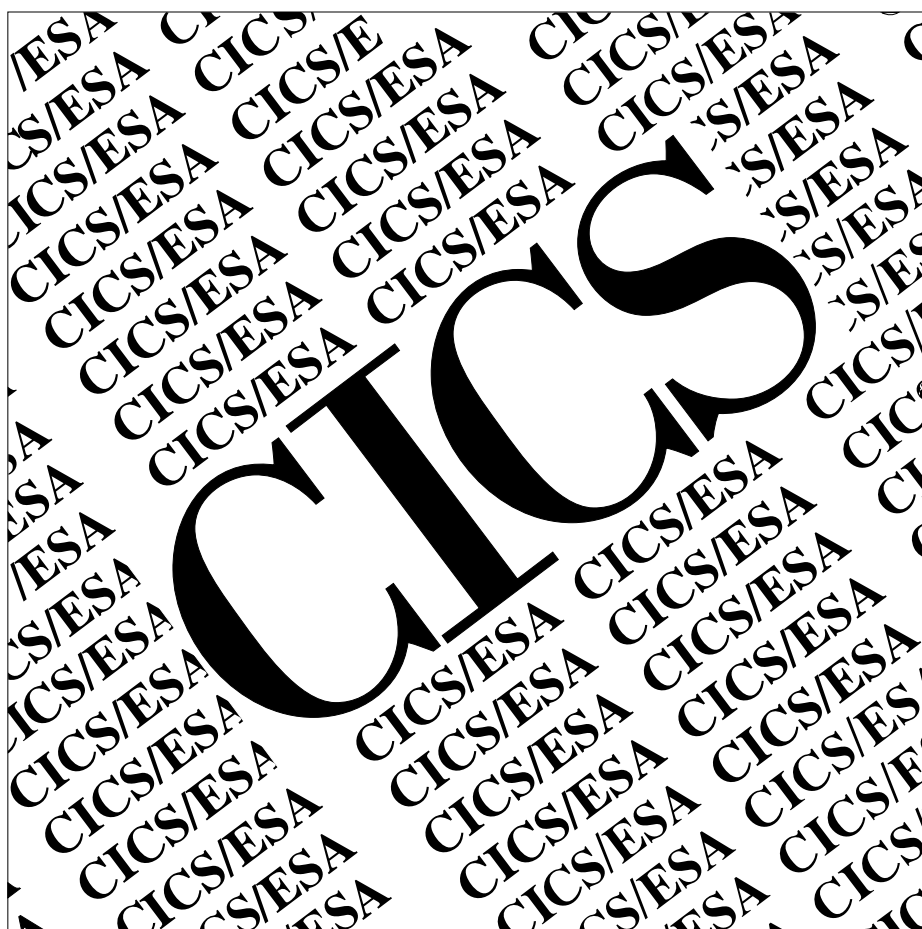


CICS/ESA



# Shared Data Tables Guide

*Version 4 Release 1*





CICS/ESA



# Shared Data Tables Guide

*Version 4 Release 1*

**Note!**

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

**First edition (October 1994)**

This edition applies to Version 4 Release 1 of the IBM licensed program Customer Information Control System/Enterprise Systems Architecture (CICS/ESA), program number 5655-018, and to all subsequent versions, releases, and modifications until otherwise indicated in new editions. Consult the latest edition of the applicable IBM system bibliography for current information on this product.

This book is based on the Shared Data Tables Guide for CICS/ESA 3.3, SC33-0887-02. Changes from that edition are marked by vertical lines to the left of the changes.

The CICS/ESA 3.3 edition remains applicable and current for users of CICS/ESA 3.3.

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

At the back of this publication is a page entitled "Sending your comments to IBM". If you want to make comments, but the methods described are not available to you, please address them to:

IBM United Kingdom Laboratories Limited, Information Development,  
Mail Point 095, Hursley Park, Winchester, Hampshire, England, SO21 2JN.

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.

© **Copyright International Business Machines Corporation 1992, 1994. All rights reserved.**

US Government Users Restricted Rights – Use, duplication or disclosure restricted by GSA ADP Schedule Contract with IBM Corp.

---

# Contents

<b>Notices</b> . . . . .	ix
Programming interface information . . . . .	ix
Trademarks and service marks . . . . .	x
<b>Preface</b> . . . . .	xi
Determining if a publication is current . . . . .	xi
Bibliography . . . . .	xiii
CICS/ESA 4.1 library . . . . .	xiii
Other CICS books . . . . .	xiv
<b>Summary of changes</b> . . . . .	xv
<b>Chapter 1. Introduction</b> . . . . .	1
Data tables . . . . .	1
Data table sharing environment . . . . .	2
Source data set . . . . .	3
Data space . . . . .	3
Global user exits . . . . .	3
Benefits of shared data tables . . . . .	4
Remote file access . . . . .	4
How a data table is shared . . . . .	5
<b>Chapter 2. CICS-maintained data table</b> . . . . .	9
CMT application programming . . . . .	9
CMT resource definition . . . . .	9
CMT operations . . . . .	10
<b>Chapter 3. User-maintained data table</b> . . . . .	11
UMT application programming . . . . .	11
UMT resource definition . . . . .	11
UMT operations . . . . .	12
<b>Chapter 4. Planning</b> . . . . .	13
Performance benefits . . . . .	13
Selecting files for use as data tables . . . . .	16
Security checking . . . . .	23
SDT support on different releases of CICS . . . . .	23
Planning for SDT support . . . . .	23
<b>Chapter 5. Application programming</b> . . . . .	27
CICS-maintained data table . . . . .	27
User-maintained data table . . . . .	28
Use of cross-memory services . . . . .	29
Differences between function-shipping and cross-memory services . . . . .	31
Differences between SDT services and VSAM . . . . .	32
<b>Chapter 6. Resource definition</b> . . . . .	35
CEDA DEFINE FILE command . . . . .	35
EXEC CICS commands . . . . .	39
CEMT commands . . . . .	40

<b>Chapter 7. Customization</b>	43
Communicating between CICS and exit programs	43
XDTRD user exit	46
XDTAD user exit	46
XD TLC user exit	47
<b>Chapter 8. Operations</b>	49
Opening a data table	49
Closing a data table	50
MVS job control	51
Interpreting data table statistics	51
Activating user exits	60
<b>Chapter 9. Problem determination</b>	63
Trace information	63
Analyzing errors from the SVC	67
Analyzing errors from cross-memory services	70
Dump information	70
<b>Chapter 10. Using Shared Data Tables support in a sysplex</b>	73
How to refresh replicated UMTs	74
Example program for refreshing a UMT	76
<b>Appendix A. Sample user exit programs</b>	93
Sample XDTRD exit program	93
Sample XDTAD exit program	102
Sample XD TLC exit program	108
<b>Index</b>	113

#  
#  
#

---

# Figures

1.	Data access using function shipping	4
2.	Data access using shared data table services	5
3.	CICFOR requested file statistics	19
4.	CICAOR1 requested file statistics	20
5.	CICAOR2 requested file statistics	21
6.	CEDA DEFINE FILE panel	36
7.	Data table user exit parameter list	44
8.	CICFOR requested file statistics	53
9.	CICAOR1 requested file statistics	54
10.	CICAOR2 requested file statistics	55
11.	Example program to refresh a replicated UMT	79
12.	Sample XDTRD user exit program	93
13.	Sample XDTAD user exit program	102
14.	Sample XDTLC user exit program	108

#





## **Tables**

1.	CICS-maintained data table performance for single-region use . . . . .	14
2.	CICS-maintained data table performance for cross-region sharing . . . . .	14
3.	User-maintained data table performance for single-region use . . . . .	15
4.	User-maintained data table performance for cross-region sharing . . . . .	15
5.	Key distribution and format . . . . .	16
6.	Load modules in SDT . . . . .	25
7.	Return codes for XDTRD user exit . . . . .	46
8.	Return codes for XDTAD user exit . . . . .	47
9.	Return codes for XD TLC user exit . . . . .	47
10.	Data tables statistics . . . . .	51



---

## Notices

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 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.

References in this publication to IBM products, programs, or services do not imply that IBM intends to make these available in all countries in which IBM operates.

Any reference to an IBM licensed program or other IBM product in this publication is not intended to state or imply that only IBM's program or other product may be used. Any functionally equivalent program that does not infringe any of IBM's intellectual property rights may be used instead of the IBM product. Evaluation and verification of operation in conjunction with other products, except those expressly designated by IBM, is the user's responsibility.

IBM may have patents or pending patent applications covering subject matter 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 the IBM Director of Licensing, IBM Corporation, 500 Columbus Avenue, Thornwood, New York 10594, U.S.A.

This publication contains sample programs. Permission is hereby granted to copy and store the sample programs into a data processing machine and to use the stored copies for internal study and instruction only. No permission is granted to use the sample programs for any other purpose.

---

## Programming interface information

This book is intended to help you set up and use CICS shared data tables. This book also documents General-use Programming Interface and Associated Guidance Information, Product-sensitive Programming Interface and Associated Guidance Information, and Diagnosis, Modification, or Tuning Information that is provided by CICS.

General-use programming interfaces allow the customer to write programs that obtain the services of CICS.

Programming Interface and Associated Guidance Information is identified where it occurs by an introductory statement to a chapter or section.

Product-sensitive programming interfaces allow the customer installation to perform tasks such as diagnosing, modifying, monitoring, repairing, tailoring, or tuning of CICS. Use of such interfaces creates dependencies on the detailed design or implementation of the IBM software product. Product-sensitive programming interfaces should be used only for these specialized purposes. Because of their dependencies on detailed design and implementation, it is to be expected that programs written to such interfaces may need to be changed in order to run with new product releases or versions, or as a result of service.

Product-Sensitive Programming Interface and Associated Guidance Information is identified where it occurs by an introductory statement to a chapter or section.

Diagnosis, Modification, or Tuning Information is provided to help you diagnose problems and tailor your CICS system.

**Warning:** Do not use this Diagnosis, Modification, or Tuning Information as a programming interface.

Diagnosis, Modification, or Tuning Information is identified where it occurs by an introductory statement to a chapter or section.

---

## Trademarks and service marks

The following terms, used in this publication, are trademarks or service marks of IBM Corporation in the United States or other countries:

ACF/VTAM, CICS, CICS/ESA, CICS/MVS, C/370, DB2, IBM, MVS/DFP, MVS/ESA, RACF, RMF

---

## Preface

### What this book is about

This book gives information about CICS shared data table services.

### Who should read this book

This book is for anyone who is involved with CICS shared data tables in one or more of the following areas:

- Planning
- Application programming
- Resource definition
- Customization
- Operations
- Problem determination

### What you need to know to understand this book

You need to have a good understanding of the area of CICS that you are responsible for. This can be obtained from the other books in the CICS library.

You should also understand how the following terms are used in this book:

#### Browse request

A STARTBR, RESETBR, ENDBR, READNEXT, or READPREV application programming command.

#### Imprecise key

A record key that is specified with the GENERIC or GTEQ option in an application programming command.

#### Precise key

A record key that is specified without the GENERIC or GTEQ option in an application programming command.

#### Update request

A WRITE, DELETE, READ UPDATE, or REWRITE application programming command.

#### Gap

When records are omitted from a CICS-maintained data table but are present in the source data set, the range of omitted keys is referred to as a "gap" in the key sequence.

## Determining if a publication is current

IBM regularly updates its publications with new and changed information. When first published, both hardcopy and BookManager softcopy versions of a publication are in step, but subsequent updates will probably be available in softcopy before they are available in hardcopy.

For CICS books, these softcopy updates appear regularly on the *Transaction Processing and Data Collection Kit* CD-ROM, SK2T-0730-xx. Each reissue of the collection kit is indicated by an updated order number suffix (the -xx part). For

example, collection kit SK2T-0730-06 is more up-to-date than SK2T-0730-05. The collection kit is also clearly dated on the cover.

Here's how to determine if you are looking at the most current copy of a publication:

- A publication with a higher suffix number is more recent than one with a lower suffix number. For example, the publication with order number SC33-0667-02 is more recent than the publication with order number SC33-0667-01. (Note that suffix numbers are updated as a product moves from release to release, as well as for hardcopy updates within a given release.)
- When the softcopy version of a publication is updated for a new collection kit the order number it shares with the hardcopy version does not change. Also, the date in the edition notice remains that of the original publication. To compare softcopy with hardcopy, and softcopy with softcopy (on two editions of the collection kit, for example), check the last two characters of the publication's filename. The higher the number, the more recent the publication. For example, DFHPF104 is more recent than DFHPF103. Next to the publication titles in the CD-ROM booklet and the readme files, asterisks indicate publications that are new or changed.
- Updates to the softcopy are clearly marked by revision codes (usually a “#” character) to the left of the changes.

