALL.C
*
* DDM Workstation Application
*
* Version: 1.0
* Release: 1.0
*
* Copyright (C)
* International Business Machines Corporation 1997
*
* DISCLAIMER OF WARRANTIES: The following (enclosed) code is sample code
* created by the IBM Corporation.  This sample code is not a part of any
* IBM product and is provided to you solely for the purpose of assisting
* you in the development of your applications.  The code is provided
* "AS IS", without warranty of any kind.  IBM shall not be liable for any
* damages arising out of your use of the sample code, even if they have
* been advised of the possibility of such damages.
*
* The sample program does the following:
*
* 1) Construct a filename and filename suffix from the input parameters.
* 2) Do a DDMOpen for the file or directory to trigger MVS suffix
*    processing.
* 3) Do a DDMClose for the file or directory to terminate processing.
*
*
* COMMAND LINE INVOCATION:
*
*   This sample can be invoked in the following formats:
*
*  DFMACALL QTSO  driveletter:          TSOcommandline  [DISPLAY]
*  DFMACALL TSO   driveletter:          [TSOcommandline]  [DISPLAY]
*  DFMACALL AGENT driveletter:[filename]
MVSprocedure[,procedural_parameters]
*         [PGM program_name] [PARM program-parameters]
[DISPLAY]
*  DFMACALL START driveletter:        MVSprocedure[,procedural_parameters]
*  DFMACALL       driveletter:filename[,filename_suffix]  [DISPLAY]
*
```

Figure 35. DFM DataAgent Sample (Part 1 of 25)

```
* The last format is free-form in which MVS parameters can be specified
* in the filename suffix.  Parameters that are relevant to DFM
* DataAgent processing are the following:
*   AGENT(agentname) - Specifies the name of a procedure in SYS1.PROCLIB
*         that provides the JCL for agent processing and, if PGM is
*         omitted, the name of the DataAgent routine (program) to run.
*         Note that procedural parameters can also be specifed.  For
*         example, AGENT(agentname,USER=userID,DSNAME=DS1,...).  If
*         you use the free-form format, remember to specify PARM also.
*   PGM(programname) - Specifies the name of the DataAgent routine.
*   PARM(program_name) - Specifies input parameters to the DataAgent
*                        routine.
*   START(procedurename,procedure parameters) - Specifies the name of an
*         MVS procedure to be started asynchronously.
*   DISPLAY - Displays the result of the call to DFM DataAgent.  In the case of
*         QTSO or TSO the result is the SYSTSPRT file.  In the case of other
*         DataAgents it is the output name returned by the DataAgent routine
*         after successful invocation.
*         See DFMXSORT for an example.
*         (Note that DISPLAY can be used as a DataAgent name with DFMACALL
*         but not as a TSO command.)
*
*   Examples:
*
*     dfmacall r:ibmuser.a.b
*        ==> Opens and closes MVS file ibmuser.a.b on remote drive r.
*
*     dfmacall r:ibmuser.a.b,agent(dfmxagnt)
*        ==> Opens and closes MVS file ibmuser.a.b invoking agent dfmxagnt.
*
*     dfmacall r:ibmuser.a.b,agent(dfmx0001),pgm(dfmxagnt)
*        ==> Opens and closes MVS file ibmuser.a.b invoking agent dfmx0001
*        ==> with program dfmxagnt.
*
*     dfmacall agent r:ibmuser.a.b dfmxagnt
*        ==> Opens and closes MVS file ibmuser.a.b invoking agent dfmxagnt
*        ==> with default program dfmxagnt and null parameters implied.
*
*     dfmacall agent r:ibmuser.a.b dfmxtso pgm ikjeft01
*        ==> Opens and closes MVS file ibmuser.a.b invoking agent dfmxtso
*        ==> with (APF-authorized) program ikjeft01.  (Equivalent to
*        ==> DFMACALL TSO.)
*
```

Figure 35. DFM DataAgent Sample (Part 2 of 25)

```
*     dfmacall start r: dfmx0001,dfminit=iefbr14
*        ==> Asynchronously starts procedure dfmx0001 with parameter
*        ==> dfminit set to iefbr14.
*
*     dfmacall qtso   r:  listc   display
*        ==> Calls TSO to list catalog entries and place the results
*        ==> in IBMUSER.DFMQTSO.SYSTSPRT.  No SYSTSIN input file is
*        ==> expected.  The SYSTSPRT file is displayed.
*
*     dfmacall tso   r:
*        ==> Calls TSO to process input file IBMUSER.DFMXTSO.SYSTSIN
*        ==> and put the results in IBMUSER.DFMXTSO.SYSTSPRT.
*
*     dfmacall tso   r:    "profile prefix(ibmuser)"  display
*        ==> After running the command passed it (in this case "profile")
*        ==> it calls TSO to process input file IBMUSER.DFMXTSO.SYSTSIN
*        ==> and put the results in IBMUSER.DFMXTSO.SYSTSPRT. Display the
*        ==> output file.
*
*
*
***************************************************************************/

#include  <os2.h>                    /* required for VSAM/X applications */
#include  <stdio.h>
#include  <string.h>
#include  <memory.h>
#include  <malloc.h>
#include  "dub.h"      /* required master include for VSAM/X applications */

/*-------------------------------------------------------------------------
--                        SYMBOLIC CONSTANTS
-------------------------------------------------------------------------*/
#define FILCLS_SIZE sizeof(OBJLENGTH) + (2 * sizeof(CODEPOINT))
#define FILCLS_NAME ".DDM_FILCLS"
#define RECDATALEN  100
#define RPYMSBFLN   546                    /* reply message buffer length   */
#define PATHLEN     300                    /* path with 45 extra bytes      */
#define PARMLEN     255                    /* arbitrary maximum parm len    */
#define USPARMLEN   255                    /* arbitrary max user parm len   */
#define MINPQTSO    4                      /* minimum parameters for QTSO   */
#define MAXPQTSO    5                      /* maximum parameters for QTSO   */
#define MINPTSO     3                      /* minimum parameters for TSO    */
#define MAXPTSO     5                      /* maximum parameters for TSO    */
#define MINPAGENT   4                      /* minimum parameters for AGENT  */
##define MAXPAGENT  9                      /* maximum parameters for AGENT  */
#define MINPSTART   4                      /* minimum parameters for START  */
#define MAXPSTART   4                      /* maximum parameters for START  */
#define MINPFF      2                      /* minimum parameters for FF     */
#define MAXPFF      3                      /* maximum parameters for FF     */
```

Figure 35. DFM DataAgent Sample (Part 3 of 25)

```
/*---------------------------------------------------------------------------
--                    LOCAL FUNCTION DECLARATIONS
---------------------------------------------------------------------------*/
int  SpecialOptions(int index, int argc, char* argv[]);
int  CheckRange(int min, int max, int argc, char uarg[PARMLEN]);
VOID DumpBuffer(PDDMOBJECT pAttribute, USHORT Count);
CODEPOINT  ReplyMsg(VOID);
VOID OmitError(VOID);
VOID GeneralError(VOID);
VOID ValueError(char *value);
VOID ParmLenError(char *value);
int  strupper(char *out, char *in, int bufflen);
VOID DisplayBuffer(ULONG count, PDDMRECORD pRcdarea);
VOID DuplicateError(VOID);
VOID HasFileNameError(VOID);
VOID NoFileNameError(VOID);
VOID TooManyError(VOID);
VOID NotEnoughError(VOID);
VOID DisplayHelp(char *helpflag);

/*---------------------------------------------------------------------------
--                            DFMACALL
---------------------------------------------------------------------------*/

    int dummy_filename = 0;      /* Dummy filename flag               */
    int display_filename = 0;    /* Display filename flag             */
    int TSO_retry = 0;           /* TSO error retry flag              */
    int debug = 1;               /* Debug flag: 0 = nothing displayed,*/
                                 /* 1 = filename display, 2 = all of  */
                                 /* the above plus major functions,   */
                                 /* 5 = all the above plus data.      */
    int display_counter = 0;     /* Record display counter            */
    int intrc;                   /* internal return code              */

main(int argc, char* argv[])
{
    int i;                       /* Loop counter                      */
    int fnlen = 0;               /* Filename length                   */
    int pgmcnt = 0;              /* Number of PGM parm occurrences    */
    int parmcnt = 0;             /* Number of PARM parm occurrences   */
    int data_follows;            /* Data follows in next arg          */
    APIRET SevCode;              /* Severity code (see DUBDEFS.H)     */
    CODEPOINT  LCodePoint;       /* Local Code Point for reply msg    */
```

Figure 35. DFM DataAgent Sample (Part 4 of 25)

```
    PDDMRECORD pRecord;
    RECLENGTH  RecordSize;
    PDDMRECAL  pRecAL;
    PDDMRECALK pRecALK;
    RECLENGTH  RecALSize;
    PBYTE      pData;
    HDDMLOAD   UnLoad;            /* File handle for unload           */
    ULONG      RecCount;
    ULONG      DDMMoreDataFlag;
    int        minparms;          /* minimum parameters current cmd   */
    int        maxparms;          /* maximum parameters current cmd   */

    HDDMFILE FileHandle;

    /* Filename to be operated on                                     */
    CHAR MVSFilename[PATHLEN];
    CHAR RootName[PATHLEN];        /* Root name for TSO retry         */
    CHAR dummy_name[9] = "NULLFILE";
    CHAR display_name[PATHLEN];

    CHAR uarg[PARMLEN];            /* Upper case argument             */
    CHAR usparg[USPARMLEN];        /* Upper case subparameter         */
    /*****************************************************************/
    /* Determine which command format was used and build the MVS      */
    /* filename and filename suffix accordingly.                      */
    /*****************************************************************/

    RootName[0] = 0;               /* Set TSO root name to null string*/
    switch (argc)
    {  case 1: /* no user arguments */
          NotEnoughError();
          DisplayHelp("N");
          return(SC_SEVERE);
       default:
         /*************************************************************/
         /* 1 or more user arguments--check further                   */
         /*************************************************************/
         /* Convert current user argument to upper case.              */
         if (intrc = strupper(uarg, argv[1], PATHLEN))
           return intrc;
```

Figure 35. DFM DataAgent Sample (Part 5 of 25)

```
    if (strcmp(uarg,"QTSO") == 0 |
        strcmp(uarg,"TSO") == 0 ) {
        if (strcmp(uarg,"QTSO") == 0) {
           /*****************************************************/
           /* QTSO format                                       */
           /*****************************************************/
           minparms = MINPQTSO;
           maxparms = MAXPQTSO;
         }
         else {
           /*****************************************************/
           /* TSO format                                        */
           /*****************************************************/
           minparms = MINPTSO;
           maxparms = MAXPTSO;
         }
         if (intrc = CheckRange(minparms,maxparms,argc, uarg) > 0)
           return(intrc);