# Bibliography

## CICS/ESA 4.1 library

<b>Evaluation and planning</b>		
<i>Release Guide</i>	GC33-1161	April 1997
<i>Migration Guide</i>	GC33-1162	April 1997
<b>General</b>		
<i>CICS Family: Library Guide</i>	GC33-1226	April 1995
<i>Master Index</i>	SC33-1187	October 1994
<i>User's Handbook</i>	SX33-1188	April 1997
<i>Glossary (softcopy only)</i>	GC33-1189	n/a
<b>Administration</b>		
<i>Installation Guide</i>	GC33-1163	April 1997
<i>System Definition Guide</i>	SC33-1164	April 1997
<i>Customization Guide</i>	SC33-1165	April 1997
<i>Resource Definition Guide</i>	SC33-1166	April 1997
<i>Operations and Utilities Guide</i>	SC33-1167	April 1997
<i>CICS-Supplied Transactions</i>	SC33-1168	April 1997
<b>Programming</b>		
<i>Application Programming Guide</i>	SC33-1169	October 1994
<i>Application Programming Reference</i>	SC33-1170	April 1997
<i>System Programming Reference</i>	SC33-1171	April 1997
<i>Sample Applications Guide</i>	SC33-1173	October 1994
<i>Distributed Transaction Programming Guide</i>	SC33-1174	October 1994
<i>Front End Programming Interface User's Guide</i>	SC33-1175	October 1994
<b>Diagnosis</b>		
<i>Problem Determination Guide</i>	SC33-1176	October 1994
<i>Messages and Codes</i>	GC33-1177	April 1997
<i>Diagnosis Handbook</i>	LX33-6093	October 1994
<i>Diagnosis Reference</i>	LY33-6082	April 1997
<i>Data Areas</i>	LY33-6083	April 1997
<i>Supplementary Data Areas</i>	LY33-6081	October 1994
<i>Closely-Connected Program Interface</i>	LY33-6084	November 1996
<b>Communication</b>		
<i>Intercommunication Guide</i>	SC33-1181	April 1997
<i>Server Support for CICS Clients</i>	SC33-1591	February 1996
<i>CICS Family: Inter-product Communication</i>	SC33-0824	October 1996
<i>CICS Family: Communicating from CICS on System/390</i>	SC33-1697	October 1996
<b>Special topics</b>		
<i>Recovery and Restart Guide</i>	SC33-1182	October 1994
<i>Performance Guide</i>	SC33-1183	October 1994
<i>CICS-IMS Database Control Guide</i>	SC33-1184	October 1994
<i>CICS-RACF Security Guide</i>	SC33-1185	October 1994
<i>Shared Data Tables Guide</i>	SC33-1186	October 1994
<i>External CICS Interface</i>	SC33-1390	April 1997
<i>CICS ONC RPC Feature for MVS/ESA Guide</i>	SC33-1119	February 1996
<i>CICS Web Interface Guide</i>	SC33-1892	November 1996

The book that you are reading was published in hardcopy format on the date shown in the right-hand column in the above table. Some of the CICS/ESA 4.1 books were republished in new hardcopy editions in April 1997 to incorporate updated information previously available only in softcopy. Note that it is possible that other books in the library will be updated after April 1997.

When a new order is placed for the CICS/ESA 4.1 product, the books shipped with that order will be the latest hardcopy editions.

The style of IBM covers changes periodically. Books in this library have more than one style of cover.

For information about the softcopy books, see “Determining if a publication is current” on page xi. The softcopy books are regularly updated to include the latest information.

## **Other CICS books**

- *CICS Application Migration Aid Guide*, SC33-0768
- *CICS Application Programming Primer (VS COBOL II)*, SC33-0674
- *CICS/ESA Facilities and Planning Guide* for CICS/ESA Version 3 Release 3, SC33-0654
- *CICS/ESA XRF Guide* for CICS/ESA Version 3 Release 3, SC33-0661
- *CICS Family: API Structure*, SC33-1007
- *CICS Family: General Information*, GC33-0155
- *IBM CICS Transaction Affinities Utility MVS/ESA*, SC33-1159

## **CICS Clients**

- *CICS Clients: Administration*, SC33-1436
- *CICS Family: Client/Server Programming*, SC33-1435



---

## Summary of changes

### Changes for this (CICS/ESA 4.1) edition

In CICS/ESA 4.1, the Shared Data Tables (SDT) facility has been incorporated into the base product. This manual has been rewritten to reflect this, and includes updated statistics. The messages and abend codes that were formerly in the Problem Determination chapter have now been moved to the *CICS/ESA Messages and Codes* manual. Details of SDT security have been moved to the *CICS/ESA CICS-RACF Security Guide*.

A new chapter has been added, telling you how to obtain some of the benefits of shared data tables support in a sysplex environment. See Chapter 10, "Using Shared Data Tables support in a sysplex" on page 73.

### Changes for the CICS/ESA Version 3 edition

This edition is a major revision of the *CICS/MVS Data Tables Guide*, SC33-0632-01. It is now applicable to CICS/MVS Version 2 and CICS/ESA Version 3.

**Note:** Because this book applies to several CICS releases, it is possible that some of the information is not applicable to your installation, even if this is not specifically indicated.

The information in the book has been extensively reorganized and rewritten. The organization is based on the user tasks that are used as the basis of the organization of the whole CICS library.

More information is included in the areas of planning and application programming.

The information on resource definition includes a description of how to use:

- Resource definition online (RDO) for files, which was introduced in CICS/ESA release 3.1.
- EXEC CICS SET FILE and EXEC CICS INQUIRE FILE commands, which have been extended to support data tables in CICS/ESA release 3.3.

Information has been added for the new trace points introduced in CICS/ESA release 3.1.



---

## Chapter 1. Introduction

The CICS Shared Data Table (SDT) facility is an extension of the CICS file management services.

The concept of shared data tables is simple; it exploits the facts that it is more efficient:

- To use MVS/ESA cross-memory services instead of CICS function shipping to share a file of data between two or more CICS regions in the same MVS/ESA image.
- To access data from memory instead of from DASD.
- To access a file of data from memory by using services integrated within CICS file management instead of by using VSAM services and a local shared resource (LSR) pool.

SDT wholly replaces and extends the original data table services that were provided as part of the base product in all CICS/ESA Version 3 releases. Under SDT, all files that are defined as data tables can potentially be shared via cross-memory services; no changes are required to the file definitions for existing data tables.

The use of cross-memory services is one of the major enhancements to data table services that is included in the SDT facility. This enhancement improves the performance of applications that currently use function shipping and makes file sharing feasible for applications that cannot accept the performance overhead of function shipping.

The other major enhancement is that nearly all read requests are supported for use with data tables. This enhancement extends the use of data tables to applications that include:

- Browse requests
- Read requests that use an imprecise key

See “What you need to know to understand this book” on page xi for the definition of application programming terms used in this book.

---

### Data tables

A CICS file is a representation of a data set on DASD. If you specify that the file is to use data table services, CICS copies the contents of the data set into an MVS/ESA data space when the file is opened and uses that copy whenever possible.

Because of the way that the data table services access the records, they can be used only with a VSAM key-sequenced data set (KSDS). The KSDS is called the *source data set*. The copy in memory is called the *data table*. The process of copying the records is called *loading* the data table.

When the file is read by a CICS application, the record is normally retrieved from the data table. When the file is updated by a CICS application, the effect depends on the type of data table that you have defined for the file.

CICS data table services support two types of data table:

- CICS-maintained data table (CMT)
- User-maintained data table (UMT)

## CICS-maintained data table

A CICS-maintained data table is a data table whose records are automatically reflected in the source data set; when you update the file, CICS changes both the source data set and the data table.

A CICS-maintained data table is easy to implement—you need to know little about the data table services, you do not need to change your existing application programs, and full recovery support of the file is retained. CICS-maintained data tables are discussed in more detail in Chapter 2, “CICS-maintained data table” on page 9.

## User-maintained data table

A user-maintained data table is a data table whose records are not automatically reflected in the source data set; when you update the file, CICS changes only the data table.

A user-maintained data table lets you optimize the benefits of using a data table by allowing you to eliminate activity on the source data set, for update requests as well as read requests.

A small number of file operations are not supported for user-maintained data tables. Thus, you might need to make minor changes to existing application programs. Also, recovery of the file is supported after a transaction failure, but not a system failure. User-maintained data tables are discussed in more detail in Chapter 3, “User-maintained data table” on page 11.

---

## Data table sharing environment

The environment for sharing a data table is the same as for any CICS file: one CICS region owns the data table—this region is known as a *file-owning region (FOR)*. Any other region that uses the data table is known as an *application-owning region (AOR)*. In the FOR, the file is known as a *local file* and, in the AOR, the file is known as a *remote file*.

In the context of shared data tables, the FOR is also known as a *server* and the AOR is also known as a *requester*.

The same region can be both an FOR for some data tables and an AOR for others.

For information about these intercommunication concepts, see the *CICS/ESA Intercommunication Guide*.

# Shared data tables support uses cross-region sharing wherever possible to provide  
# access to data tables that are in the same MVS image as the requesting CICS  
# region. This means that most read accesses within the same MVS image are  
# satisfied by cross-region sharing using shared data tables services. If cross-region  
# sharing is not possible for the request, function shipping is used. This means that  
# update requests from CICS regions within the same MVS image and all requests

# from CICS regions in different MVS images use function shipping. Chapter 5,  
# “Application programming” on page 27 tells you when commands are satisfied by  
# either cross-region sharing or function shipping.

# **Note:** Similarly, XCF/MRO does not provide shared data table access between  
# CICS regions in different MVS images.

# Although shared data tables support is primarily intended for sharing data within an  
# MVS image, the support may be extended to a sysplex environment for applications  
# that require only read access to a shared user-maintained data table or can operate  
# with data that might not be up-to-date. The data table must be replicated across  
# each MVS region in the sysplex, and updated periodically. See Chapter 10, “Using  
# Shared Data Tables support in a sysplex” on page 73.

---

## Source data set

The source data set must be a base VSAM KSDS, not an alternate index. However, updates made to the KSDS via an alternate index are reflected in a CICS-maintained data table.

The VSAM definition of the KSDS supplies the values for maximum record length and key length.

---

## Data space

The data table records are stored in an MVS/ESA data space, whether the data table is to be shared by more than one region or not. A separate data space is used for each CICS region; it is obtained when the first file that is defined as a data table is opened in the region; it is used by all CICS data tables that are owned by that region; and it is retained until the shutdown of CICS in the region.

The data space storage that is used by the data table is freed when the file is closed in the FOR. This storage is made available for reuse in such a way that the integrity of any AOR that was using the data table is protected.

---

## Global user exits

Three global user exits are provided to extend the normal processing done by data table services:

- XDTRD, to select the records that are copied to the data table during loading when the file is opened. For a user-maintained data table, it can also be used to modify the records.
- XDTAD, to select the records that are copied to the data table when new records are added to the file.
- XD TLC, to perform processing at the end of the loading operation.

These user exits are fully described in Chapter 7, “Customization” on page 43.

---

## Benefits of shared data tables

SDT offers many additional benefits over the data table services that were included as part of CICS/ESA. For example:

- Very large reductions in path length can be achieved for remote accesses because function shipping is avoided for most read and browse requests.
- When cross-memory services are used, the requests are processed by the AOR, thus freeing the FOR to process other requests. This increases multiprocessor exploitation.
- Increased security of data is provided because the record information in shared data tables is stored outside the CICS region and is not included in CICS system dumps (either formatted or unformatted).
- For CICS-maintained data tables, all forms of non-update, keyed access (including browse requests and imprecise-key read requests) are processed by reference to the data table.
- For user-maintained data tables, all forms of non-update, keyed access (including browse requests and imprecise-key read requests) are supported.
- Any number of files referring to the same source data set that are open at the same time can retrieve data from the one CICS-maintained data table.
- An enhancement to the XDTRD user exit allows you to skip over a range of records while loading the data table.

---

## Remote file access

This section illustrates the differences between using function shipping and using shared data table services to access a CICS file in another region.

### Using function shipping

Figure 1 shows the use of function shipping to access a data set owned by another CICS region.

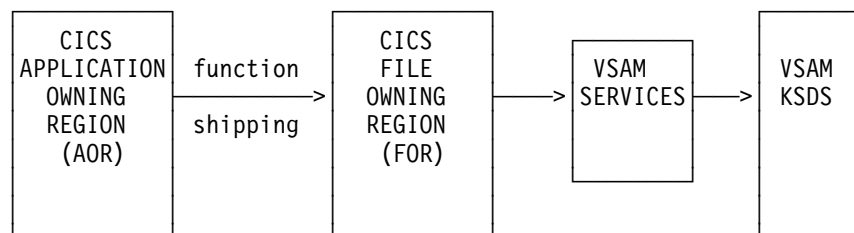


Figure 1. Data access using function shipping

### Using shared data table services

Figure 2 on page 5 shows an AOR using cross-memory services to execute reads or browses, using shared data table services in an FOR in order to access the data table. (Function shipping is used for update requests and for any request that needs to access the source data set, in the same way as shown in Figure 1.)

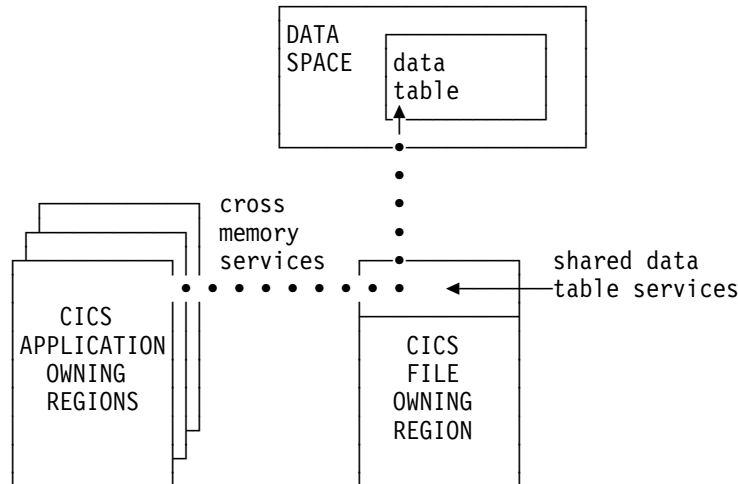


Figure 2. Data access using shared data table services. This diagram shows read-only access only.

---

## How a data table is shared

SDT performs two operations—LOGON and CONNECT—in order to establish a data table for sharing. These operations are described below.

### LOGON

When the first file that is defined as a data table is opened in an FOR, the FOR attempts to register itself as an SDT server. This operation is performed automatically and is known as an **SDT LOGON**. The opening of the file can be caused by the FOR or by the AOR that first accesses the file.

Regardless of whether the LOGON is successful or not, the file is opened and the data table is loaded. If the LOGON is successful, all other CICS regions in the MVS/ESA operating system are notified that the data table is available.

If the LOGON fails because of a permanent condition (such as CICS not being defined as an MVS subsystem), no further LOGON attempts are made during the CICS run.

If the LOGON fails because of a potentially transient condition, another LOGON attempt is made the next time that a file that is defined as a data table is opened. This type of condition includes:

- Failing a security check
- Failing to obtain storage
- Failing to load a program

When a region's LOGON requests are rejected due to failing a security check, security violation messages might be issued each time a file that is defined as a data table is opened.

After an FOR logs on successfully, it remains in that state for the rest of the CICS run; no more LOGON requests are issued.

## CONNECT

When an AOR with SDT issues a read request (or starts a browse sequence) for a remote file, SDT attempts to establish a connection to a data table for that file. If the FOR is registered as an SDT server, SDT establishes a cross-memory link from the AOR to the FOR (subject to security checks) and calls the SDT server to ask whether there is an available data table for the file. If there is, a connection is made between the AOR and the data table. This operation is performed automatically, and is known as an **SDT CONNECT**.

If the **CONNECT** is successful, cross-memory services are then used, whenever possible, to access the file while the connection exists.



If the CONNECT fails, the file request is function shipped exactly as it would have been in the absence of SDT. The action taken for subsequent remote file requests depends on the type of failure, as described in the following paragraphs.

If the CONNECT fails because of a permanent condition (such as CICS not being defined as an MVS subsystem), no further CONNECT attempts are made during the CICS run.

If the CONNECT fails because of a potentially transient condition that is not under the control of the file owner, another CONNECT attempt is made for the next suitable request after approximately ten minutes have elapsed. This type of condition includes:

- Failing a security check
- Failing to obtain storage
- Failing to load a program.

When a region's CONNECT requests are rejected due to failing a security check, the related security violation messages might be issued at ten minute intervals.

If the CONNECT fails because of a potentially transient condition that is under the control of the file owner, another CONNECT attempt is made for the next suitable request following notification that at least one new file is available for shared access on the MVS system. This type of condition includes:

- File owner is not logged on as a server
- File is not associated with a data table
- File is disabled, although associated with a data table
- File is closed, although defined as a data table.

After an AOR connects to a remote file successfully, it remains connected unless one of the following events occurs:

- The AOR deletes its remote file definition.  
In this event, the connection is broken immediately.
- The FOR closes or disables the file.  
In this event, the disconnection is scheduled at the next non-update request and is effected after all current browse sequences have terminated (see "Disconnection" on page 30).

If these events are later reversed, a valid connection is established in the same way as before.

**Notification that a new file is available for shared access:** When a data table is opened by an FOR, it becomes available for CONNECT attempts at the start of loading for a CICS-maintained data table, or at the completion of loading of a user-maintained data table. Other CICS regions are notified that a data table has become available. Notification is also made when a data table (or a file that uses a CICS-maintained data table) is enabled, having been previously disabled.

## Security

To provide security for a data table when cross-memory services are used, SDT must ensure that:

- The FOR cannot be impersonated. This is prevented by checking at LOGON time that the FOR is allowed to log on with the specified generic *applid* of the CICS system.

- An AOR cannot gain access to data that it is not supposed to see. This is prevented by checking at CONNECT time that the AOR is allowed access to the FOR and, if file security is in force, that the AOR is allowed access to the requested file.

These security checks are performed by using the system authorization facility (SAF) to invoke the Resource Access Control Facility (RACF) or an equivalent security manager.

**Note:** A region is still able to use data tables locally even if it does not have authority to act as a shared data table server.

SDT reproduces the main characteristics of function-shipping security that operate at the region level, but the following differences should be noted:

- SDT does not provide any mechanism for the FOR to perform security checks at the transaction level (the equivalent of ATTACHSEC(IDENTIFY) or ATTACHSEC(VERIFY)). Therefore, if you consider that the transaction-level checks performed by the AOR are inadequate, for some files, you must ensure that those files are not associated with data tables in the FOR.
- SDT does not support preset security.
- SDT does not pass any installation parameter list (INSTLN) information to the security user exits.

For a description of the steps required to implement SDT security, see the *CICS/ESA CICS-RACF Security Guide*.

---

## Chapter 2. CICS-maintained data table

If a file is defined as a CICS-maintained data table (CMT), the source data set and the data table are treated by CICS as a single entity. This means that:

- Changes to the file are made to both the source data set and the data table.
- If another file is defined to use the same source data set, changes that are made by that file to the source data set are also made to the data table.
- If another file is defined to use the same source data set, records can be retrieved by that file from the data table.

---

### CMT application programming

All CICS file control commands can be used in applications that access a CICS-maintained data table. This means that the benefits of data tables can be obtained immediately without any changes to existing applications.

CICS uses the data table to perform most read requests. Other requests might need to access the source data set. See Chapter 5, “Application programming” on page 27 for more information.

---

### CMT resource definition

You define a file as a CICS-maintained data table by one of the following methods:

- CEDA DEFINE FILE command
- DFHFCT macro

Also, you can change the definition of an existing file by:

- EXEC CICS SET FILE command
- CEMT SET FILE command

Only the base VSAM cluster can have a CICS-maintained data table based on it. Read requests via alternate index paths do not use the data table, but changes to the source data set via alternate index paths are reflected in the data table.

After a file that is defined as a CICS-maintained data table has been opened, any other non-UMT file (whether defined as a CMT or not) that names the same source data set in its definition automatically uses the same data table. If any of these other files are defined as CMTs, message DFHFC0934 is issued to the console when they are opened. This is not an error situation; the files are opened and use the existing data table whenever possible.

Either fixed-or variable-length record format can be specified for a CICS-maintained data table. The maximum record length that is supported by SDT is 32KB. This length exceeds that supported by CICS file management, which thus imposes the actual limit (see the topic dealing with lengths of areas passed to CICS commands in the *CICS/ESA Application Programming Guide* for information). The maximum number of records that is supported is 16 777 215.

For more information, see Chapter 6, “Resource definition” on page 35.

## VSAM shareoption

If the source data set is allocated with DISP=SHR, there is a risk that it could be updated by a region other than the FOR. If this happened, the data table would no longer match the source data set. To minimize this risk, the VSAM cross-region SHAREOPTION should be set to 1 or 2.

- 1 means that either one region can have update access to the data set or many regions can have read-only access.
- 2 means that one region can have update access to the data set and, at the same time, many regions can have read-only access.

Regardless of the setting of DISP, a warning message is issued if the cross-region SHAREOPTION is 3 or 4, or if it is 2 but the CICS-maintained data table has read-only access (which means another region might be able to update the data set).

## Data integrity

A file that uses a CICS-maintained data table can be defined as a recoverable resource. The source data set is recovered in the normal way after a system or transaction failure:

- After a system failure, the data table is reloaded from the recovered source data set when the file is reopened.
- After a transaction failure, changes that are made to the source data set by dynamic transaction backout are also made to the data table.

Automatic journaling is supported (in the same way as for any other file) for file operations that access the source data set. File operations that do not access the source data set are not journaled.

---

## CMT operations

CICS loads a data table by copying each record from the source data set when the file is opened. A global user exit, XDTRD, can be invoked for each record before it is copied. The user-written exit program can reject records that are not to be copied. If you are using this user exit, you should ensure that the user exit is activated before the file is opened.

For information about writing user exits, see Chapter 7, “Customization” on page 43. For information about activating user exits, see “Activating user exits” on page 60.

---

## Chapter 3. User-maintained data table

If a file is defined as a user-maintained data table (UMT), the source data set and the data table are treated by CICS as separate entities. After a user-maintained data table has been loaded, it is independent of its source data set; the source data set is not updated when the data table is updated. Thus, a user-maintained data table is particularly suited to applications that make frequent updates to data that is of a transitory nature.

If the data table and source data set are updated separately, by defining them as different files, changes to one are not automatically reflected in the other.

---

### UMT application programming

If a request cannot be satisfied from a user-maintained data table, CICS does not access the source data set (as it would for a CICS-maintained data table); instead it returns an exceptional-condition response.

Records that were in the source data set when the data table was opened might be absent from the data table because they were not copied during loading. This could be due to suppression by the user exit XDTRD or some abnormal event such as the data table becoming full.

Some application programming requests are not supported for a user-maintained data table. They include, for example, read requests that use the UPDATE option with an imprecise key. You might need to change existing applications to avoid these requests or to handle the exceptional conditions returned by CICS. For more information, see "User-maintained data table" on page 28.

You can use the user exits in data table services to put only the records that you need to access in the data table; there is no possibility of the source data set being accessed for those that you do not load.

You can also use the user exit XDTRD to modify each record (by selecting only a subset of its fields, for example) when it is loaded.

---

### UMT resource definition

You define a file as a user-maintained data table by one of the following methods:

- CEDA DEFINE FILE command
- DFHFCT macro

You can also change the definition of an existing file by:

- EXEC CICS SET FILE command
- CEMT SET FILE command

You can load multiple user-maintained data tables from the same source data set by using a separate command or macro to define each data table and making all the definitions refer to that data set.

Though a data table must be loaded from a VSAM KSDS, an application can then copy records to a user-maintained data table from any data source that is accessible from the CICS address space. This could be an IMS or DB2 file. The KSDS that is used as the source data set for the data table can be empty; it is needed only to define the maximum record length and the key length and position.

**Variable-length record format must be specified for a user-maintained data table.** The maximum record length that is supported by SDT is 32KB. This length exceeds that supported by CICS file management, which thus imposes the actual limit. (See the topic dealing with lengths of areas passed to CICS commands in the *CICS/ESA Application Programming Guide* for further information.) The maximum number of records supported is 16 777 215.

For more information, see Chapter 6, “Resource definition” on page 35.

## Data integrity

A user-maintained data table can be defined as a recoverable resource. However, changes to the data table are not recorded in the system log. Thus the data table can be recovered after a transaction failure (by dynamic backout) but not after a system failure.