         /*******************************************************/
         /* Set special processing flags                        */
         /*******************************************************/
         for (i=minparms; i < argc; i++) {
               if (intrc = SpecialOptions(i, argc, argv) > 0)
             return(intrc);
         }
         /*******************************************************/
         /* Build filename in the format of                     */
         /*  x:fn,agent(dfmqtso,u=userID),pgm(dfmqtso),         */
         /*         parm(...)                                      */
         /*                  -- OR --                           */
         /*  x:fn,agent(dfmxtso,u=userID),pgm(ikjeft01),        */
         /*         parm(...)                                      */
         /* (Note that MVS will append u=userid)                */
         /*******************************************************/
         if (intrc = strupper(MVSFilename,argv[2],PATHLEN))
           return(intrc);
         fnlen = strlen(MVSFilename);  /* Save true filename len */
         if ( fnlen > 2 | strncmp(&MVSFilename[1],":",1) != 0  ) {
            HasFileNameError();
            DisplayHelp(&uarg[0]);
            return(SC_SEVERE);
         }
```

Figure 35. DFM DataAgent Sample (Part 6 of 25)

```
      /*******************************************************/
      /* Build root filename.                                */
      /*******************************************************/
      if (display_filename) {
        if (strcmp(uarg,"QTSO") == 0)
          strcat(MVSFilename,"DFMQTSO.SYSTSPRT");
        else
          strcat(MVSFilename,"DFMXTSO.SYSTSPRT");
        strcpy(RootName,MVSFilename);
      } else {
        dummy_filename = 1;
        strcat(MVSFilename,dummy_name);
      } /* endif */

      /*******************************************************/
      /* Attach filename suffix.                             */
      /*******************************************************/
      if (strcmp(uarg,"QTSO") == 0)
        strcat(MVSFilename,",agent(dfmqtso),parm(");
      else
        strcat(MVSFilename,",agent(dfmxtso),pgm(ikjeft01),parm(");

      /*******************************************************/
      /* Concatenate parm field                              */
      /*******************************************************/
      if (argc >= 4 ) {
        if (intrc = strupper(usparg, argv[3], USPARMLEN))
          return (intrc);
        if (strcmp(usparg,"DISPLAY") != 0)
          strcat(MVSFilename,argv[3]);
        else if (strcmp(uarg,"QTSO") == 0) {
          /* DFMQTSO requires a parameter field              */
          OmitError();
          DisplayHelp(&uarg[0]);
          return(SC_SEVERE);
        }
      }
      strcat(MVSFilename,")");   /* Terminate parameter field */

  }                               /* End of QTSO/TSO case     */
```

Figure 35. DFM DataAgent Sample (Part 7 of 25)

```
else if (strcmp(uarg,"AGENT") == 0) {
    /*****************************************************/
    /* AGENT format                                      */
    /*****************************************************/
    minparms = MINPAGENT;
    maxparms = MAXPAGENT;
    if (intrc = CheckRange(minparms,maxparms,argc, uarg) > 0)
      return(intrc);

    /*****************************************************/
    /* Set special processing flags                      */
    /*****************************************************/
    for (i=minparms; i < argc; i++) {
          if (intrc = SpecialOptions(i, argc, argv) > 0)
        return(intrc);
    }

    /*****************************************************/
    /* Build filename in the format of                   */
    /*  x:fn,agent(agentname),pgm(program_name),parm(parms) */
    /*****************************************************/
    if (intrc = strupper(MVSFilename,argv[2], PATHLEN))
      return(intrc);
    fnlen = strlen(MVSFilename);  /* Save true filename len */
    /* Add dummy filename if one wasn't specified          */
    if ( fnlen == 2  & strncmp(&MVSFilename[1],":",1) == 0  ) {
      dummy_filename = 1;
      strcat(MVSFilename,dummy_name);
    }

    /* Check the filename format for obvious errors.       */
    fnlen = strlen(MVSFilename);  /* Save true filename len */
    if (fnlen < 3 ) {
      NoFileNameError();
      DisplayHelp(&uarg[0]);
      return(SC_SEVERE);
     }

    strcat(MVSFilename,",agent(");
    strcat(MVSFilename,argv[3]);
    strcat(MVSFilename,")");
    /*****************************************************/
    /* Concatenate optional fields                       */
    /*****************************************************/
    if (argc > 4) {
      /***************************************************/
      /* Optional parameters are present                 */
      /***************************************************/
```

Figure 35. DFM DataAgent Sample (Part 8 of 25)

```
        /*****************************************************/
        /* All but DISPLAY are in format of keyword + value.  */
        /*****************************************************/
        data_follows = 0;
        for (i=4; i < argc; i++) {
          if (!data_follows) {
            /* Not data object -- process the keyword.        */
            data_follows = 1;
            if (intrc = strupper(usparg, argv[i], USPARMLEN))
              return (intrc);
            if (strcmp(usparg,"PGM") == 0) {
                 pgmcnt++;
                 strcat(MVSFilename,",pgm(");
            }
            else if (strcmp(usparg,"PARM") == 0) {
                 parmcnt++;
                 strcat(MVSFilename,",parm(");
            }
            else {                    /* Unidentified keyword?*/
              /* Make sure it's not a display option          */
               if (strcmp(usparg,"DISPLAY") != 0 ) {
                 GeneralError();
                 DisplayHelp(&uarg[0]);
                 return(SC_SEVERE);
               }
               else
                 data_follows = 0;    /* No following data    */
            }                         /* End, unidentified kwd*/
          }                           /* End, even number     */
          else  {
            /* Process the keyword's data.                    */
            data_follows = 0;
            strcat(MVSFilename,argv[i]);
            strcat(MVSFilename,")");
          }                           /* End, data field      */
        }    /* End of for loop                               */
      }
      /*******************************************************/
      /* Ensure no duplicate parameters                       */
      /*******************************************************/
      if ( pgmcnt > 1 | parmcnt > 1 ) {
        DuplicateError();
        DisplayHelp(&uarg[0]);
        return(SC_SEVERE);
      }
      /*******************************************************/
      /* Ensure AGENT is invoked with a PARM                  */
      /*******************************************************/
      if (parmcnt == 0)
        strcat(MVSFilename,",parm()");
 }                               /* End of AGENT              */
```

Figure 35. DFM DataAgent Sample (Part 9 of 25)

```
else if (strcmp(uarg,"START") == 0) {

     /*******************************************************/
     /* START format                                        */
     /*******************************************************/
     minparms = MINPSTART;
     maxparms = MAXPSTART;
     if (intrc = CheckRange(minparms,maxparms,argc, uarg) > 0)
       return(intrc);

     /*******************************************************/
     /* Build filename in the format of                     */
     /*     x:fn,start(proc,parms)                          */
     /*******************************************************/
     if (intrc = strupper(MVSFilename,argv[2],PATHLEN))
       return(intrc);
     fnlen = strlen(MVSFilename);
     if ( (fnlen > 2) | (strncmp(&MVSFilename[1],":",1) != 0)  ) {
       HasFileNameError();
       DisplayHelp(&uarg[0]);
       return(SC_SEVERE);
      }
     dummy_filename = 1;
     strcat(MVSFilename,dummy_name);
     strcat(MVSFilename,",start(");
     strcat(MVSFilename,argv[3]);
     strcat(MVSFilename,")");
}                            /* End of START parameter        */
else {                       /* None of the above             */
     /*******************************************************/
     /* Free-form command otherwise                         */
     /*******************************************************/
     /*******************************************************/
     /* Check for help request.                             */
     /*******************************************************/
     if (strncmp(uarg, "?",1) == 0) {
       DisplayHelp(&uarg[0]);
       return(SC_WARNING);
      }

     minparms = MINPFF;
     maxparms = MAXPFF;
     if (intrc = CheckRange(minparms,maxparms,argc, uarg) > 0)
       return(intrc);
     /*******************************************************/
     /* Set special processing flags                        */
     /*******************************************************/
     for (i=minparms; i < argc; i++) {
           if (intrc = SpecialOptions(i, argc, argv) > 0)
         return(intrc);
     }
```

Figure 35. DFM DataAgent Sample (Part 10 of 25)

```
          if (intrc = strupper(MVSFilename,argv[1], PATHLEN))
            return(intrc);
          fnlen = strlen(MVSFilename);  /* Save true filename len */
          if (fnlen < 3) {
            NoFileNameError();
            DisplayHelp(&uarg[0]);
            return(SC_SEVERE);
           }

     }                           /* End of free form parameters   */
}

/*******************************************************************/
/* Begin processing the MVS file                                   */
/*******************************************************************/

if (debug >= 1)  printf
  ("DFMACALL: Processing filename and filename suffix of %s.\n",
             MVSFilename);

if (display_filename) {
  /*****************************************************************/
  /* Perform unload of records to satisfy display request         */
  /*****************************************************************/

  /*****************************************************************/
  /* Allocate a record buffer                                      */
  /*****************************************************************/
  RecALSize = 64000;        /* Try to use a 64K Buffer for records*/
  if ((pRecord = (PDDMRECORD)malloc(RecALSize)) == NULL) {
     /* Not enough storage--make one last try for 32K             */
     RecALSize = RecALSize / 2;               /* Try 32K Buffer  */
     if ((pRecord = (PDDMRECORD)malloc(RecALSize)) == NULL) {
       printf("DFMACALL: Out of memory\n");
       return(SC_SEVERE);
     }
  }
  /*****************************************************************/
  /* Unload the first batch of records in rcd number order        */
  /*****************************************************************/
  SevCode = 1;
  while (SevCode) {
    SevCode = DDMUnLoadFileFirst
           (MVSFilename,              /* FileName                */
            &UnLoad,                  /* UnLoadHandle            */
            0UL,                      /* AccessFlags             */
            &DDMMoreDataFlag,         /* Flags                   */
            pRecord,                  /* RecordBuf               */
            RecALSize,                /* RecordBufLen            */
            (CODEPOINT) RECSEQ,       /* UnloadOrder=rcd number */
            &RecCount                 /* RecCount                */
           );
```

Figure 35. DFM DataAgent Sample (Part 11 of 25)

```
        if (SevCode > SC_WARNING) {
           printf("DFMACALL: Error on DDMUnLoadFileFirst for %s.\n",
                    MVSFilename);
           printf("Severity code = %u\n",SevCode);
           LCodePoint = ReplyMsg();
           /* Retry TSO unload output file if not tried already.      */
           if (strcmp(MVSFilename,RootName) != 0  &
               LCodePoint == VALNSPRM ) {
             TSO_retry = 1;
             strcpy(MVSFilename,RootName);
           }                                /* End, TSO retry         */
           else {                           /* Permanent error        */
             free(pRecord);
             return(SevCode);
           }                                /* End, permanent error   */
        }     /* End of error from unload file first.                 */
      }       /* End of while no error.                               */

      if (SevCode > SC_WARNING) {
        free(pRecord);
        return(SevCode);
      }                                      /* End, unload first err */

      if (debug >= 2 )
         printf ("DDMUnLoadFileFirst: %d records in buffer .\n",
                  RecCount);
      if (TSO_retry > 0 ) {
         printf ("\n** The TSO Output File associated with the error is\
 as follows: **\n");
         printf ("**    (Note that its contents may be from a previou\
s run.)      **\n");
      }

      DisplayBuffer(RecCount,pRecord);

      /***************************************************************/
      /* Unload remaining records in record number order.            */
      /* When DDMMoreDataFlag is 0x00UL then the file handle is      */
      /* invalid and the file will be closed.                        */
      /***************************************************************/
      while (DDMMoreDataFlag == 0x01UL)
        {
           SevCode = DDMUnLoadFileNext
              (UnLoad,                         /* UnLoadHandle       */
               0x0000UL,                       /* Flags              */
               &DDMMoreDataFlag,               /* UnloadFlags        */
               pRecord,                        /* RecordBuf          */
               RecALSize,                      /* RecordBufLen       */
               &RecCount                       /* RecCount           */
              );
```

Figure 35. DFM DataAgent Sample (Part 12 of 25)

```
        if (SevCode > SC_WARNING) {
           printf("DFMACALL: Error on DDMUnLoadFileNext for %s.\n",
                    MVSFilename);
           printf("Severity code = %u\n",SevCode);
           ReplyMsg();
           free(pRecord);
           return(SevCode);
        }
        if (debug >= 2)
          printf ("DDMUnLoadFileNext: %d records in buffer.\n",
            RecCount);
        DisplayBuffer(RecCount,pRecord);

    }                                       /* End of WHILE loop  */
 free(pRecord);

  } else {
    /***************************************************************/
    /* No display--just open to trigger DataAgent and then close    */
    /***************************************************************/

    /***************************************************************/
    /* Open the file                                               */
    /***************************************************************/
    SevCode = DDMOpen
            (MVSFilename,                   /* FileName        */
             &FileHandle,                   /* FileHandle      */
             RELRNBAM,                      /* AccessMethod    */
             DDM_GETAI,                     /* AccIntList      */
             DDM_UPDATERS,                  /* FileShare       */
             NULL,                          /* EABuf           */
             NULL                           /* reserved        */
            );
    if (SevCode != SC_NO_ERROR)
    {
       if (dummy_filename == 0)
       {
       printf("Error opening file %s\n",MVSFilename);
       printf("Severity code = %u\n",SevCode);
       }
       ReplyMsg();
       return(SevCode);

    }
```

Figure 35. DFM DataAgent Sample (Part 13 of 25)

```
      /****************************************************************/
      /* Close the file                                               */
      /****************************************************************/
      SevCode = DDMClose
              (FileHandle             /* FileHandle      */
              );

      if (SevCode != SC_NO_ERROR)
      {
         if (dummy_filename == 0)
         {
         printf("Error closing file %s\n",MVSFilename);
         printf("Severity code = %u\n",SevCode);
         }
         ReplyMsg();
         return(SevCode);
      }
    }

    return(SC_NO_ERROR);

} /* End--sample main */

/***************************************************************************
****************************  ReplyMsg  ***********************************
***************************************************************************
*   Process the reply message if there is a Severity Code other than
*   SC_NO_ERROR;
*
***************************************************************************/
CODEPOINT   ReplyMsg(VOID)
{
    static BYTE pRpyMsgBuf[RPYMSBFLN];

    APIRET     rc;
    CODEPOINT  CodePoint;
    PDDMOBJECT pReplyObject;
    USHORT     index;
```

Figure 35. DFM DataAgent Sample (Part 14 of 25)

```
/*--------------------------------------------------------------------
--  The following table contains the count for the number of parameters
--  expected for each reply message (1st column), and it also contains
--  the expanded error messages
--
-- The first message in the table, KEYUDIRM, has the lowest
-- code point value.  It is also the first message in a block of
-- message code points that ends with RECNAVRM.
--
-- The next block of message code points (in ascending code point order)
-- begins with OS2ERRRM and ends with FILERRRM.
-- The low-order byte is used as the index into this block.
---------------------------------------------------------------------*/
static struct
{  USHORT   Count;
   BYTE     msg[52];
} ErrorMsgBuffer[] =
  { 6, "Key Update Not Allowed by Different Index         \0",
    0, "                                                  \0",
    0, "                                                  \0",
    0, "Default Record Error                              \0",
    5, "Cursor Not Selecting a Record Position            \0",
    7, "Invalid Data Record                               \0",
    3, "Duplicate File Name                               \0",
    8, "Duplicate Key Different Index                     \0",
    7, "Duplicate Key Same Index                          \0",
    7, "Duplicate Record Number                           \0",
    3, "End of File                                       \0",
    7, "File is Full                                      \0",
    4, "File in Use                                       \0",
    3, "File Not Found                                    \0",
    6, "File Space Not Available                          \0",
    0, "                                                  \0",
    0, "                                                  \0",
    3, "Invalid File Name                                 \0",
    0, "                                                  \0",
    0, "                                                  \0",
    7, "Record Length Mismatch                            \0",
    0, "                                                  \0",
    0, "                                                  \0",
    0, "                                                  \0",
    0, "                                                  \0",
    0, "                                                  \0",
    0, "                                                  \0",
    2, "Not Authorized to Function                        \0",
    0, "                                                  \0",
    4, "File Temporarily Not Available                    \0",
    0, "                                                  \0",
    0, "                                                  \0",
    0, "                                                  \0",
    0, "                                                  \0",
    0, "                                                  \0",
```

Figure 35. DFM DataAgent Sample (Part 15 of 25)

```
        7, "Record Number Out of Bounds                       \0",
        5, "Record Not Found                                  \0",
        0, "                                                  \0",
        0, "                                                  \0",
        0, "                                                  \0",
        0, "                                                  \0",
        0, "                                                  \0",
        0, "                                                  \0",
        0, "                                                  \0",
        3, "Invalid Key Length                                \0",
        0, "                                                  \0",
        0, "                                                  \0",
        3, "Not Authorized to Access Method                   \0",
        0, "Invalid Access Method                             \0",
        3, "Permanent Agent Error                             \0",
        6, "Resource Limits Reached on Target System          \0",
        3, "Invalid Base File Name                            \0",
        0, "                                                  \0",
        0, "                                                  \0",
        2, "Not Authorized to Directory                       \0",
        0, "Management Class Conflict                         \0",
        0, "Storage Class Conflict                            \0",
        3, "Existing Condition                                \0",
        4, "Not Authorized to File                            \0",
        6, "Invalid Request                                   \0",
        4, "Invalid Key Definition                            \0",
        0, "                                                  \0",
        5, "Key Update Not Allowed by Same Index              \0",
        8, "Invalid Key Value                                 \0",
        0, "                                                  \0",
        0, "                                                  \0",
        3, "Open Exclusive by Same User                       \0",
        4, "Concurrent Open Exceeds Maximum                   \0",
        4, "Conversational Protocol Error                     \0",
        0, "                                                  \0",
        0, "                                                  \0",
        0, "                                                  \0",
        7, "Record Damaged                                    \0",
        7, "Record in Use                                     \0",
        0, "                                                  \0",
        5, "Data Stream Syntax Error                          \0",
        7, "Update Cursor Error                               \0",
        5, "No Update Intent on Record                        \0",
        3, "Invalid New File Name                             \0",
        3, "Function Not Supported                            \0",
        3, "Parameter Not Supported                           \0",
        4, "Parameter Value Not Supported                     \0",
        4, "Object Not Supported                              \0",
        5, "Command Check                                     \0",
        0, "                                                  \0",
```

Figure 35. DFM DataAgent Sample (Part 16 of 25)

```
   0, "                                                   \0",
   2, "File Handle Not Found                              \0",
   3, "Directory Full                                     \0",
   3, "Record Inactive                                    \0",
   7, "File Damaged                                       \0",
   4, "Load Records Count Mismatch                        \0",
   3, "Not Authorized to Open Intent for Named File      \0",
   0, "                                                   \0",
   3, "File Closed with Damage                            \0",
   2, "Target Not Supported                               \0",
   5, "Key Value Modified after Cursor was Last Set      \0",
   0, "                                                   \0",
   0, "                                                   \0",
   0, "                                                   \0",
   0, "                                                   \0",
   0, "                                                   \0",
   0, "Access Intent List Error                           \0",
   0, "                                                   \0",
   0, "                                                   \0",
   0, "                                                   \0",
   0, "                                                   \0",
   0, "                                                   \0",
   0, "                                                   \0",
   0, "                                                   \0",
   0, "                                                   \0",
   5, "Record Not Available                               \0",

/************ START OF SECOND CODE POINT RANGE *************/
   0, "OS/2 Error                                         \0",
   0, "Data Description File Not Found                    \0",
   0, "Conversion Table Not Found                         \0",
   2, "Translation Error                                  \0",
   0, "                                                   \0",
   2, "Invalid Flag                                       \0",
   0, "                                                   \0",
   2, "Communications Error                               \0",
   0, "                                                   \0",
   0, "                                                   \0",
   0, "                                                   \0",
   0, "                                                   \0",
   0, "                                                   \0",
   0, "                                                   \0",
   0, "                                                   \0",
   0, "                                                   \0",
   2, "Resource Limit Reached in OS/2 V2.0 Source System \0",
   2, "Field Length Error                                 \0",
   2, "Address Error                                      \0",
   0, "                                                   \0",
   2, "Function Continuation Error                        \0",
   0, "                                                   \0",
   2, "File Error                                         \0"
 };
```

Figure 35. DFM DataAgent Sample (Part 17 of 25)