After a system failure, the data table is reloaded from the source data set when the file is reopened. Remember that, at the time of failure, the contents of the source data set and data table would not have been the same unless you had ensured that either:

- No change is made to either, or
- Any change is made to both.

Automatic journaling is supported only for requests that access the source data set during loading. The records that are accessed by the loading process are journaled before user exit XDTRD, and the records that are accessed due to application requests are journaled after user exit XDTRD.

---

## UMT operations

Like a CICS-maintained data table, a user-maintained data table is loaded when the file is opened. However, unlike a CICS-maintained data table, the global user exit XDTRD can be used to both select *and modify* the records from the source data set that are included in the data table.

The user exit XDTAD can be used to select the records that are added to the table after initial loading. This user exit cannot modify the records because, as the records are written by the application, it is assumed that they are already in the format used in the data table.

If you are using these user exits, you should ensure that the user exits are activated before the file is opened.

For information about writing user exits, see Chapter 7, “Customization” on page 43. For information about activating user exits, see “Activating user exits” on page 60.

---

## Chapter 4. Planning

The sole reason for using data tables is to take advantage of the performance benefits that they offer. This chapter discusses:

- Performance benefits of data tables
- Selecting files for use as data tables
- Security checking
- SDT support on different releases of CICS
- Planning for SDT support

---

### Performance benefits

This section contains Diagnosis, Modification, or Tuning Information.

#### Performance of a CICS-maintained data table

If all the data and index records of a file are completely contained in an LSR pool, defining the file as a CICS-maintained data table does not reduce DASD I/O activity. There is, however, considerable potential for reduction in CPU consumption. Also, you might be able to reduce the number of buffers in the LSR pool.

If the file is not completely contained in an LSR pool, using a CICS-maintained data table could result in reductions in both DASD I/O activity and CPU consumption.

The saving of CPU consumption for a CICS-maintained data table, compared with a VSAM KSDS resident in a local shared resource (LSR) pool, depends on the application usage. Table 1 on page 14, and Table 2 on page 14 show comparative performance measurements that can be achieved with CICS/ESA release 3.3, MVS/DFP release 3.3, and MVS/ESA release 4.2.2 running on a uni-processor capable of executing one million instructions per second.

The data table was loaded before taking the measurements. The file was not defined as recoverable, thus no time was spent on logging for either the VSAM file or the data table.

*Table 1. CICS-maintained data table performance for single-region use*

Application request	CPU time for VSAM file	CPU time for CMT	CMT as % of VSAM file
Read	3.69	1.08	29
Read (not found)	5.07	1.01	20
Read for update	3.81	3.82	100
Rewrite	9.83	11.16	114
Delete	13.44	14.81	110
Write	13.10	14.86	113
Start browse	3.54	1.08	31
Read next	1.37	0.99	72
Read previous	1.41	0.99	70
End browse	1.26	0.69	55

*Table 2. CICS-maintained data table performance for cross-region sharing. The AOR and FOR figures apply to the dispatchable unit that is being used, not the address space. Function shipping, when required, was done with MRO.*

Application request	CPU time for VSAM file			CPU time for CMT			
	AOR	FOR	Total	AOR	FOR	Total	% of VSAM
Read	4.76	8.02	12.78	1.28	0	1.28	10
Read (not found)	4.96	9.30	14.26	1.23	0	1.23	9
Read for update	4.75	8.15	12.90	4.77	8.14	12.91	100
Rewrite	4.76	14.08	18.84	4.73	15.18	19.91	106
Delete	4.70	17.68	22.38	4.72	18.76	23.48	105
Write	4.72	17.27	21.99	4.74	18.81	23.55	107
Start browse	4.98	7.71	12.69	1.30	0	1.30	10
Read next	4.75	5.74	10.49	1.18	0	1.18	11
Read previous	4.75	5.70	10.45	1.19	0	1.19	11
End browse	4.67	5.40	10.07	0.72	0	0.72	7

## Performance of a user-maintained data table

After the loading of a user-maintained data table, DASD I/O activity is eliminated from all data table operations, so the saving of CPU consumption compared with a VSAM KSDS resident in an LSR pool is considerable.

Table 3 on page 15, and Table 4 on page 15 show comparative performance measurements that can be achieved with CICS/ESA release 3.3, MVS/DFP release 3.3, and MVS/ESA release 4.2.2 running on a uni-processor capable of executing one million instructions per second. The data table was loaded before taking the measurements.



*Table 3. User-maintained data table performance for single-region use*

Application request	CPU time for VSAM file	CPU time for UMT	UMT as % of VSAM file
Read	3.69	1.08	29
Read (not found)	5.07	1.01	20
Read for update	3.81	1.19	31
Rewrite	9.83	1.33	14
Delete	13.44	1.54	11
Write	13.10	1.91	15
Start browse	3.54	1.08	31
Read next	1.37	0.99	72
Read previous	1.41	0.99	70
End browse	1.26	0.69	55

*Table 4. User-maintained data table performance for cross-region sharing. The AOR and FOR figures apply to the dispatchable unit that is being used, not the address space. Function shipping, when required, was done with MRO.*

Application request	CPU time for VSAM file			CPU time for UMT			
	AOR	FOR	Total	AOR	FOR	Total	% of VSAM
Read	4.76	8.02	12.78	1.28	0	1.28	10
Read (not found)	4.96	9.30	14.26	1.23	0	1.23	9
Read for update	4.75	8.15	12.90	4.76	5.50	10.26	80
Rewrite	4.76	14.08	18.84	4.76	5.49	10.25	54
Delete	4.70	17.68	22.38	4.71	5.71	10.42	47
Write	4.72	17.27	21.99	4.73	6.09	10.82	49
Start browse	4.98	7.71	12.69	1.30	0	1.30	10
Read next	4.75	5.74	10.49	1.18	0	1.18	11
Read previous	4.75	5.70	10.45	1.19	0	1.19	11
End browse	4.67	5.40	10.07	0.72	0	0.72	7

## Storage use

Shared data tables provide efficient use of data in memory. This means that considerable performance benefits are achieved at the cost of some additional use of storage.

This overview of the use of storage assumes that you understand the distinction between various types of storage, such as real and virtual storage, and address space and data space storage.

SDT uses virtual storage as follows:

- Record data is stored in a data space, which is virtual storage separate from address space virtual storage. The total record data storage at loading time is basically the total size of all records (without keys, which are stored in

table-entry storage) plus a small amount of control information. Data space storage is acquired in units of 16MB, and then allocated to individual tables in increments of 128KB.

If many records are increased in length after loading, or new records are added randomly throughout a large part of the file, the amount of storage will be increased, possibly up to twice the original size.

- Table-entry storage is allocated from MVS storage above the 16MB line in the address space of the CICS file-owning region. It is allocated in increments of 32KB.

There is one entry for each record in the table, plus one entry for each gap in the key sequence. The size of each entry is the keylength + 9 bytes, rounded up to the next multiple of 8 bytes.

- Index storage is also allocated from MVS storage above the 16MB line in the address space of the CICS file-owning region. It is allocated in increments of 32KB.

The size of this area depends on the distribution and format of the key values as well as the actual number of records, as indicated in Table 5.

Key distribution	Key format	Bytes per record
Dense (all keys are consecutive)	binary	5.1
	decimal	8.5
	alphabetic	19
Sparse (no keys are consecutive)	decimal	44
	alphabetic	51
Worst possible case	-	76

- ECSA storage is used for some small control blocks that need to be accessed by all regions that share data tables.

Converting a file into a shared data table could lead to an increased use of real storage, but the use of real storage for VSAM LSR buffers might be reduced if few updates are made. Also, an application that currently achieves high performance by replicating read-only tables in each CICS region might be able to make large storage savings by sharing a single copy of each table.

---

## Selecting files for use as data tables

It is not possible to lay down any exact rules about whether a file will benefit from conversion to a shared data table. Many factors need to be taken into consideration, and an analysis of the potential uses of shared data tables support should ideally be undertaken by someone with a knowledge of how the files are used by the various applications, and of the configuration of the CICS regions.

The following checklist gives some general guidance. Additional sources of information that could help you to select the files include:

- File statistics. "Using statistics to select data tables" on page 18 describes how you can use statistics information as one of the inputs to the selection task.
- The LSR pool statistics.

- Trace entries.
- Monitoring data.

However, the most beneficial input to the selection process is a thorough understanding of the applications and the way in which they use the files.

## Checklist

If your installation is using data tables for the first time, the following checklist gives some general principles to help you select files for defining as data tables.

- You should consider using CICS-maintained data tables first, as these are easier to implement. If you use a CICS-maintained data table, no changes are required to the applications. If you use a user-maintained data table, some changes might be required.
- Use a CICS-maintained data table if you need to ensure the integrity of the data table across a CICS restart.
- Use a CICS-maintained data table if you require journaling of updates. If you require journaling of all access requests, the file is not suitable as a data table.
- The exec interface user exits XEIN and XEIOU, and the file control user exits XFCREQ and XFCREQC are not invoked in the file-owning region if a request to access a data table is satisfied by cross-memory services. When selecting a file, you should ensure that successful operation of your application does not depend on any activity performed at these user exits.
- You should be aware of the security implications of sharing a data table, as described in “Security checking” on page 23.
- If a file is frequently accessed from another region, or if it is accessed by many other regions, or if the accesses are predominantly read requests, then the benefits of making it a data table can be very large. Remember that the performance gain for a remote file is greater than for a local file (see “Performance benefits” on page 13).
- For a CICS-maintained data table, select files that have a reasonably high proportion of requests that will access only the data table (see Chapter 5, “Application programming” on page 27). From among those, select the files with the highest usage of these requests, in order to maximize the performance gains.

Information on file usage can be found in the CICS statistics for file control, which are described in the *CICS/ESA Performance Guide*. Not all read requests can take advantage of the data table, so you should check the data table information in the CICS statistics report afterward to verify that the data table is being used effectively. See “Interpreting data table statistics” on page 51 for more information.

- For a user-maintained data table, select files that have a large proportion of update activity but do not require the updates to be recovered across a CICS restart (see “Data integrity” on page 12).
- Use the performance measurements given earlier in this chapter to estimate the approximate CPU savings, bearing in mind any forecasts for future usage.
- Select one or two files with the best estimates. Give preference to a small file over a large file when the estimated savings are similar, because a small file will probably use less real storage.

- Monitor your real storage consumption. If your system is already real-storage constrained, using a large data table could increase your page-in rates. This in turn could adversely affect CICS system performance. Use your normal performance tools, such as RMF II, to look at real-storage usage and paging rates.
- Consider reducing the number of buffers in the LSR pool because the use of data tables could reduce the number of times that the LSR pool is used.
- You can use the user exit XDTRD to select the records included in the data table. In addition, for a user-maintained data table, you can use user exit XDTRD to modify the records. Thus you can optimize the use of virtual and real storage by storing in the data table only the data that you need.
- A very large data table might require more virtual storage than your usual region limit. In this case, you can either increase the region size (using the JCL REGION parameter) or use the user exit XDTRD to suppress some records.

**Note:** The effect of the REGION parameter on requests for storage above the 16MB line in your installation can be modified or overridden by the IEFUSI exit. The IEFUSI exit is described in the *MVS/ESA SPL: System Modifications* manual (GC28-1831).

**Existing data table users:** If your installation already used the basic data tables support available in CICS/ESA Version 3 before you migrated to CICS/ESA 4.1 and SDT support, you might notice a very small performance degradation (up to 3%) with applications that perform a large proportion of READ commands with precise keys and that do not share the data table between regions. On the other hand, with SDT, you can probably extend the use of data tables to other applications that were previously not able to take advantage of the performance benefits of data tables.

## Using statistics to select data tables

This section covers just one of the possible inputs to the selection task—the information available from the file statistics.

Figure 3 on page 19, Figure 4 on page 20, and Figure 5 on page 21 show some extracts from a hypothetical set of file statistics, which are used in the following discussion to demonstrate how CICS statistics can aid the selection process. The statistics are displayed as they would be reported by the CICS/ESA release 4.1 offline formatting utility. Requested file statistics are shown, but Interval or End of Day statistics would be equally suitable. The section of File “Performance Information” statistics, which reports use of VSAM strings and buffers, is not shown here.

The numbers shown in the figures are purely for the purposes of illustration, and you should not expect the statistics at your installation to resemble them at all closely. Similarly, the configuration of CICS regions and files has been chosen to highlight certain points; it is not suggested that this is either a typical or a desirable configuration.

“Interpreting data table statistics” on page 51 discusses the statistics that are reported for files that have been defined as data tables, which you can use to assess the benefits being obtained.

Requested Statistics Report      Collection Date-Time 12/25/99-11:51:51    Last Reset 09:00:00    Applid CICFOR    Jobname SDTGSTF1

FILES - Resource Information

File Name	Dataset Name Base Dataset Name (If Applicable)	Dataset Type	DT Indicator	Time Opened	Time Closed	Remote Name	Remote Sysid	Lsrpool ID
APPLE	CIC01.CICOWN.APPLES	K		07:44:12	OPEN			1
BANANA	CIC01.CICOWN.BANANAS	K		09:45:08	OPEN			1
ORANGE	CIC01.CICOWN.CITRUS	K		10:51:10	OPEN			2
PEAR	CIC01.CICOWN.PEARS	K		07:30:14	OPEN			3

Requested Statistics Report      Collection Date-Time 12/25/99-11:51:51    Last Reset 09:00:00    Applid CICFOR    Jobname SDTGSTF1

FILES - Requests Information

File Name	Get Requests	Get Upd Requests	Browse Requests	Update Requests	Add Requests	Delete Requests	VSAM EXCP Data	Requests Index
APPLE	2317265	1020	0	1019	21	1	11503	310
BANANA	536452	1674	20344	1674	908	0	2651	70
ORANGE	2069454	98560	17831	98327	4543	2563	8511	481
PEAR	45871	65493	6512	65493	30109	362	3773	231
*TOTALS*	4969042	166747	44687	166513	35581	2926		

Requested Statistics Report      Collection Date-Time 12/25/99-11:51:51    Last Reset 09:00:00    Applid CICFOR    Jobname SDTGSTF1

FILES - Data Table Requests Information

File Name	Close Type	Read Requests	Recs in Table	Adds from Reads	Add Requests	Adds rejected - Exit	Adds rejected - Table Full	Rewrite Requests	Delete Requests	Highest Table Size	Storage Alloc(K)
-----------	------------	---------------	---------------	-----------------	--------------	----------------------	----------------------------	------------------	-----------------	--------------------	------------------

DFHST0223 I There are no data table statistics to report.

Figure 3. CICFOR requested file statistics

Requested Statistics Report      Collection Date-Time 12/25/99-11:51:38    Last Reset 09:00:00    Applid CICAOR1    Jobname SDTGSTA1

FILES - Resource Information

File Name	Dataset Name Base Dataset Name (If Applicable)	Dataset Type	DT Indicator	Time Opened	Time Closed	Remote Name	Remote Sysid	Lsrpool ID
APPLE	REMOTE			CLOSED	CLOSED	APPLE	CIF1	N
BANANA	REMOTE			CLOSED	CLOSED	BANANA	CIF1	N
ORANGE	REMOTE			CLOSED	CLOSED	ORANGE	CIF1	N
ZUCCHINI	REMOTE			CLOSED	CLOSED	COURGETT	CIA2	N

Requested Statistics Report      Collection Date-Time 12/25/99-11:51:38    Last Reset 09:00:00    Applid CICAOR1    Jobname SDTGSTA1

FILES - Requests Information

File Name	Get Requests	Get Upd Requests	Browse Requests	Update Requests	Add Requests	Delete Requests	VSAM Data	EXCP Index	Requests
APPLE	1158701	532	0	531	11	1	0	0	0
BANANA	305641	0	19067	0	0	0	0	0	0
ORANGE	58709	32854	4265	32621	1018	1001	0	0	0
ZUCCHINI	78914	0	14765	0	0	0	0	0	0
*TOTALS*	1601965	33386	38097	33152	1029	1002			

Requested Statistics Report      Collection Date-Time 12/25/99-11:51:38    Last Reset 09:00:00    Applid CICAOR1    Jobname SDTGSTA1

FILES - Data Table Requests Information

File Name	Close Type	Read Requests	Recs in Table	Adds from Reads	Add Requests	Adds rejected - Exit	Adds rejected - Table Full	Rewrite Requests	Delete Requests	Highest Table Size	Storage Alloc(K)
-----------	------------	---------------	---------------	-----------------	--------------	----------------------	----------------------------	------------------	-----------------	--------------------	------------------

DFHST0223 I There are no data table statistics to report.

Figure 4. CICAOR1 requested file statistics

Requested Statistics Report      Collection Date-Time 12/25/99-11:49:31    Last Reset 09:00:00    Applid CICAOR2    Jobname SDTGSTA2

FILES - Resource Information

File Name	Dataset Name Base Dataset Name (If Applicable)	Dataset Type	DT Indicator	Time Opened	Time Closed	Remote Name	Remote Sysid	Lsrpool ID
COURGETT	CIC02.CICOWN.COURGETT	K		08:22:15	OPEN			1
LEMON	REMOTE			CLOSED	CLOSED	ORANGE	CIF1	N

Requested Statistics Report      Collection Date-Time 12/25/99-11:49:31    Last Reset 09:00:00    Applid CICAOR2    Jobname SDTGSTA2

FILES - Requests Information

File Name	Get Requests	Get Upd Requests	Browse Requests	Update Requests	Add Requests	Delete Requests	VSAM Data	EXCP Requests	Index
COURGETT	78914	27469	14765	27469	336472	0	8212		481
LEMON	2010745	65706	13566	65706	3525	1562	0		0
*TOTALS*	2089659	93175	28331	93175	339997	1562			

Requested Statistics Report      Collection Date-Time 12/25/99-11:49:31    Last Reset 09:00:00    Applid CICAOR2    Jobname SDTGSTA2

FILES - Data Table Requests Information

File Name	Close Type	Read Requests	Recs in Table	Adds from Reads	Add Requests	Adds rejected - Exit	Adds rejected - Table Full	Rewrite Requests	Delete Requests	Highest Table Size	Storage Alloc(K)
DFHST0223 I There are no data table statistics to report.											

Figure 5. CICAOR2 requested file statistics

The examples use a hypothetical configuration of three CICS regions. Most of the files used by CICS applications are owned by the file-owning region CICFOR, and the applications mostly run in the application-owning regions CICAOR1 and CICAOR2. This discussion assumes that each of the data sets shown in the statistics reports is a VSAM base KSDS (as indicated by the Dataset Type of K), so any of them can be defined as data tables.

The 4.1 statistics also show you which file names in one region are defined to access which file names in another region. The Remote Sysid is the name given on the connection between the two regions. In the examples, the SYSID of CICFOR is CIF2 and that of CICAOR2 is CIA2.

**A file with a high read-to-update ratio**

The file APPLE is used by applications that run on the application-owning region CICAOR1. It is defined in CICAOR1 as a remote file, and the file definition points to the file APPLE owned by CICFOR. This file would benefit from being redefined in CICFOR as a CICS-maintained data table because it has a high ratio of remote reads (1158701 Get Requests in the time period covered by the reports) to remote updates (11 adds, 1 delete and 531 updates) as seen in Figure 4 on page 20.

See the *CICS/ESA Performance Guide* for guidance on the meanings of the “FILES - Requests Information” section of a statistics report.

### **A file with a high proportion of remote reads**

The file BANANA is updated and read on CICFOR, but is also accessed by CICAOR1. Because all the remote accesses are reads and browses, with no updates, the applications running in CICAOR1 would probably see large benefits if BANANA was defined as a data table, and the applications on CICFOR would also benefit by reading from the local data table.

### **A file shared by several regions**

From a study of the statistics in Figure 4 on page 20 it might appear that ORANGE is not an especially suitable data table candidate, as the numbers of remote retrievals from CICAOR1 (58709 Get Requests and 4265 Browse Requests) are relatively low. However, the remote file LEMON in CICAOR2 also points to ORANGE in CICFOR, so defining ORANGE in CICFOR as a shared CICS-maintained data table would probably benefit the performance of the applications in both AORs.

### **A good UMT candidate**

The file COURGETT owned by CICAOR2 is accessed via the filename ZUCCHINI in CICAOR1. CICAOR1 only reads or browses the file; any updating is issued by the owning region. Also, it is known that these updates are relevant only to the day's CICS run and do not need to be retained permanently (in fact, they are deleted at shutdown). The file is therefore an excellent candidate for defining as a user-maintained data table. Then, all the updates can be made to the data table without any VSAM I/O activity, and all the remote retrievals can be made without function shipping.

Note that COURGETT could not be defined as a user-maintained data table if CICAOR2 were not a CICS system with SDT support (that is, a CICS/ESA 4.1 system or a CICS/ESA 3.3 system with the SDT feature installed), because browse requests are issued to that file and browsing of a user-maintained data table is supported only by SDT. Also, the statistics do not show whether any of the Get requests specify the GENERIC or GTEQ options; these also are supported for a user-maintained data table only by SDT.

### **A rather poor candidate**

The file PEAR would probably not benefit much from shared data tables support because it is not accessed remotely and has many update and browse requests. Local browsing does not offer as much benefit as either local reading or any form of remote retrieval, because VSAM browsing (apart from processing of the STARTBR command) is very efficient. This analysis, of course, does not consider the relative importance of the various file accesses: the reading might be done by critical applications, but the time taken for updates might not be important.

### **Other possible candidates**

The preceding examples illustrate only a small sample of the possible configurations and uses of files that could benefit from shared data tables support.

You could also use shared data tables support to avoid the need to duplicate files or data tables in each region. And, in addition to looking at existing files, you could consider moving files from an AOR to an FOR where this was not practical before because of the cost of file accesses using function shipping.



---

## Security checking

The security checking that is performed by the SDT LOGON and CONNECT operations is introduced in “How a data table is shared” on page 5. You should consider the implications of the security checks before sharing a file that is associated with a data table.

For information about RACF, function-shipping security, and implementing security checking for shared data tables, see the *CICS/ESA CICS-RACF Security Guide*.

## LOGON security check

To minimize the risk that an application-owning region (AOR) might accept counterfeit data records from a file-owning region (FOR) which is in fact an impostor, LOGON processing includes a security check to verify that the FOR is authorized to act as a server with the specified application name. This check is never bypassed, even when SEC=NO is specified at system initialization.

## CONNECT security checks

The security checks performed at CONNECT time provide two levels of security:

- **Bind security** allows an FOR that runs without CICS file security to be able to restrict shared access to selected AORs. (Running without file security minimizes run-time overheads and the number of security definitions.)
- **File security** can be activated in the FOR if you need a finer granularity of security checking. Then, SDT implements those checks that apply to the AOR as a whole.

SDT provides no way of implementing those security checks that an FOR makes at the transaction level when ATTACHSEC(IDENTIFY) or ATTACHSEC(VERIFY) is used with function shipping.

---

## SDT support on different releases of CICS

To benefit from the cross-memory support provided by SDT, you must be running with SDT support in both the requesting and serving CICS systems. This means that each system must either be running CICS/ESA 4.1, or be a CICS/ESA 3.3 system with the SDT feature installed.

If only the requesting CICS system has SDT support, there is no effect apart from the very small overhead of occasional attempts to determine whether the server system supports sharing. All requests continue to be function shipped.

If only the serving CICS system has SDT support, all requests continue to be function shipped by the requester. The requester does, however, obtain the benefits of local data table accesses made by the server.

---

## Planning for SDT support

To use SDT support, you must perform the following tasks. Some of them will already have been done for an installation that currently uses function shipping and/or data tables.

- Either ensure that the following modules are in an authorized system library in the LNKLST of the MVS system, or move them into a library in the LPALST concatenation.
  - DFHDTSVC and DFHDTCV, because all regions using shared data tables must use the same level of SVC code
  - DFHMVRMS, the RESMGR exit stub, because CICS JOBLIB/STEPLIB data sets are unavailable at end-of-memory.

These modules are placed by the installation of CICS/ESA 4.1 into the target library SDFHLINK, which is normally included in the LNKLST concatenation.