```
/*------------------------------------------------------------------
-- For each reply message available, retrieve and display it.
------------------------------------------------------------------*/
do
{  /*---------------------------------------------------------------
   -- Get the reply message
   ---------------------------------------------------------------*/
   rc = DDMGetReplyMessage(pRpyMsgBuf, (ULONG)RPYMSBFLN, (ULONG)1);

   switch (rc)
   {  case SC_NO_ERROR:      /* All reply messages have been received */
      case SC_WARNING: /* There are more reply messages to be received*/
         break;
      case SC_ERROR:
         printf("   ReplyMsg: reply message buffer is too small -\n");
         printf("             enlarge and recompile ...\n");
         return(rc);
         break;
      case SC_SEVERE:
         printf("   ReplyMsg: Warning: A reply message was requested,\n");
         printf("             but there are none available ...\n");
         return(rc);
         break;
      case SC_ACCESSDAMAGE:
         printf("   ReplyMsg: Error: An invalid reply message buffer\n");
         printf("             address was specified ...\n");
         return(rc);
         break;
      case SC_PERMDAMAGE:
         printf("   ReplyMsg: Severe Error: An unarchitected reply message\n");
         printf("             object was encountered ...\n");
         return(rc);
         break;
      default:
         printf("   ReplyMsg: Unknown return code from DDMGetReplyMessage\n");
         return(rc);
         break;
   } /* endswitch */
```

Figure 35. DFM DataAgent Sample (Part 18 of 25)

```
      /*-----------------------------------------------------------------
      -- Get the reply message
      -----------------------------------------------------------------*/
      pReplyObject = (PDDMOBJECT)pRpyMsgBuf;

      CodePoint = pReplyObject->cpObject;              /* get code point */

      /* reset pointer to first parm base */
      pReplyObject = (PDDMOBJECT)((PBYTE)pReplyObject
                                  + (sizeof(CODEPOINT)
                                     + sizeof(OBJLENGTH))
                                 );

      /*-----------------------------------------------------------------
      -- Calculate the index into the parameter/msg table based on
      -- the codepoint.
      -----------------------------------------------------------------*/
      if (CodePoint <= RECNAVRM)         /* if code point in first block */
         index = (USHORT)(CodePoint - KEYUDIRM);
      else                               /* code point in second block */
         index = (USHORT)
                 ((RECNAVRM - KEYUDIRM + 1) /* number of entries in
                                               first block */
                  + (CodePoint % 0x0100UL) /* index into second block */
                 );

      /* If the index indicates "file not found" and a dummy filename    */
      /* is being used, ignore the error.                                */
      if (index == 13 & dummy_filename == 1 )
        return(SC_NO_ERROR);

      /*-----------------------------------------------------------------
      -- Begin dissecting the reply message buffer
      -----------------------------------------------------------------*/
      if (ErrorMsgBuffer[index].Count > 0)
      {  printf("RPYMSG: %s\n",ErrorMsgBuffer[index].msg);
         DumpBuffer(pReplyObject, ErrorMsgBuffer[index].Count);
         printf("\n");
      }

   } while (rc == SC_WARNING); /* enddo */
  return(CodePoint);

} /* ReplyMsg */
```

Figure 35. DFM DataAgent Sample (Part 19 of 25)

```
/***************************************************************************
*************************  DumpBuffer  **********************************
***************************************************************************
*
*   For each object in the reply message buffer, print out its contents.
*
***************************************************************************/
VOID DumpBuffer(PDDMOBJECT  pAttribute,
                USHORT      Count)
{
    int i;                               /* Local loop counter  */
    do
    {  if (pAttribute->cbObject == (sizeof(CODEPOINT) + sizeof(OBJLENGTH)))
       {  printf("Null object returned = %x\n",pAttribute->cbObject);
          pAttribute->cpObject = 0;
       }
       else
       {  switch(pAttribute->cpObject)
          {  case ACCMTHCL:                 /* Access Method Class */
                printf("ACCMTHCL = 0x%X\n", *(PCODEPOINT)(pAttribute-
>pData));
                break;
             case BASFILNM:                 /* Base File Name */
               printf("BASFILNM = %s\n", pAttribute->pData);
                break;
             case CODPNT:                   /* Code Point */
                printf("CODPNT = 0x%X\n", *(PCODEPOINT)(pAttribute-
>pData));
                break;
             case CSRPOSST:                 /* Cursor Position Status */
                printf("CSRPOSST = 0x%hX\n", *(PBYTE)(pAttribute-
>pData));
                break;
             case DTALCKST:                 /* Data Lock Status */
                printf("DTALCKST = 0x%hX\n", *(PBYTE)(pAttribute-
>pData));
                break;
             case ERRFILNM:                 /* Error File Name */
                printf("ERRFILNM = %s\n", pAttribute->pData);
                break;
             case FILNAM:                   /* File Name */
                printf("FILNAM = %s\n", pAttribute->pData);
                break;
             case KEYDEFCD:                 /* Key Definition Error Code */
                printf("KEYDEFCD = 0x%hX\n", *(PBYTE)(pAttribute-
>pData));
                break;
             case MAXOPN:                   /* Maximum Number of File Extents
                                               Concurrent Opens Allowed */
                printf("MAXOPN = %d\n", *(PUSHORT)(pAttribute->pData));
                break;
```

Figure 35. DFM DataAgent Sample (Part 20 of 25)

```
            case NEWFILNM:                 /* New File Name */
               printf("NEWFILNM = %s\n", pAttribute->pData);
               break;
            case PRCCNVCD:                 /* Conversational Protocol Error Code */
               printf("PRCCNVCD = 0x%hX\n", *(PBYTE)(pAttribute-
>pData));
               break;
            case RECCNT:                   /* Record Count */
               printf("RECCNT = %ld\n", *(PULONG)(pAttribute->pData));
               break;
            case RECNBR:                   /* Record Number */
               printf("RECNBR = %ld\n", *(PRECNUM)(pAttribute->pData));
               break;
            case SRVDGN:      {            /* Server Diagnostic Information */
               printf("SRVDGN = 0x\n");
               for (i=1; i < (pAttribute->cbObject-5); i++) /* 2 byte len, 2 byte codept*/
                   { if (i % 16 ==0)
                                printf("%02X\n", *(PBYTE)(pAttribute->pData+i-
1));
                      else
                      if (i % 4 ==0)
                                printf("%02X ", *(PBYTE)(pAttribute->pData+i-
1));
                      else
                          printf("%02X", *(PBYTE)(pAttribute->pData+i-1));
                     }
              }
               break;
            case SVRCOD:                   /* Severity Code */
               printf("SVRCOD = 0x%X\n", *(PCODEPOINT)(pAttribute-
>pData));
               break;
            case SYNERRCD:                 /* Syntax Error Code */
               printf("SYNERRCD = 0x%hX\n", *(PBYTE)(pAttribute-
>pData));
               break;
            default:
               printf("Unknown code point - 0x%X\n",
                      *(PCODEPOINT)(pAttribute->pData));
               break;
         } /* endswitch */
      } /* endif */

      /* go to next object */
      pAttribute = (PDDMOBJECT)((PBYTE)pAttribute + pAttribute->cbObject);

   } while(--Count > 0);

} /* DumpBuffer */
```

Figure 35. DFM DataAgent Sample (Part 21 of 25)

```
/*****************************************************************************
****************************  Error Routines ******************************
*****************************************************************************/
VOID GeneralError()
{
  printf("DFMACALL: Incorrect command line syntax.\n");
} /* GeneralError */

VOID OmitError()
{
  printf("DFMACALL: A required parameter was omitted.\n");
}

VOID TooManyError()
{
  printf("DFMACALL: Too many parameters were on the command line.\n");
}

VOID NotEnoughError()
{
 printf("DFMACALL: Not enough parameters were on the command line.\n");
}

VOID HasFileNameError()
{
 printf("DFMACALL: Filename is not allowed for QTSO, TSO, or START.\n");
}

VOID NoFileNameError()
{
 printf("DFMACALL: A filename must be specified.\n");
}

VOID ValueError(char *value)
{
 printf("DFMACALL: Incorrect parameter value %s.\n",value);
}

VOID ParmLenError(char *value)
{
 printf("DFMACALL: Parameter %s is too long.\n",value);
}

int strupper(char *oarg, char *iarg, int bufflen)
{
 /* Convert string to upper case.                                     */
 int i;
 if (strlen(iarg) > bufflen) {
   ParmLenError(iarg);
   return(SC_SEVERE);
 }
```

Figure 35. DFM DataAgent Sample (Part 22 of 25)

```
 for (i=0; i < strlen(iarg); i++) {
    oarg[i] = toupper(iarg[i]);
 }
 oarg[i] = 0;
 return (0);

}

VOID DuplicateError()
{
 printf("DFMACALL: One or more parameters were duplicated.\n");
}

VOID  DisplayBuffer(ULONG count, PDDMRECORD pCurrentRecord)
{
    /* Display a buffer full of records                                      */
    ULONG i;                         /* record counter                       */
    int j;                           /* index to character in record         */
    int cRecLen;                     /* current record length                */
    UCHAR c;                         /* current converted character          */
    UCHAR savechar;                  /* savearea for trailing character      */

         for (i=1; i <= count; i++) {
           cRecLen = pCurrentRecord->cbRecord - sizeof(pCurrentRecord->cbRecord)
                          - sizeof(pCurrentRecord->cpRecord);
           /*****************************************************************/
           /* Replace all instances of non-printable characters,            */
           /*****************************************************************/
           /* Make sure the string is printable and                         */
           /* make sure that there is no 0 in the middle of string.         */
           for (j=0; j < cRecLen; j++) {
             if (!(c = pCurrentRecord->pRecord[j]))
               pCurrentRecord->pRecord[j] = ' ';  /* Replace x00 with blank   */
             else if (!isprint(c))
               pCurrentRecord->pRecord[j] = '.';  /* Make nonprintable a "."  */
           }                                       /* End of for j= loop       */
           savechar = pCurrentRecord->pRecord[cRecLen]; /* save trailing char */
           pCurrentRecord->pRecord[cRecLen] = '\0';

           if (debug >= 5) {
             display_counter++;
             printf ("Displaying record %d with length %d:\n",
                      display_counter,cRecLen);
           }
           printf ("%s\n",pCurrentRecord->pRecord);
           pCurrentRecord->pRecord[cRecLen] = savechar;  /* restore trailing   */
           pCurrentRecord = (PDDMRECORD)  (pCurrentRecord->pRecord + cRecLen);
         }

}
```

Figure 35. DFM DataAgent Sample (Part 23 of 25)

```
     SpecialOptions(int index, int argc, char  *argv[])
{
 CHAR uarg[PARMLEN];            /* Upper case argument                */
  /* Check for special processing options.                            */
  if (intrc = strupper(uarg, argv[index], PARMLEN))
    return(intrc);
  if (strcmp(uarg,"DISPLAY") == 0)
    display_filename = 1;

 return(0);
}         /* End of SpecialOptions  */

     CheckRange(int minparms, int maxparms, int argc, char uarg[PARMLEN])
{
 /* Ensure number of parameters is reasonable for the command   */