- If SDFHLINK is in the LNKLST concatenation, you should issue the operator command `MODIFY LLA,REFRESH` and wait for the confirmatory message `CSV210I LIBRARY LOOKASIDE REFRESHED` in order to make the modules available.
- If SDFHLINK is not in the LNKLST concatenation, you should either copy the modules into a suitable library that is included and issue an LLA refresh, or copy the modules into a library in the LPALST concatenation and re-IPL the MVS system specifying CLPA.
- If any files in any AOR are to exploit sharing, then make sure that CICS is defined as an MVS subsystem.
- Define security authorization so that FORs can act as SDT servers and AORs can access files owned by servers, depending on the level of security required. In a single MVS image:
  - Any number of FORs can act as SDT servers
  - A single AOR can use any number of these FORs
  - A single FOR can serve any number of AORs
  - A region can act as an AOR for one data table and as an FOR for a different data table.
- If, for some reason, two FORs have the same APPLID, at any given time SDT ensures that only one of these FORs is used as an SDT server. However, there is nothing to prevent one FOR acting as an SDT server and another FOR, with the same APPLID, being used for function shipped requests. You should check that your operational procedures do not allow this because, otherwise, there is a risk that data table requests that use SDT services will not be directed to the same region as requests that use function shipping.
- Define those files in the FOR which are to be data tables as either CICS-maintained data tables or user-maintained data tables.
- Create additional remote file definitions in the AOR if required. No changes are needed to existing remote file definitions.
- For any AOR that is to share data tables, specify `ISC=YES` as a system initialization parameter and define MRO or ISC links to the relevant FORs.
- Before using shared data tables, you might need to change some of your JCL statements, modify your operational procedures, or increase the value of the `MAXUSER` MVS initialization parameter. For more information, see “MVS job control” on page 51.

## Load modules

Table 6 shows the load modules that need to be installed in your CICS system in order to use SDT.

*Table 6. Load modules in SDT*

Load module	Load library	How loaded	Description
DFHDTINS	SDFHLOAD	CICS load above the 16MB line	Initialization
DFHDTSVC	SDFHLINK	MVS LOAD above the 16MB line from link-list	Performs all functions that need MVS authorization
DFHDTFOR	SDFHAUTH	MVS LOAD above the 16MB line	Data table FOR module
DFHDTAM	SDFHAUTH	MVS LOAD into subpool 252 storage above the 16MB line	Data table access manager. It includes code that is executed in cross-memory mode from an AOR
DFHDTAOR	SDFHAUTH	MVS LOAD above the 16MB line	Data table AOR module
DFHDTCV	SDFHLINK	MVS LOAD into ECSA from link-list	Connection validation (AOR)
DFHDTXS	SDFHAUTH	MVS LOAD into ECSA	Connection security checking (FOR)
DFHMVRMS	SDFHLINK	MVS LOAD above the 16MB line from link-list	Resource manager EOT/EOM interface code

**Storage occupancy:** The total size of the modules that occupy storage above the 16MB line is about 41KB. For modules that are in ECSA storage, about 1.5KB are required for each logged-on FOR, and about 0.5KB for each AOR.

The modules are all eligible for inclusion in the link pack area (LPA), but only DFHDTFOR, DFHDTAM, DFHDTAOR, and possibly DFHDTCV are used sufficiently frequently to be worth considering.

**Warning:** If you place the module DFHDTINS in the LPA of an MVS system in which you use CICS/ESA release 3.2.1 in any region, or if you mistakenly place DFHDTINS in the SDFHLOAD library of a CICS/ESA 3.2.1 region, the CICS/ESA 3.2.1 regions will not be able to use any data tables that they have defined. Messages DFHFC0403 or DFHFC0412 are issued to warn you that this situation has occurred. If you wish to avoid any such problems, you should install in all of your CICS/ESA 3.2.1 regions the PTF that is associated with APAR PN18841.



---

## Chapter 5. Application programming

*This chapter contains General-use Programming Interface and Associated Guidance Information.*

You access a data table with the same EXEC CICS file control commands that you use with any normal CICS file. These commands can be used fully with a CICS-maintained data table and with some restrictions with a user-maintained data table. General information about using these commands is in the *CICS/ESA Application Programming Guide*; for programming information, see the *CICS/ESA Application Programming Reference*.

---

### CICS-maintained data table

CICS handles a CICS-maintained data table and its source data set as a single entity. After the data table has been loaded, CICS automatically keeps the contents of the data table and the source data set consistent; any changes that an application makes to the file are reflected in both.

All file control commands and options can be used and the use of a data table is transparent to the application programmer. The following information is provided to allow you to get the maximum benefits from your data tables.

Some commands are performed by access only to the data table (via cross-memory services for shared files), some by access only to the source data set (via function shipping for shared files), and some by access to both.

The following commands usually access only the data table:

- READ commands without the UPDATE or RBA options
- STARTBR, RESETBR, READNEXT, and READPREV commands without the RBA option

The following commands access only the source data set:

- READ commands with the UPDATE or RBA options
- STARTBR, RESETBR, READNEXT, and READPREV commands with the RBA option
- ENDBR command for a browse sequence that has accessed the source data set

The following commands might access both the data table and the source data set:

- READ and browse commands (which would usually access only the data table) that find a gap in the sequence of records in the data table. This gap might indicate that one or more records are missing from the data table because:
  - Records have been suppressed by a user exit
  - The maximum number of records has been reached
  - Insufficient virtual storage is available for the data table
  - Some abnormal event has occurred
- READ, READNEXT, and READPREV commands for records that are currently being processed by a WRITE, REWRITE, or DELETE command. These

commands need to first access the data table to determine that this situation exists.

- WRITE, REWRITE, and DELETE commands. These commands are always executed in the FOR, where they first update the source data set. If this is successful, a corresponding change to the data table is attempted, using local SDT services in the FOR. In the case of a WRITE command, the addition of the record to the data table might be rejected by the XDTAD user exit or might fail because the data table is full or insufficient virtual storage is available.

## Using a CICS-maintained data table during loading

It is possible to use a CICS-maintained data table while it is being loaded. If the required record has already been loaded, processing of the request is handled in the normal way. If the record has not yet been loaded, the following is done:

- For a READ command, the record is read from the source data set and returned to the application program. It is added to the data table when the normal loading sequence reaches it.
- For a WRITE command, the record is added to the source data set and the data table (if not suppressed by the user exit XDTAD).
- For a REWRITE or DELETE command, the change is applied to the source data set. This change will then be reflected in the data table by the normal loading process.

---

## User-maintained data table

CICS handles a user-maintained data table and its source data set as separate entities. When loading is complete, all file control commands that access the filename are performed only on the data table.

There are some restrictions on which commands and options can be used. There are also some exceptional conditions that are unique to user-maintained data tables. These restrictions and conditions are described below.

The following commands are not supported; they return the INVREQ condition and a value of 44 in the EIBRESP2 field:

- Commands with the RBA option
- WRITE commands with the MASSINSERT option
- DELETE commands with the GENERIC option
- READ commands with the UPDATE option plus either the GENERIC or GTEQ options

The following commands are supported (via cross-memory services for remote accesses):

- READ commands with neither the RBA option nor the UPDATE option. If the record does not exist in the data table, the NOTFND condition is returned.
- STARTBR, RESETBR, READNEXT, and READPREV commands without the RBA option.
- ENDBR commands.

The following commands are supported (via function shipping for remote requests):

- WRITE commands without the RBA or MASSINSERT options. The record is added to the data table (if not suppressed by the XDTAD user exit).

The NOSPACE condition is returned if:

- There is not enough virtual storage to add the record to the data table.
- The data table already contains the maximum number of records that is specified in the file definition.

The SUPPRESSED condition is returned if the user exit XDTAD suppresses the addition of the record to the data table.

- REWRITE commands without the RBA option. The record is updated in the data table. The NOSPACE condition is returned if there is insufficient virtual storage for the updated record.
- DELETE commands without the GENERIC or RBA options. The record is deleted from the data table. The NOTFND condition is returned if the record does not exist in the data table. The NOSPACE condition is returned if the data table is recoverable and there is insufficient virtual storage for the information that CICS writes about the deleted record.

## Using a user-maintained data table during loading

A user-maintained data table can be accessed only by the FOR during loading. All remote requests are function shipped to the FOR, which processes them in the same way as for a local request, as described below.

While a user-maintained data table is being loaded, you can use only non-update read requests with precise keys. If the record has already been loaded, processing of the request is handled in the normal way. If the record has not yet been loaded, the record is read from the source data set and submitted to the user exit XDTRD (if activated):

- If it is not suppressed by XDTRD, the record is added to the data table and also returned to the application program.
- If it is suppressed by XDTRD, the NOTFND condition is returned.

The LOADING condition is returned for other requests that would have been valid had loading been complete.

---

## Use of cross-memory services

Cross-memory services are used to satisfy an application programming command when all of the following conditions have been met:

- CICS must retrieve the SYSID of the target system from the file's resource definition in the AOR. This condition is met when the application programming command either specifies no explicit SYSID or specifies a SYSID which is that of the AOR itself, and the SYSID given in the file resource definition is that of the FOR.

Within a single browse sequence, an application should not change between specifying an explicit SYSID and not specifying one, as this is likely to lead to unpredictable results.

- The serving system has logged on. That is, it has registered itself as a shared data table owner.

- The requesting system has connected to the server for the files specified on the application programming command.
- The file supports the requested function.

**Note:** Function shipping of a request might result in “daisy chaining”; that is, the request passes through one or more intermediate CICS nodes between the region issuing the request (an AOR) and the region owning the resource (the FOR). In such cases, use of shared data tables cross-memory services is limited to the final link (from the last intermediate system to the FOR).

## Connection

Commands cannot use cross-memory services until the SDT connection is made between the AOR and the remote data table. Also, if a browse sequence starts before the connection is made, all subsequent requests in the sequence use function shipping services. This is likely to occur if the connection cannot be established at the STARTBR command because the data table is not open, and the command causes the data table to be implicitly opened. The connection will then be made on the next new request to the data table, but the original browse sequence continues to use function shipping services.

## Disconnection

When a connection has been made, it remains in force until either the AOR deletes its remote file definition or the FOR closes or disables the file. The effects of close or disable are described below.

- If the FOR closes the file (with or without the FORCE option), disconnection is scheduled at the next non-update request that is issued for the file (that is, the next request to attempt to use cross-memory services to access the data table).

The disconnection takes place as soon as all outstanding browse sequences (if any) against the file have terminated. Each browse sequence terminates either at the next browse request (and the transaction is abended with code AFCH unless the request is an ENDBR command) or when the transaction terminates.

After the disconnection is scheduled, all requests (except any outstanding browse requests, as described above) are function shipped until a connection is re-established.

- If the FOR disables the file without the FORCE option, disconnection is scheduled at the next non-update READ or STARTBR command issued for the file, unless the FOR re-enables the file before then.

If scheduled, disconnection takes place as soon as all outstanding browse sequences (if any) against the file have ended. Such browse sequences continue normally; they are unaffected by the disabling unless a browse of the source data set is started in the FOR in order to satisfy a request in the browse sequence (see “Disabling a data table” on page 31).

- If the FOR disables the file with the FORCE option, the effect is the same as when a file is closed except that, if the FOR re-enables the file before the AOR issues the next non-update request for the file, the disabling is not observed by the AOR and so disconnection is not scheduled.



---

## Differences between function-shipping and cross-memory services

You should be aware of the following differences between the way requests are handled, depending on whether function-shipping or cross-memory services are used to access the data table.

### Closing a data table

When function shipping is used for a browse sequence of a remote file, the file cannot be closed (except by using the FORCE option) until after the browse sequence ends.

When cross-memory services are used, it is possible for the file to be closed during the browse sequence. In this case, the transaction is ended with abend code AFCH at the next request for that file. If your applications or operational procedures rely on the quiescing of browse activity either when closing a file or at the normal shutdown of an FOR, you should review them before using a shared data table for the file.

### Disabling a data table

When function shipping is used for a browse sequence of a remote file, the browse sequence, once started, can continue normally even if the file is then disabled (unless the FORCE option is used).

When cross-memory services are used, the effect is the same unless, during the browse sequence, it is necessary to function ship a STARTBR command to the FOR. This can happen if, for example, a gap in a CICS-maintained data table makes it necessary to browse the VSAM source data set to retrieve records. The function-shipped STARTBR command fails if the file is then disabled by a request that was issued by the FOR after the browse sequence started in the AOR. In this case, the browse sequence is unable to continue normally, so the transaction in the AOR is abended with code AFCH.

If the FORCE option is used with the disable request, all function-shipped browse requests are always terminated. If the file is re-enabled, it is possible for browse requests that use cross-memory services to continue unaffected (see “Disconnection” on page 30).

### User exits

For function-shipped requests, the exec interface user exits XEIN and XEIOU, and the file control user exits XFCREQ and XFCREQC are invoked in both the AOR and FOR.

For cross-memory requests, these user exits are invoked only in the AOR.

### Security checking

For function-shipped requests, security checking in the FOR is invoked for the first request that refers to a given file in each unit of work. Thus transaction-level security checks can be performed in the FOR.

For cross-memory requests, security checking is invoked only at CONNECT time. Thus transaction-level security checks cannot be performed in the FOR.

## Read request failure

If a read request via function shipping fails, the input area is unchanged.

If a read request via cross-memory services fails, there is a chance that the input area will be altered although no record was retrieved. You should not therefore rely on the input area being unchanged, although you can be sure that the key will not have been changed.

## EXEC interface block

You might notice that read requests via cross-memory services return a value in the EIBRESP2 field. However, function-shipped requests do not, so your applications should not be dependent on this field being set by read requests.

## Key length

For function-shipped requests, you must specify the correct key length either in the remote-file definition in the AOR or explicitly on the file request (to match the key length in the VSAM definition in the FOR). If you do not, the INVREQ condition is returned for any request that accesses the file. This applies to any file, not just one that is defined as a data table.

For cross-memory requests, the key length in the AOR is not used; requests can complete successfully even if the key length is not specified in the AOR, or if the key length specified in the AOR does not match that in the FOR. However, your applications should not depend on this because some of the requests might be function shipped.

---

## Differences between SDT services and VSAM

Because SDT services replace VSAM for many data table requests, you should be aware of the following differences in the way that certain requests are implemented.

### Read while updating (different transactions)

In the case of a READ command for a data table record following a READ UPDATE issued for that record by *another* transaction and preceding the associated update request, when SDT services are used, the READ command is processed immediately.

When VSAM is used, the READ command waits until the update request is complete.

### Read while updating (same transaction)

In the case of a READ command for a data table record following a READ UPDATE issued for that record by *the same* transaction and preceding the associated update request, when SDT services are used, the READ command is processed immediately.

When VSAM is used, the transaction incurs a deadlock abend AFCG.

### **Delete during browse**

When SDT services are used for a STARTBR or RESETBR command for a data table record, it is possible for the record to be deleted before the associated READNEXT or READPREV command is issued. When VSAM is used, the record cannot be deleted before the associated READNEXT or READPREV command is issued.

Thus, when SDT services are used, if a STARTBR or RESETBR command is issued with a key other than the special 'last record' key, X'FF....', and the record selected is deleted before the READNEXT command, the READNEXT command reads the succeeding record.

If there is no succeeding record, the ENDFILE condition is returned. If the EQUAL option was used on the STARTBR or RESETBR, the key of the record that is read might not match the key specified.

If a STARTBR or RESETBR command is issued with the special 'last record' key, and the selected record is deleted before the READPREV command, the READPREV command reads the preceding record, or returns the ENDFILE condition if there is none.

#  
#  
#  
#

### **Write during browse**

When SDT services are used, if a browse reads to the end of a file, raising the ENDFILE condition, and a new record is then inserted beyond the end of the file, a subsequent READNEXT will be able to read the new record.

#  
#

When VSAM is used, the subsequent READNEXT may not be able to find the new record, but will instead report the ENDFILE condition again.

#  
#  
#  
#  
#  
#

### **Delete while updating (same transaction)**

When SDT services are used for a DELETE command which specifies a RIDFLD for a data table record after a READ UPDATE has been issued for that record by the same transaction and before the associated update request, the DELETE command is processed successfully, and the associated update request receives a NOTFND condition.



---

## Chapter 6. Resource definition

You define a data table in the same way as a CICS file except that you need to specify in addition:

- Which type of data table is to be used
- The maximum number of records that can be held in the data table

**Note:** The VSAM KSDS definition supplies the maximum record length and the key length.

You can define a file as a data table by one of the following methods:

- CEDA DEFINE FILE command
- DFHFCT macro

Also, you can use:

- The EXEC CICS SET FILE or CEMT SET FILE command to change the data table attributes of an existing file
- The EXEC CICS INQUIRE FILE or CEMT INQUIRE FILE command to check the data table attributes of an existing file

The recommended method of defining a file is the CEDA DEFINE FILE command. This is described below, together with the SET FILE and INQUIRE FILE commands. For information about the DFHFCT macro, see the *CICS/ESA Resource Definition Guide*.

---

### CEDA DEFINE FILE command

Figure 6 on page 36 shows the CEDA panel for defining a file, including the data table parameters.

<b>File</b>	==>	.....	
<b>Group</b>	==>	.....	
<b>DEscription</b>	==>	.....	
<b>VSAM PARAMETERS</b>			
<b>DSName</b>	==>	.....	
<b>Password</b>	==>	.....	PASSWORD NOT SPECIFIED
<b>Lsrpoolid</b>	==>	1	1-8   None
<b>DSNSharing</b>	==>	Allreqs	Allreqs   Modifyreqs
<b>STRings</b>	==>	001	1 - 255
<b>Nsrgroup</b>	==>	.....	
<b>REMOTE ATTRIBUTES</b>			
<b>REMOTESystem</b>	==>	....	
<b>REMOTENAME</b>	==>	.....	
<b>RECORDSize</b>	==>	.....	1 - 32767
<b>Keylength</b>	==>	...	1 - 255
<b>INITIAL STATUS</b>			
<b>STATus</b>	==>	Enabled	Enabled   Disabled   Unenabled
<b>Opentime</b>	==>	Firstref	Firstref   Startup
<b>DISposition</b>	==>	Share	Share   Old
<b>BUFFERS</b>			
<b>DATABuffers</b>	==>		2 - 32767
<b>INDEXbuffers</b>	==>		1 - 32767
<b>DATATABLE PARAMETERS</b>			
<b>Table</b>	==>	No	No   Cics   User
<b>Maxnumrecs</b>	==>	.....	16 - 16777215
<b>DATA FORMAT</b>			
<b>RECORDFormat</b>	==>	V	V   F
<b>OPERATIONS</b>			
<b>Add</b>	==>	No	No   Yes
<b>BRowse</b>	==>	No	No   Yes
<b>DElete</b>	==>	No	No   Yes
<b>READ</b>	==>	Yes	Yes   No
<b>Update</b>	==>	No	No   Yes
<b>AUTO JOURNALING</b>			
<b>JOurnal</b>	==>	No	No   1 - 99
<b>JNLRead</b>	==>	None	None   Updateonly   Readonly   All
<b>JNLSYNRead</b>	==>	No	No   Yes
<b>JNLUpdate</b>	==>	No	No   Yes
<b>JNLAdd</b>	==>	None	None   Before   After   All
<b>JNLSYNWrite</b>	==>	Yes	Yes   No
<b>RECOVERY PARAMETERS</b>			
<b>RECOVery</b>	==>	None	None   Backoutonly   All
<b>Fwdrecovlog</b>	==>	No	No   1-99
<b>BAckuptype</b>	==>	Static	Static   Dynamic
<b>SECURITY</b>			
<b>RESsecnum</b>	==>	00	0-24   Public

Figure 6. CEDA DEFINE FILE panel

Full details of how to use the CEDA DEFINE FILE command to define files are given in the *CICS/ESA Resource Definition Guide*. Only the parameters that relate to data tables are described in this chapter.

**TABLE({NO|CICS|USER})**

Specify **TABLE(CICS)** to define the file as a CICS-maintained data table.

Specify **TABLE(USER)** to define the file as a user-maintained data table.

If you do not specify the **TABLE** parameter, or specify **TABLE(NO)**, the file is not defined as a data table.

**MAXNUMRECS(value)**

Specify the maximum number of records that can be contained in the data table, in the range 16 through 16777215.

**FILE(name)**

Specify the name of the file.

For a CICS-maintained data table, this name is used to refer to both the data table and the source data set, which are treated as a single entity by CICS.

For a user-maintained data table, this name is used to refer to only the data table.

**DSNAME(name)**

Specify the name of the VSAM KSDS that is to be used as the source data set.

**LSRPOOLID(number|1)**

Specify the number of the VSAM local shared resource (LSR) pool that is to be used by the data table. CICS uses the LSR integrity function to prevent concurrent reading and updating of the same record by multiple users, so you must specify an LSRPOOL number, in the range 1 through 8. The default is **LSRPOOLID(1)**.

**OPENTIME({FIRSTREF|STARTUP})**

Specify when the file is to be opened, either on first reference or immediately after startup by the automatically-initiated transaction CSFU.

**OPENTIME(FIRSTREF)** is assumed by default.

Remember that the data table is loaded when the file is opened, so, if you are using the user exit XDTRD, make sure that the user exit is activated before the file is opened (see “Activating user exits” on page 60).

**RECORDFORMAT({V|F})**

Specify the format of the records in the file—either **RECORDFORMAT(V)** for variable-length records or **RECORDFORMAT(F)** for fixed-length records.

**RECORDFORMAT(V)** is assumed by default. A user-maintained data table must have variable-length records.

**ADD(NO|YES), BROWSE(NO|YES), DELETE(NO|YES), READ(YES|NO), and UPDATE(NO|YES)**

Specify which of these file operations can be requested for the data table.

**RECOVERY({NONE|BACKOUTONLY|ALL})**

Specify the type of recovery support that is required for the data table. The default is **RECOVERY(NONE)**.

For a user-maintained data table, only dynamic transaction backout is supported by CICS, so **RECOVERY(BACKOUTONLY)** and **RECOVERY(ALL)** have the same meaning.

For a CICS-maintained data table, the **RECOVERY** parameter applies to the source data set; it must be consistent with any other file definition for the same data set.

## Example of a CICS-maintained data table definition

The following example shows the definition of a CICS-maintained data table. Only the relevant parameters are shown.

```

File          ==> APPLE
Group         ==> FRUIT
Description   ==> .....
VSAM PARAMETERS
DSName       ==> CIC01.CICOWN.APPLES
Password     ==> .....          PASSWORD NOT SPECIFIED
Lsrpoolid   ==> 2              1-8 | None
DSNSharing   ==> Allreqs      Allreqs | Modifyreqs
STRings      ==> 005          1 - 255
Nsrgr       ==> .....
INITIAL STATUS
STATUS       ==> Enabled      Enabled | Disabled | Unenabled
Opentime     ==> STARTUP      Firstref | Startup
DISposition  ==> Share       Share | Old
DATATABLE PARAMETERS
Table        ==> CICS         No | Cics | User
Maxnumrecs   ==> 1000000     16 - 16777215
DATA FORMAT
RECORDFormat ==> F           V | F
OPERATIONS
Add          ==> YES         No | Yes
BRowse      ==> No          No | Yes
DElete      ==> YES         No | Yes
REAd        ==> YES         Yes | No
Update      ==> YES         No | Yes
RECOVERY PARAMETERS
RECOVery    ==> ALL         None | Backoutonly | All
Fwdrecovlog ==> 10          No | 1-99
BACKuptype  ==> STAtic      STAtic | DYNamic

```



## Example of a user-maintained data table definition

The following example shows the definition of a user-maintained data table. Only the relevant parameters are shown.

```
File          ==> COURGETT
Group         ==> VEGS
Description   ==> .....
VSAM PARAMETERS
DSName       ==> CIC02.CICOWN.COURGETT
Password     ==> .....          PASSWORD NOT SPECIFIED
Lsrpoolid    ==> 5              1-8 | None
DSNSharing   ==> Allreqs       Allreqs | Modifyreqs
STRings      ==> 005          1 - 255
Nsrgroup     ==> .....
INITIAL STATUS
STatus       ==> Enabled       Enabled | Disabled | Unenabled
Opentime     ==> FIRSTREF      Firstref | Startup
DISposition  ==> Share        Share | Old
DATATABLE PARAMETERS
Table        ==> USER         No | Cics | User
Maxnumrecs   ==> 200000       16 - 16777215
DATA FORMAT
RECORDFormat ==> V            V | F
OPERATIONS
Add          ==> YES           No | Yes
BRowse      ==> YES           No | Yes
DElete      ==> No            No | Yes
REAd        ==> YES           Yes | No
Update      ==> YES           No | Yes
RECOVERY PARAMETERS
RECOvery    ==> BACKOUTONLY    None | Backoutonly | All
Fwdrecovlog ==> No             No | 1-99
Backuptype  ==> STAtic         STAtic | DYNamic
```

---

## EXEC CICS commands

*This section contains General-use Programming Interface and Associated Guidance Information.*

You can use the EXEC CICS SET FILE command to change the definition of an existing file, and the EXEC CICS INQUIRE FILE command to check the definition of an existing file. For programming information, including details of how to use these commands and the parameters described here, see the *CICS/ESA System Programming Reference* manual. The parameters that are relevant to data tables are described below.

## SET FILE

The following parameters are relevant to data tables; you can use them only when the file is closed and disabled. You can specify a data table attribute of a file in a CICS-value data area (cvda):

### TABLE(cvda)

Specify a cvda value of **CICSTABLE** to define the file as a CICS-maintained data table.