    /*******************************************************/
    /* Ensure enough parameters                            */
    /*******************************************************/
    if (argc < minparms) {
      NotEnoughError();
      DisplayHelp(&uarg[0]);
      return(SC_SEVERE);
    }

    /*******************************************************/
    /* Ensure no leftover parameters                       */
    /*******************************************************/
    if (argc > maxparms) {
      TooManyError();
      DisplayHelp(&uarg[0]);
      return(SC_SEVERE);
    }

return(0);

}
```

Figure 35. DFM DataAgent Sample (Part 24 of 25)

```
/**********************************************************************
****************************  DisplayHelp *****************************
**********************************************************************
*   Display the correct syntax for invoking this function.
*
*********************************************************************/
VOID DisplayHelp(char *fullhelp)
{
if (strncmp(fullhelp,"?",1) == 0) {
  /* Print full help text.                                          */
  printf("Correct syntax:                                             \n\n");
  printf("  DFMACALL QTSO  driveletter:  TSOcommandline  [DISPLAY]      \n");
  printf("  DFMACALL TSO   driveletter:  [TSOcommandline] [DISPLAY]
\n");
  printf("  DFMACALL AGENT driveletter:[filename]
MVSproc[,proc_parms]  \n");
  printf("     [PGM prog_name] [PARM prog_parms]
[DISPLAY]              \n");
  printf("  DFMACALL START driveletter:    MVSproc[,proc_parms]         \n");
  printf("  DFMACALL driveletter:filename[,filename_suffix]
[DISPLAY] \n\n");
  printf(" Examples:                                             \n\n");
  printf("   dfmacall qtso  r: listc  display                      \n");
  printf("   dfmacall tso   r:        display                      \n");
  printf("   dfmacall agent r:ibmuser.a.b  dfmxagnt                \n");
  printf("   dfmacall agent r:ibmuser.a.b  dfmxtso                 \n");
  printf("           pgm ikjeft01 parm listc                       \n");
  printf("   dfmacall start r:          dfmx0001,dfminit=iefbr14   \n");
  printf("   dfmacall r:ibmuser.a.b,agent(dfmxagnt),parm(hello)    \n");
  printf("   dfmacall r:ibmuser.a.b,agent(dfmx0001),pgm(dfmxagnt)\n\n");
}
else {
  /* Print clue for getting correct help text.                          */
  printf("DFMACALL: Enter DFMACALL ? to get the correct command syntax. \n\n");
}
}
```

Figure 35. DFM DataAgent Sample (Part 25 of 25)

Appendix K. DDM File Attributes

Table 2 summarizes the DDM file attributes by DDM file class.

Table 2. DDM File Attributes

Attribute	Description	Chg	PDSE Member (SEQFIL)	PDSE Member (STRFIL)	SAM	ESDS	KSDS/ VRRDS	RRDS
ACCMTHLS	Access method list	NO	YES	YES	YES	YES	YES	YES
DELCP	Delete capable	NO	YES	...	YES	YES	YES	YES
DFTREC	Default record	NO	YES	...	NO	NO	NO	NO
DTACLSNM	Data class name	NO	YES	YES	YES	YES	YES	YES
DTAFMT	Data format	YES	...	YES	...	...	...	...
EOFNBR	End of file number	NO	YES	YES	NO	YES	YES	YES
EOFOFF	End of file offset	NO	...	YES	...	...	...	...
FILEBYTCN	File byte count	NO	YES	YES	NO	NO	NO	NO
FILCHGDT	File change date	YES	YES	YES	YES	YES	YES	YES
FILCLS	File class	NO	YES	YES	YES	YES	YES	YES
FILCRTDT	File creation date	NO	YES	YES	YES	YES	YES	YES
FILEXNCN	File extent count	NO	NO	NO	NO	NO	NO	NO
FILEXNSZ	File extent size	NO	YES	YES	NO	NO	NO	YES
FILEXPDT	File expiration date	YES	YES	YES	YES	YES	YES	YES
FILHDD	File hidden	MEM	YES	YES	YES	YES	YES	YES
FILINISZ	Initial file size	NO	YES	YES	NO	NO	NO	YES
FILMAXEX	Maximum number extents	NO	YES	YES	YES	YES	YES	YES
FILNAM	File name	NO	YES	YES	YES	YES	YES	YES
FILOPNLO	Open lock options	NO	YES	NO	NO	NO	NO	...
FILPRT	File protected	YES	YES	YES	YES	YES	YES	YES
FILSIZ	File size	NO	YES	...	NO	...	...	YES
FILSYS	System file	MEM	YES	YES	YES	YES	YES	YES
GETCP	Get capable	MEM	YES	YES	YES	YES	YES	YES
INSCP	Insert capable	MEM	YES	...	YES	YES	YES	YES
KEYDEF	Key definition	NO	...	...	...	...	YES/ ...	...
KEYDUPCP	Duplicate key capable	NO	...	...	...	...	YES/ ...	...
LSTACCDT	Last access date	NO	NO	NO	YES	YES	YES	YES
LSTARCDT	Last archive date	NO	YES	YES	NO	NO	NO	NO
MGMCLSNM	Management class name	DS	YES	YES	YES	YES	YES	YES
MODCP	Modify capable	MEM	YES	YES	YES	YES	YES	YES
RECLEN	Record length	NO	YES	...	YES	YES	YES	YES
RECLENCL	Record length class	NO	YES	...	YES	YES	YES	YES
RTNCLS	Retention class	NO	YES	YES	YES	YES	YES	YES

Table 2. DDM File Attributes (continued)

Attribute	**Description**	**Chg**	**PDSE Member (SEQFIL)**	**PDSE Member (STRFIL)**	**SAM**	**ESDS**	**KSDS/ VRRDS**	**RRDS**
SHDEXS	Shadow exists	NO	YES	YES	YES	YES	YES	YES
STGCLSNM	Storage class name	DS	YES	YES	YES	YES	YES	YES
STRSIZ	Stream size	YES	...	NO	...	...	...	...
TITLE	Title	MEM	YES	YES	NO	NO	NO	NO

Legend for Chg column:

YES	Attribute value can be changed with the CHGFAT command for a full access data set.
NO	Attribute value cannot be changed with the CHGFAT command.
DS	Attribute value can be changed for data sets only with the CHGFAT command.
MEM	Attribute value can be changed for PDSE members only with the CHGFAT command.

Legend for Data Set/Member columns:

YES	Supported; that is, an attribute value is returned when requested on a LSTFAT command.
NO	Not supported; that is, no attribute value is returned when requested on a LSTFAT command.
...	Does not apply to any of the possible file classes for this data set member.

Appendix L. Application Programming Considerations

This appendix contains programming considerations relevant to DDM client record and stream file access. When programming with these application programming interfaces (APIs), you need to consider Distributed FileManager/MVS file creation support, access command support, access restrictions, and logon mode requirements.

Distributed FileManager/MVS Implementation

Distributed FileManager/MVS provides a subset of DDM access methods, file types, and commands. In some cases, Distributed FileManager/MVS does not support certain record access and stream file API commands or command parameters.

DDM Record Access File Creation

Distributed FileManager/MVS creates record-oriented files based on DDMCreateRecFile command parameter settings. The following are optional parameter settings which govern data set creation within the indicated FileClass. Also provided are some mandatory flag and parameter settings for supported Distributed FileManager/MVS functions.

FileClass SEQFIL

Results in the creation of a SAM data set, a PDSE member, a PDS member, or a VSAM RRDS or VRRDS.

- The type of data set created depends on these parameters:
  - If the FileName parameter includes a *member name*, and the Delete Capability parameter is *off*, a PDSE member or PDS member is created. (If the data set does not exist, a PDSE is created first, and then the member is created.)
  - If the FileName does *not* include a member name, and Delete Capability is *on*, a VSAM RRDS or VRRDS is created.
  - If the FileName does *not* include a member name, and Delete Capability is *off*, a SAM data set is created.
- FileClass SEQFIL CreateFlags mandatory settings are:
  Set off the following bit flag: DDM_TMPFIL

FileClass KEYFIL

Results in the creation of a VSAM KSDS data set.

- FileClass KEYFIL CreateFlags mandatory settings are:
  Set off the following bit flags: DDM_TMPFIL, DDM_ALDUPKEY
- In FileClass KEYFIL, for the parameters DftRec and DftRecOp, the only valid value is NIL.

FileClass DIRFIL

Results in the creation of a VSAM RRDS or VRRDS data set.

- The type of data set created is determined by the following:
  - A RecLenCls value of RECFIX results in a RRDS data set
  - A RecLenCls value of RECIVL or RECVAR results in a VRRDS data set
- FileClass DIRFIL CreateFlags mandatory settings are:
  Set off the following bit flag: DDM_TMPFIL

Additional Considerations

You should also be aware that:

- If you do not specify InitFileSiz for DDMCreateRecFile, the file size is determined by your ACS routines.

Stream File Creation

You can use the DDM Stream access method to create stream files in SAM data sets or PDSE members. The type of data set created depends on:

- If the FileName parameter does not include a member name, a SAM data set is created.
- If the FileName includes a member name, a PDSE member is created. (If the data set does not exist, a PDSE is created first, and then the member is created.)

You should also be aware that:

- Distributed FileManager/MVS uses a tunable parameter in DFM00 called STREAM_LRECL for record length and RECFM = V to create new SAM data sets or PDSE data sets. These attributes override LRECL and RECFM of the SMS data class defaulted by the ACS routine.

File Access Commands Supported by Distributed FileManager/MVS

Only commands supported by the following DDM access methods can be issued using Distributed FileManager/MVS:

Access Method	**Description**
RELRNBAM	Relative by record number access method
RNDRNBAM	Random by record number access method
CMBRNBAM	Combined record number access method
RELKEYAM	Relative by key access method
RNDKEYAM	Random by key access method
CMBKEYAM	Combined key access method
STRAM	Stream access method

Sequential Files

Table 3 lists the DDM access method commands supported for Distributed FileManager/MVS sequential files.

Table 3. DDM Access Method Commands Supported for Distributed FileManager/MVS Sequential Files

Access Commands	**RELRNBAM**	**RNDRNBAM**	**CMBRNBAM**	**STRAM**
CHGEOF	...	...	...	YES*
CLOSE	YES	YES	YES	YES
DELREC	YES#	YES#	YES#	...
FRCBFF	YES	YES	YES	YES
GETREC	YES	YES	YES	...
GETSTR	...	...	...	YES

Table 3. DDM Access Method Commands Supported for Distributed FileManager/MVS Sequential Files (continued)

Access Commands	**RELRNBAM**	**RNDRNBAM**	**CMBRNBAM**	**STRAM**
INSRECEF	YES	YES	YES	...
INSRECNB	...	YES+	YES+	...
LCKSTR	...	...	...	YES
MODREC	YES	YES	YES	...
OPEN	YES	YES	YES	YES
PUTSTR	...	...	...	YES*
SETBOF	YES	YES	YES	...
SETEOF	YES@	YES@	YES@	...
SETFRS	YES	YES	YES	...
SETLST	YES@	YES@	YES@	...
YES	...	YES	...	
...	YES	YES	...	
SETNXT	YES	...	YES	...
YES	...	YES	...	
YES	...	YES	...	
NO	YES	YES	...	
UNLIMPLK	YES	YES	YES	...
UNLSTR	...	...	...	Yes

Legend:

YES	The command is supported.
NO	The command is not supported.
...	The command does not apply to the access method.
#	DELREC is only supported for RRDSs and VRRDSs.
@	SETEOF and SETLST are not supported for PDS members.
+	INSRECNB returns “duplicate record number” for PDSE members and PDS members.
*****	Distributed FileManager/MVS limits stream access to read-only support for VRRDS and RRDS data sets. All stream access to non-reusable VSAM data sets is read-only.

Direct Files

Table 4 lists the DDM access method commands supported for Distributed FileManager/MVS direct files.

Table 4. DDM Access Method Commands Supported for Distributed FileManager/MVS Direct Files

Access Commands	**RELRNBAM**	**RNDRNBAM**	**CMBRNBAM**	**STRAM**
CHGEOF	...	...	...	YES*
CLOSE	YES	YES	YES	YES
DELREC	YES	YES	YES	...
FRCBFF	YES	YES	YES	YES

Table 4. DDM Access Method Commands Supported for Distributed FileManager/MVS Direct Files (continued)

Access Commands	RELRNBAM	RNDRNBAM	CMBRNBAM	STRAM
GETREC	YES	YES	YES	...
GETSTR	...	...	...	YES
INSRECEF	YES	YES	YES	...
INSRECNB	...	YES	YES	...
LCKSTR	...	...	...	YES
MODREC	YES	YES	YES	...
OPEN	YES	YES	YES	YES
PUTSTR	...	...	...	YES*
SETBOF	YES	YES	YES	...
SETEOF	YES	YES	YES	...
SETFRS	YES	YES	YES	...
SETLST	YES	YES	YES	...
YES	...	YES	...	
...	YES	YES	...	
SETNXT	YES	...	YES	...
YES	...	YES	...	
YES	...	YES	...	
NO	YES	YES	...	
YES	YES	YES	...	
UNLSTR	...	...	...	YES

Legend:

YES	The command is supported.
NO	The command is not supported.
...	The command does not apply to the access method.
*****	Distributed FileManager/MVS limits stream access to read-only support for RRDSs and VRRDSs. All stream access to non-reusable VSAM data sets is read-only.

Keyed Files

Table 5 lists the DDM access method commands supported for Distributed FileManager/MVS keyed files.

Table 5. DDM Access Method Commands Supported for Distributed FileManager/MVS Keyed Files

Access Commands	RELKEYAM	RNDKEYAM	CMBKEYAM	STRAM
CHGEOF	...	...	...	YES*
CLOSE	YES	YES	YES	YES
DELREC	YES#	YES#	YES#	...
FRCBFF	YES	YES	YES	YES
GETREC	YES	YES	YES	...
GETSTR	...	...	...	YES

Table 5. DDM Access Method Commands Supported for Distributed FileManager/MVS Keyed Files (continued)

Access Commands	**RELKEYAM**	**RNDKEYAM**	**CMBKEYAM**	**STRAM**
INSRECKY	YES	YES	YES	...
INSRECNB	NO	NO	NO	...
LCKSTR	...	...	...	YES
MODREC	YES	YES	YES	...
OPEN	YES	YES	YES	YES
PUTSTR	...	...	...	YES*
SETBOF	YES	YES	YES	...
SETEOF	YES	YES	YES	...
SETKEY	YES@	YES	YES	...
SETKEYFR	YES	YES	YES	...
SETKEYLM	YES	...	YES	...
SETKEYLS	YES	YES	YES	...
SETKEYNX	YES	...	YES	...
SETKEYPR	YES	...	YES	...
SETNXTKE	YES	...	YES	...
YES	YES	YES	...	
YES	YES	YES	...	
UNLSTR	...	...	...	YES

Legend:

Yes	The command is supported.
No	The command is not supported.
...	The command does not apply to the access method.
*****	Distributed FileManager/MVS limits stream access to read-only support for keyed files.
#	DELREC is only supported for KSDSs.
@	RELKEYAM is promoted to CMBKEYAM.

Stream Files

Table 6 lists the DDM access method commands supported for Distributed FileManager/MVS stream files.

Table 6. DDM Access Method Commands Supported for Distributed FileManager/MVS Stream Files

Access Commands	**STRAM**
CHGEOF	YES
CLOSE	YES
FRCBFF	YES
GETSTR	YES
LCKSTR	YES
OPEN	YES
PUTSTR	YES

Table 6. DDM Access Method Commands Supported for Distributed FileManager/MVS Stream Files (continued)

Access Commands	STRAM
UNLSTR	YES

Legend:
YES The command is supported.

DDM Record Access Restrictions

Restrictions for applications doing record access to Distributed FileManager/MVS are:

- Distributed FileManager/MVS does not support the following functions for accessing multivolume data sets:
  - Backward processing functions: DDMSetMinus and DDMSetPrevious
  - Direct positioning functions: DDMSetBOF, DDMSetEOF, DDMSetFirst, and DDMSetLast
  - DDMInsertRecEOF function (An alternative is to use the DDMLoadFileFirst function to write records to an empty file or to extend the file.)
  - Under some conditions, DDMGetRec and DDMModifyRec functions for accessing records that span physical volumes

  The equivalent DDM commands for these DDM record access functions are SETPRV, SETMNS, SETBOF, SETEOF, SETFRS, SETLST, INSRECEF GETREC and MODREC. The TRGNSPRM reply message is returned if these commands are used to access multivolume data sets.
- The DDMOpen AccIntList includes DDM_MODAI, DDM_INSAI, and DDM_GETAI bit flags. You must explicitly state all your access intents for the duration of the file being open under control of that DDMOpen command.

  There is not a more powerful access intent which implicitly permits another less powerful access. For example, if you specify DDM_MODAI but not DDM_GETAI and then attempt to retrieve a record, you will receive an error reply message.
- The following are other Distributed FileManager/MVS access restrictions:
  - For the DDMModifyRec command, the AccessFlags DDM_INHMODKY bit flag must be set on.
  - For DDMUnLoadFileFirst, you must explicitly specify UnloadOrder KEYORD for KSDS.
  - For DDMSetKey, the AccessFlag DDM_HLDCSR bit flag must be set off.
  - DDMSetLast with the AccessFlag DDM_RECNBRFB bit flag set on for ESDS and partitioned sequential data sets returns the special value of -1 for the record number feedback. This indicates that the number is not known.
- If you are accessing KSDSs or AIFs, you can only use DDM keyed access method commands. You can only access these records by key, not by record number.
- If you are accessing a PDS member:
  - Only sequential load (using INSRECEF or LODRECF) is supported; random load (by record number) is not supported.
  - These DDM access commands are not supported: DELREC, INRECNB, SETEOF, and SETLST.

Stream File API Restrictions

Stream file API restrictions, from OS/2 DDM clients:

The OS/2 E editor currently reports a critical error when you create a remote stream file using Distributed FileManager/MVS. The file is, however, actually created and can subsequently be accessed without E editor error messages.

Logon Mode Requirements

The IBM OS/400 and OS/2 DDM clients have the following logon mode requirements:

- It requires a logon mode named QPCSUPP in order to perform authorization checking.
- It only supports one logon mode name specification to be used for all target systems.
- If you do not successfully place a logon mode table entry named QPCSUPP in the MVS VTAM logon mode table concatenation for your MVS APPC APPL, you will receive error messages, see Figure 36, on your MVS console:

```
 IST663I BFINIT REQUEST FROM DFMNCP (my NCP major node) FAILED,
SENSE=...
 IST664I REAL OLU=PELNET01.PS2ILU1    REAL DLU=PELNET01.DFMILU1
 IST889I SID=...
 IST264I REQUIRED LOGMODE NAME QPCSUPP UNDEFINED
 IST314I END
 IST663I BFTERM REQUEST FROM DFMNCP RECEIVED, SENSE=...
 IST664I (same as before)
 IST889I SID=...
 IST891I PELNET01.VTAMF GENERATED FAILURE NOTIFICATION
 IST893I ORIGINAL FAILING REQUEST IS BIND
 IST314I END
```

Figure 36. Error Messages

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.

IBM may use or distribute any of the information you supply in any way it believes appropriate without incurring any obligation to you.

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 Corporation
Information Enabling Requests
Dept. DZWA
5600 Cottle Road
San Jose, CA 95193 U.S.A.

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, IBM International Program License Agreement, or any equivalent agreement between us.

Programming Interface Information

This publication documents intended Programming Interface that allow the customer to write programs to obtain services of DFSMS.

Trademarks

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

AnyNet	IBM
ADSTAR	IMS
AIX	Language Environment
Application System/400	MVS/ESA
BookManager	OpenEdition
CICS	Operating System/2
CICS/ESA	OS/2
CICS/MVS	OS/390
DFSMS	OS/400
DFSMS/MVS	RACF
DFSMSdfp	RMF
DFSMSdss	SystemView
DFSMShsm	SOMobjects
DFSMSrmm	VisualLift
DFSORT	VTAM
ESCON	z/OS
FFST/MVS	
GDDM	

Other company, product, and service names, which may be denoted by a double asterisk (**), may be trademarks or service marks of others.

Glossary

A

access method. (1) A mainframe data management routine that moves data between storage and an I/O device in response to requests made by a program. (2) The part of the distributed data management architecture which accepts commands to access and process the records of a file.

ACS. See *Automatic class selection (ACS)*.

Advanced Program-to-Program Communications (APPC). An implementation of the Systems Network Architecture (SNA) logical unit (LU) 6.2 protocol that allows interconnected systems to communicate and share the processing of programs.

ADSM. ADSTAR Distributed Storage Manager

agent. Manages the parsing and routing of DDM commands and replies.

AIF. See *Alternate index file.*

AIX. Advanced Interactive Executive

alias. An alternative name for an ICF user catalog, a non-VSAM file, or a member of a partitioned data set (PDS) or PDSE.

alternate index file. A file that supports keyed forms of access to the records of a base file.

API. See *application programming interface (API).*

APPC. See *Advanced Program-to-Program Communications (APPC).*

APPC/MVS. In MVS/SP™, a new session environment in support of LU 6.2 transaction scheduling and communications. The MVS implementation of APPC.

application programming interface (API). A formally defined programming language interface between an IBM system control program or a licensed program and the user of a program.

architecture. A set of defined terms and rules used as instructions to build products.

ascending key sequence. Specifies that the records of a file are in ascending key sequence. If the key class is BYTSTRDR (byte string), the collating sequence is a simple binary sequence with X'00' as the lowest value and X'FF' as the highest value.

ASCH. APPC/MVS scheduler. See also *APPC/MVS*.

ASCII. American National Standard Code for Information Interchange

ASID. Address space identifier

associated DDM attributes. Associated DDM attributes are MVS/ESA data set attributes that are defined in DDM. Examples of associated DDM attributes are file size, lock options or end-of-file offset for byte-stream files. Associated DDM attributes are not necessarily exclusive to DDM, but can be common to other applications that access the same data sets.

automatic class selection (ACS). A mechanism for assigning Storage Management Subsystem classes and storage groups to data sets.

automatic class selection (ACS) routine. A procedural set of ACS language statements. Based on a set of input variables, the ACS language statements generate the name of a predefined SMS class, or a list of names of predefined storage groups, for a data set.

B

backup. The process of creating a copy of a data set or object to be used in case of accidental loss.

base data set. Data set or file stored on MVS, in contrast to the view of the file as seen by the workstation. Is also used to refer to the VSAM ESDS or KSDS upon which an alternate index is built.

BCP. Base control program

BDAM. Basic direct access method

BSAM. Basic sequential access method

byte. The amount of storage required to represent one character; the basic unit of data.

byte stream. A simple sequence of bytes stored in a stream file.

C

C language. A language used to develop software applications in compact, efficient code that can be run on different types of computers with minimal change.

CCSID. Coded character set identifier

CD. Change directory

CDRA. Character Data Representation Architecture

CL. Control language

client. (1) A user. (2) A consumer of resources or services. (3) A functional unit that receives shared services from a server. (4) A system that is dependent on a server to provide it with programs or access to programs. (5) On a network, the computer requesting services or data from another computer.

client-server. (1) In TCP/IP, the model of interaction in distributed data processing in which a program at one site sends a request to a program at another site and waits for a response. The requesting program is called a client; the answering program is called a server. (2) A model of computer interaction in which a server provides resources for other systems on a network, and a client accesses those resources. See also *client, server*.

code point. Specifies the data representation of a dictionary code point. Code points are hexadecimal aliases for the named terms of DDM architecture. Code points are used to reduce the number of bytes required to identify the class of an object in memory and in data streams.

command. A message sent to an object requesting that the object carry out one of its operations.

communications manager. Manages the use of the system's communication facilities.

conversation. In Advanced Program-to-Program Communications (APPC), a connection between two transaction programs over a logical unit-logical unit (LU-LU) session that allows them to communicate with each other while processing a transaction.

conversational transaction. In Advanced Program-to-Program Communications (APPC), two or more programs communicating using the services of logical units (LUs).

cursor. A cursor is a displayed symbol that acts as a marker to help the user locate a point in text, in a system command, or in storage. Cursors mark file position and access information in Distributed Data Management architecture.

D

DASD volume. A DASD space identified by a common label and accessed by a set of related addresses.

data class. A collection of allocation and space attributes, defined by the storage administrator, that are used to create a data set.

data management services. The storage, organization, and access of data.

data set. In DFSMS, the major unit of data storage and retrieval, consisting of a collection of data in one of several prescribed arrangements and described by control information to which the system has access. In z/OS non-UNIX environments, the terms *data set* and *file* are generally equivalent and sometimes are used interchangeably. See also *file*. In z/OS UNIX® environments, the terms *data set* and *file* have quite distinct meanings.

data stream. All data transmitted through a data channel in a single read or write operation.

DCAS. DFM central address space

DDM. See *Distributed Data Management Architecture.*

DDM file name. Distributed Data Management file name.

device name. This term is used interchangeably with device number, unit number, and unit name. It is the number by which a specific device is known. For example, and installation with two tape drives might assign them device names 181 and 182.

DFM. Distributed FileManager

DFM/2. Distributed FileManager for OS/2

DFSMS. See *Data Facility Storage Management Subsystem.*

DFSMSdfp. A DFSMS functional component or base element of z/OS, that provides functions for storage management, data management, program management, device management, and distributed data access.

DFSMSdss™. A DFSMS functional component or base element of z/OS, used to copy, move, dump, and restore data sets and volumes.

DFSMShsm™. A DFSMS functional component or base element of z/OS, used for backing up and recovering data, and managing space on volumes in the storage hierarchy.

DFSMSrmm™. A DFSMS functional component or base element of z/OS, that manages removable media.

direct file. A file that contains records that have a relationship between the contents of the record and the record position at which the record is stored.

directory. A file that maps the names of other directories and files to their locations.

distributed computing. Computing that involves the cooperation of two or more machines communicating over a network. Data and resources are shared among the individual computers.

distributed data. Data that is stored in more than one system in a network and is available to remote users and application programs.

distributed data management. A methodology that allows data on one system to be shared and accessed by another system.

Distributed Data Management Architecture (DDM). Distributed Data Management Architecture (DDM) offers a vocabulary and a set of rules for sharing and accessing data among like and unlike computer systems. DDM includes a set of standardized file models for keyed, relative record, sequential, and stream data. It allows users and applications to access data without concern for the location or format of the data.

distributed file. A file that can be accessed by remote applications or remote users. Also, the capability of accessing such a file.

Distributed FileManager/MVS. Distributed FileManager/MVS is an implementation of target (server) support as defined by Distributed Data Management Architecture (DDM). DDM permits systems in an extended enterprise that have DDM source (client) capability to access file data on a DDM target MVS/ESA system. See definitions for *source*, *target*, and *extended enterprise*.

distributed processing. A capability that enables applications and data located at remote sites or processors connected by a communications link to be used as if they were local.

DSAS. Data space address space

DSS. Data set services

E

EBCDIC. Extended binary coded decimal interchange code

extended enterprise. A heterogeneous computing environment that often includes both centralized hosts and distributed workstations connected in a network. Gateways within the extended enterprise provide connections to local area networks (LANs). These LANs can serve any computing systems architecture.

ESDS. Entry-sequenced data set

extent. A file extent is a storage area for records allocated to a file by the server. Extents are not formally architected in DDM.

F

file. A collection of information treated as a unit. In z/OS non-UNIX environments, the terms *data set* and *file* are generally equivalent and are sometimes used interchangeably. See also *data set*.

file class. Refers to the DDM file class (FILCLS) used when writing VSAM for OS/2 or VSAM for AIX applications.

file model. A description of how information is organized and managed within a file.

fixed-length record. A fixed-length record is one whose length is established as an attribute of the file in which it is stored, and can not be changed. Every record in such a file has the same length, which is specified by the record length attribute (RECLEN) of the file.

G

gateway. A functional unit that interconnects two computer networks with different network architectures. A gateway connects networks or systems of different architectures. A bridge interconnects networks or systems with the same or similar architectures.

GDG. Generation data group

GDS. Generation data set

H

heterogeneous computer network. A computer network in which computers have dissimilar architecture, but nevertheless are able to communicate.

HFS. Hierarchical file system

I

ICF. See *Integrated catalog facility (ICF).*

IDCAMS. Integrated catalog access method services

integrated catalog facility (ICF). In the Data Facility Product (DFP), a facility that provides for integrated catalog facility catalogs.

Interactive Storage Management Facility (ISMF). The interactive interface of DFSMS that allows users and storage administrators access to the storage management functions.

IPL. Initial program load

ISAM. Indexed sequential access method

ISMF. See *Interactive Storage Management Facility (ISMF).*

J

JCL. job control language

K

keyed field. The portion of a record which is used (possibly with other key fields) to locate a data record in a keyed file.

KSDS. Key-sequenced data set

L

LAN. See *local area network*.

LDMI. Local data management interface.

LDS. Linear data set (VSAM)

LE. Language environment

local. Local is your reference point when discussing such entities as platforms or applications. For example, when discussing network conversations from the reference point of an MVS/ESA platform, local refers to entities located on the MVS/ESA system. Similarly, when discussing data access methods from the reference point of an MVS/ESA platform, local refers to the MVS access methods. Contrast with *remote*.

local area network (LAN). A computer network located on a user’s premises within a limited geographical area. Communication within a local area network is not subject to external regulations; however, communication across the LAN boundary can be subject to some form of regulation.

local location name. The name by which a system is know to other systems in an SNA network. A local location name is equivalent to an SNA local logical unit name.

locking. The process of restricting resources to provide protection from concurrent users of the system.

logical unit (LU). In SNA, a logical port through which an end user accesses the SNA network in order to communicate with another end user and through which the end user accesses the functions provided by system services.

logical unit 6.2 (LU 6.2). A particular type of Systems Network Architecture (SNA) logical unit (LU) that provides a connection between resources and transactions programs running on different network nodes.

LU. See *logical unit.*

LU 6.2. See *logical unit 6.2*.

M

mainframe. A large computer, particularly one to which other computers can be connected so that they can share facilities the mainframe provides.

management class. A named collection of management attributes describing the retention, backup, and class transition characteristics for a group of objects in an object storage hierarchy.

migration. The process of moving unused data to lower cost storage in order to make space for high-availability data. If you wish to use the data set, it must be recalled. See also *migration level 1* and migration level 2.

migration level 1. DFSMShsm-owned DASD volumes that contain data sets migrated from primary storage volumes. The data can be compressed. See also *storage hierarchy*. Contrast with *primary storage* and *migration level 2*.

migration level 2. DFSMShsm-owned tape or DASD volumes that contain data sets migrated from primary storage volumes or from migration level 1 volumes. The data can be compressed. See also *storage hierarchy*. Contrast with *primary storage* and *migration level 1*.

mode name. The name used by the initiator of a session to designate the characteristics desired for the session, such as traffic pacing values, message-length limits, sync point and cryptography options, and the class of service within the transport network.

MVS/ESA. Multiple Virtual Storage/Enterprise Systems Architecture. A z/OS operating system environment that supports ESA/390.

MVS/ESA SP™. An IBM licensed program used to control the z/OS operating system. MVS/ESA SP together with DFSMS compose the base MVS/ESA operating environment. See also *z/OS*.

O

object storage hierarchy. A hierarchy consisting of objects stored in DB2® table spaces on DASD, on optical or tape volumes that reside in a library, and on optical or tape volumes that reside on a shelf. See also *storage hierarchy.*

Operating System/2® (OS/2). An operating system used in IBM PC AT®, PS/2®, and compatible computers.

optical volume. Storage space on an optical disk, identified by a volume label. See also *volume*.

OS/2. See *Operating System/2*.

P

partitioned data set (PDS). A data set on direct access storage that is divided into partitions, called members, each of which can contain a program, part of a program, or data.

partitioned data set extended (PDSE). A system-managed data set that contains an indexed directory and members that are similar to the directory and members of partitioned data sets. A PDSE can be used instead of a partitioned data set.

partner. In data communications, the remote application program or the remote computer. Also refers to complementary information or function on a remote platform. For example, for Distributed FileManager/MVS to conduct a network conversation requires a local logical unit (LU) on the target MVS/ESA system and a partner LU on the source system.

PDS. See *Partitioned data set*.

PDSE. See *Partitioned data set extended.*

platform. A computer system running a specific operating system connected in a network. For example, OS/2, OS/400, and MVS/ESA are different operating system platforms.

PPT. Program property table

primary space allocation. Amount of space requested by a user for a data set when it is created. Contrast with *secondary space allocation*.

primary storage. A DASD volume available to users for data allocation. The volumes in primary storage are called primary volumes. See also *storage hierarchy*. Contrast with *migration level 1* and *migration level 2.*

protocol. (1) A set of semantic and syntactic rules that determines the behavior of functional units in achieving communication. (2) A specification for the format and relative timing of information exchanged between communicating parties.

Q

QSAM. Queued sequential access method

R

RACF. See *Resource Access Control Facility (RACF).*

record. The basic unit of data stored in a record-oriented file.

record data. Data sets with a record-oriented structure, which are accessed record by record. This data set structure is typical of data sets on VM, MVS, and OS/400 systems.

record-level access. A means of supporting distributed files. Enables an application or user to read and update individual records of files on a remote system without specifying the data's location.

record-oriented file. File with a record-oriented structure that is accessed record by record. This file structure is typical of data sets on VM, MVS, and OS/400 systems. Contrast with *stream-oriented file*.

remote. Remote is relative to your reference point when discussing such entities as platforms or applications. For example, when discussing network conversations from the reference point of an MVS/ESA platform, remote refers to entities that access MVS/ESA data across an network. An OS/2 application accessing the MVS/ESA data would be remote. Contrast with *local*.

Resource Access Control Facility (RACF). An IBM licensed program that is included in z/OS Security Server and is also available as a separate program for the z/OS and VM environments. RACF provides access control by identifying and verifying the users to the system, authorizing access to protected resources, logging detected unauthorized attempts to enter the system, and logging detected accesses to protected resources.

RLS. Record-level sharing

RRDS. Relative record data set

S

SAM. Sequential access method

SDDM/400. DDM source on OS/400

SdU. See *SMARTdata UTILITIES.*

secondary space allocation. Amount of additional space requested by the user for a data set when primary space is full. Contrast with *primary space allocation*.

sequential file. A type of MVS file that has its records stored and retrieved according to their physical order within the file. It must be on a direct access volume.

server. (1) A functional unit that provides shared services to workstations over a network; for example, a file server, a print server, a mail server. (2) On a network, the computer that contains the data or provides the facilities to be accessed by other computers in the network. (3) A program that handles protocol, queuing, routing, and other tasks necessary for data transfer between devices in a computer system.

session. A logical connection between two stations or network addressable units (NAUs) that allows them to communicate.

SMARTdata Utilities (SdU). SMARTdata Utilities (SdU) is a component of ADSTAR Distributed Storage Manager (ADSM) on OS/2 and AIX systems. SdU provides source DDM services.

SMS. See *Storage Management Subsystem (SMS).*

SNA. See *Systems Network Architecture*.

source. Source is the term used in Distributed Data Management Architecture (DDM) to refer to the platform that originates a request for remote data. Source is also known as client. Contrast with *target*.

source server. DDM term for the function that converts source requests to data streams containing DDM commands and output data and sends them over the network to the target server.

source system. A system containing an application program that requests access to data in another system.

SPE. Small programming enhancement

storage administration group. A centralized group within the data processing center that is responsible for managing the storage resources within an installation.

storage administrator. A person in the data processing center who is responsible for defining, implementing, and maintaining storage management policies.

storage class. A collection of storage attributes that identify performance goals and availability requirements, defined by the storage administrator, used to select a device that can meet those goals and requirements.

storage hierarchy. An arrangement of storage devices with different speeds and capacities. The levels of the storage hierarchy include main storage (memory, DASD cache), primary storage (DASD containing uncompressed data), migration level 1 (DASD containing data in a space-saving format), and migration level 2 (tape cartridges containing data in a space-saving format). See also *primary storage*, *migration level 1*, *migration level 2*, and *object storage hierarchy*.

storage management. The activities of data set allocation, placement, monitoring, migration, backup, recall, recovery, and deletion. These can be done either manually or by using automated processes. The Storage Management Subsystem automates these processes for you, while optimizing storage resources. See also *Storage Management Subsystem*.

Storage Management Subsystem (SMS). A DFSMS facility used to automate and centralize the management of storage. Using SMS, a storage administrator describes data allocation characteristics, performance and availability goals, backup and retention requirements, and storage requirements to the system through data class, storage class, management class, storage group, and ACS routine definitions.

Stream data file. Data sets with a byte-oriented structure, which are accessed as continuous streams of data bytes. This data set (file) structure is common in workstation environments.

stream-oriented file. File with a byte-oriented structure that is accessed as continuous streams of data bytes. This file structure is common in workstation environments. Contrast with *record-oriented file*.

system administrator. The person at a computer installation who designs, controls, and manages the use of the computer system.

system operator. An operator responsible for performing system-oriented procedures.

system programmer. A programmer who plans, generates, maintains, extends, and controls the use of an operating system and applications with the aim of improving overall productivity of an installation.

system-managed storage. Storage managed by the Storage Management Subsystem. SMS attempts to deliver required services for availability, performance, and space to applications. See also *system-managed storage environment*.

system-managed storage environment. An environment that helps automate and centralize the management of storage. This is achieved through a combination of hardware, software, and policies. In the system-managed storage environment for z/OS, the function is provided by DFSORT™, RACF, and the combination of DFSMS and z/OS.

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

T

tape volume. A tape volume is the recording space on a single tape cartridge or reel. See also *volume*.

target. Target is the term used in Distributed Data Management Architecture (DDM) to refer to the platform that fulfills a request for remote data. Target is also known as server. Contrast with *source*.

target server. DDM term that describes the function that converts DDM data streams received from a source server to local data management requests and sends reply messages and input data back to the source server over a network.

target system. A system containing data that has been requested by another system.

TCP/IP. See *Transmission Control Protocol/Internet Protocol (TCP/IP).*

TP. See *Transaction program (TP).*

transaction program (TP). A program that uses the Advanced Program-to-Program Communications (APPC) application programming interface (API) to communicate with a partner application program on a remote system.

Transmission Control Protocol/Internet Protocol (TCP/IP). The two fundamental protocols of the Internet protocol suite. The abbreviation TCP/IP is frequently used to refer to this protocol suite. TCP/IP provides for the reliable transfer of data, while IP transmits the data through the network in the form of datagrams. Users can send mail, transfer files across the network, or execute commands on other systems.

TSO. Time Sharing Option

U

user interface. (1) The means by which a user communicates with a system, program, or device. (2) The hardware, software, or both that implements a user interface, allowing the user to interact with and perform operations on a system, program, or device. Examples are a keyboard, mouse, command language, or windowing subsystem.

V

variable-length record. A variable-length record is one whose length can be changed after it has been written to a file. The length of individual records in the file varies from record to record but cannot exceed the maximum length specified by the RECLEN attribute of the file. The length of a record is initially set by the DDMInsertRecEOF, DDMInsertRecNum or DDMInsertRecKey function, but can be changed by a subsequent function (DDMModifyRec, DDMInsertRecNum, DDMInsertRecKey, or DDMDeleteRec).

Virtual Storage Access Method for OS/2 and AIX. A record-oriented application programming interface allowing applications to open, access, modify, and close record-file data. The local record access component of SdU.

volume. The storage space on DASD, tape, or optical devices, which is identified by a volume label. See also *DASD volume*, *optical volume*, and *tape volume*.

VRRDS. Variable-length relative record data set

VSAM. Virtual Storage Access Method

VSAM for OS/2 and AIX. See *Virtual Storage Access Method for OS/2 and AIX.*

VSE. Virtual Storage Extended

VTAM. Virtual Telecommunications Access Method

W

wild card. A character or sequence of characters that can be included in a character string to represent zero or more characters in the string.

workstation. (1) A device that enables users to transmit information to or receive information from a computer; for example, a display station or printer. (2) A functional unit at which a user works. It can be a programmable workstation, such as an IBM PS/2 computer, or a nonprogrammable workstation, such as a terminal. (3) A terminal or microcomputer, usually one that is connected to a mainframe or to a network, at which a user can perform applications.

WTO. Write-to-Operator

Index

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

z/OS
Distributed FileManager Guide and Reference
Release 1

Publication No. SC26-7395-00

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

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

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

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

Please tell us how we can improve this book:

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

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

Name	Address
Company or Organization	
Phone No.	

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

SC26-7395-00

Cut or Fold
Along Line

Fold and Tape

Please do not staple

Fold and Tape

NO POSTAGE
NECESSARY
IF MAILED IN THE
UNITED STATES

BUSINESS REPLY MAIL

FIRST-CLASS MAIL PERMIT NO. 40 ARMONK, NEW YORK

POSTAGE WILL BE PAID BY ADDRESSEE

International Business Machines Corporation
RCF Processing Department
M86 050
5600 Cottle Road
San Jose, CA
U.S.A. 95193-0001

Fold and Tape

Please do not staple

Fold and Tape

SC26-7395-00

Cut or Fold
Along Line

Program Number: 5694-A01

Printed in the United States of America
on recycled paper containing 10%
recovered post-consumer fiber.

SC26-7395-00

Spine information:

z/OS

z/OS V1R1.0 Distributed FileManager Guide and Reference

Release 1