Specify a cvda value of **USERTABLE** to define the file as a user-maintained data table.

Specify a cvda value of **NOTTABLE** to indicate that the file is not a data table.

**MAXNUMRECS(value)**

Specify the maximum number of records that can be contained in the data table, in the range 16 through 16777 215.

## INQUIRE FILE

The following parameters are relevant to data tables. You can request that each data table attribute of a file is returned in a CICS-value data area (cvda) by specifying:

**TABLE(cvda)**

If the value **CICSTABLE** is returned, the file has been defined as a CICS-maintained data table.

If the value **USERTABLE** is returned, the file has been defined as a user-maintained data table.

If the value **NOTTABLE** is returned, the file is not currently defined as a data table.

If the value **NOTAPPLIC** is returned, the option is not applicable because the file is a remote file.

**MAXNUMRECS(cvda)**

The value returned indicates the maximum number of records that can be contained in the data table.

---

## CEMT commands

You can use the CEMT SET FILE command to change the definition of an existing file, and the CEMT INQUIRE FILE command to check the definition of an existing file. Full details of how to use these commands, including the parameters described here, are given in *CICS/ESA CICS-Supplied Transactions*. The parameters that are relevant to data tables are described below.

## SET FILE

The following parameters are relevant to data tables; you can use them only when the file is closed and disabled.

**{CICSTABLE|USERTABLE|NOTTABLE}**

Specify **CICSTABLE** to define the file as a CICS-maintained data table.

Specify **USERTABLE** to define the file as a user-maintained data table.

Specify **NOTTABLE** to indicate that the file is not a data table.

**MAXNUMRECS(value)**

Specify the maximum number of records that can be contained in the data table, in the range 16 through 16777 215.

## INQUIRE FILE

The following parameters are relevant to data tables.

### Data table

If the value **CICSTABLE** is returned, the file has been defined as a CICS-maintained data table.

If the value **USERTABLE** is returned, the file has been defined as a user-maintained data table.

If the value **NOTTABLE** is returned, the file is not currently defined as a data table.

### MAXNUMRECS(value)

The value returned indicates the maximum number of records that can be contained in the data table. The value is zero if no value has been set previously.



---

## Chapter 7. Customization

*This chapter contains Product-sensitive Programming Interface and Associated Guidance Information.*

This chapter describes the three global user exit points that are included in data table services. You can supply one or more assembler-language programs to be executed at each of these points in order to extend or modify the function provided by CICS.

The three global user exits are:

- **XDTRD**, which is invoked for each record that is read from the source data set (normally when the file is being loaded). You can then select whether to load the record into the data table or not. For a user-maintained data table, you can also modify the record.
- **XDAD**, which is invoked for each record that is added to the source data set. You can then select whether to add the record to the data table or not.
- **XDCLC**, which is invoked when loading of the data table is complete, whether successful or not.

This chapter describes how to write exit programs for these user exits. Samples of exit programs are shown in Appendix A, “Sample user exit programs” on page 93.

The method of defining and activating the user exits is described in “Activating user exits” on page 60. Programming information about global user exits and how to use them is given in the *CICS/ESA Customization Guide*.

**Exec interface and file control user exits:** The exec interface user exits XEIIIN and XEIOUT and the file control user exits XFCREQ and XFCREQC are not invoked in the file-owning region if a request to access a data table is satisfied by cross-memory services.

---

### Communicating between CICS and exit programs

A parameter list is used to pass information between CICS and the data table exit programs. In the CICS410.SDFHMAC library, CICS/ESA 4.1 supplies a copybook, named DFHXDTDS, which contains a DSECT to define this parameter list. You should include a COPY DFHXDTDS statement in each of your exit programs. The DSECT is shown in Figure 7 on page 44.

The field names used in this DSECT are referenced in the user exit descriptions that follow the figure.

```

*****
*
*   Data Table Parameter List for User Exits XDTRD, XDTAD and XDTLC.
*
*   Some of the parameters are only used by one or two of the exits.
*   This is indicated in the comments for those parameters.
*   The comments also indicate whether the field is used for input
*   (In), output (Out), or both (In/Out).
*
*   NOTE that this definition could be used by exit programs running
*   both on CICS regions which have shared data tables support
*   (SDT) installed and on ones which do not, providing the UEPDTSDT
*   flag is used to test whether SDT is installed, and that the
*   parameters which are unique to SDT are only used when it is set.
*
*****
DT_UE_PLIST_DSECT  DSECT ,
DT_UE_PLIST        DS  0XL84          Data Table User Exits      X
                                     Parameter List
UEPDTNAM           DS  CL8            Data table name (In)
UEPDTFLG           DS  0CL1          Flags (In):
                                     DS  BL1
UEPDTSDT           EQU X'80'        Exit invoked by SDT support
*-----*
*           The remaining flags are only available to exits which
*           have been invoked by shared data tables support
*-----*
UEPDTTCMT          EQU X'40'        Table is CICS-maintained
UEPDTOPT           EQU X'20'        Exit invoked by table loader, X
                                     so optimization by skipping X
                                     may be requested - XDTRD only
*
*           EQU X'1F'
UEPDTORC           DS  AL1          Data table load return code - X
                                     XDTLC only, values below (In)
                                     DS  BL2
                                     Reserved

```

Figure 7 (Part 1 of 2). Data table user exit parameter list

UEPDTRA	DS	A	Data record address - XDTRD and XDTAD only (In)	X
UEPDTRBL	DS	F	Data buffer length - XDTRD and XDTAD only (In)	X
UEPDTRL	DS	F	Data table record length - XDTRD and XDTAD only, XDTRD can return new length in here if it amends record (only allowed for UMT) (In/Out)	X
UEPDTKA	DS	A	Key address - XDTRD and XDTAD only (In)	X
UEPDTKL	DS	F	Key length - XDTRD and XDTAD only (In)	X
*-----*				
*           The following fields are only available to exits which           *				
*           have been invoked by shared data tables support           *				
*-----*				
UEPDTDSL	DS	F	Length of data set name (In)	
UEPDTDSN	DS	CL44	Source data set name (In)	
UEPDTSKA	DS	A	Address of skip-key area: exit should return a key of length UEPDTKL in this area if it has requested optimization of load by skipping - XDTRD only (In)	X
*-----*				
*   Values for UEPDTORC (supplied to XD TLC exit only)           *				
*-----*				
UEPDTLCS	EQU	0	load completed successfully	
UEPDTLFL	EQU	128	load failed	

Figure 7 (Part 2 of 2). Data table user exit parameter list

The user exits should set a return code in register 15. The return code values are supplied by the DFHUEEXIT macro. The valid values for each user exit are given in the following descriptions.

If you want your exit programs to work for both versions of data tables support<sup>1</sup>, you can check UEPDTFLG to find out which version of data tables support invoked the exit program. For SDT, this flag byte also indicates which type of data table is being used and whether the exit program is being invoked during loading.

The exit program should use either the filename (field UEPDTNAM) or the name of the source data set (see fields UEPDTDSN and UEPDTDSL) to determine whether this is a file for which any action is to be taken.

You can enable several exit programs at the same exit point, each of which, for example, takes action for a particular file or data set.

<sup>1</sup> The two versions are:

- Basic data tables support, supplied as a feature for CICS/MVS releases 2.1.1 and 2.1.2, and as part of the base CICS product for CICS/ESA releases 3.1.1, 3.2.1, and 3.3
- Shared data tables (SDT) support, supplied as a feature for CICS/ESA release 3.3, and as part of the base CICS product for CICS/ESA 4.1.

---

## XDTRD user exit

The XDTRD user exit is invoked just before CICS attempts to add to the data table a record that has been retrieved from the source data set.

This normally occurs when the loading process retrieves a record during the sequential copying of the source data set. However, it can also occur when an application retrieves a record that is not in the data table and:

- For a user-maintained data table, loading is still in progress, or
- For a CICS-maintained data table, loading terminated before the end of the source data set was reached (because, for example, the data table was full).

The record retrieved from the source data set is passed as a parameter to the user exit program—see fields UEPDTRA and UEPDTRL. This program can choose (depending, for example, on the key value—see fields UEPDTKA and UEPDTKL) whether to include the record in the data table or not.

Alternatively, the exit program can request that all subsequent records up to a specified key are skipped—see field UEPDTSKA; these records are not passed to the exit program. This facility is available only during loading. You can specify the key as a complete key, or you can specify just the leading characters by padding the skip-key area with binary zeros.

The action required is indicated by setting the return code. Depending on the return code value, the following action is taken by CICS:

Return code	Action
UERCDTAC	Include the record in the data table. This is the default if the exit is not activated.
UERCDTRJ	Do not include the record in the data table.
UERCDTOP	Skip over this record and the following records until a key is found that is equal to or greater than the key specified in the skip-key area.

For a user-maintained data table, the program can also modify the data in the record to reduce the storage needed for the data table. Application programs that use the data table must be aware of any changes made to the record format by the exit program. If the record length is changed, the exit program must set the new length in the parameter list—see field UEPDTRL. The new length must not exceed the data buffer length—see field UEPDTRBL.

---

## XDTAD user exit

The XDTAD user exit is invoked when a write request is issued to a data table.

- For a user-maintained data table, the user exit is invoked once—before the record is added to the data table.



- For a CICS-maintained data table, the user exit is invoked twice—before the record is added to the source data set and then again before the record is added to the data table.

The record written by the application is passed as a parameter to the user exit program—see fields UEPDTRA and UEPDTRL. This program can choose (depending on the key value, for example—see fields UEPDTKA and UEPDTKL) whether to include the record in the data table or not. This decision is indicated by setting the return code.

Depending on the return code value, the following action is taken by CICS:

<i>Table 8. Return codes for XDTAD user exit. A value of UERCPURG should be returned if the exit program has received a PURGED response to a call that it has issued.</i>	
<b>Return code</b>	<b>Action</b>
UERC DTAC	Add the record to the data table. This is the default if the exit is not activated.
UERC DTRJ	Do not add the record to the data table.

The XDTAD exit must not modify the data in the record. If you used XDTRD to truncate the data records when the user-maintained data table was loaded, you must code your application so that it only tries to write records of the correct format for the data table.

---

## XDTLC user exit

The XDTLC user exit is invoked at the completion of data table loading—whether successful or not. **The user exit is not invoked if the data table is closed for any reason before loading is complete.**

The exit program is informed if the loading did not complete successfully—see field UEPDTORC. This could occur, for example, if the maximum number of records was reached or there was insufficient virtual storage. In this case, the exit program can request that the file is closed immediately, by setting the return code.

Depending on the return code value, the following action is taken by CICS:

<i>Table 9. Return codes for XDTLC user exit. A value of UERCPURG should be returned if the exit program has received a PURGED response to a call that it has issued.</i>	
<b>Return code</b>	<b>Action</b>
UERC DTOK	No action; the file remains open. This is the default if the exit is not activated.
UERC DTCL	Close the file.



---

## Chapter 8. Operations

This chapter describes the operational aspects of using data tables, such as:

- Opening and closing data tables
- MVS job control
- Interpreting data table statistics
- Activating user exits

---

### Opening a data table

A data table must be opened before it can be used by an application. This is done in the same way as for any CICS file, by one of the following methods:

- Automatically, by the CICS-supplied transaction CSFU, at the end of CICS startup, if the data table is defined with `OPENTIME(STARTUP)` or `FILSTAT=OPENED`
- Explicitly, by a `CEMT` or `EXEC` CICS request issued by the user
- Implicitly, on first reference to the data table, if the data table is defined with `OPENTIME(FIRSTREF)` or `FILSTAT=CLOSED`. The first remote access to a closed data table implicitly opens it.

All the rules and options for opening a CICS file also apply to a file that is defined as a data table. In addition, the loading of the data table is initiated.

For a large data table, loading could take a significant time. The application programming commands that can be used with a user-maintained data table, and the performance gains that can be achieved with a CICS-maintained data table are limited until loading is completed (see Chapter 5, “Application programming” on page 27).

The following steps are done during the opening of the file:

1. The access method control block (ACB) for the VSAM source data set is opened under a separate MVS task control block (TCB). This step is the same as for any CICS file.
2. For the first data table used by a region, CICS:
  - Creates MVS storage pools for use by SDT
  - Creates an MVS/ESA data space for use by this region’s data tables
  - Attempts a LOGON operation as a server.
3. A special CICS transaction, `CSSY`, is attached to load the data table into the data space.
4. The transaction that issued the request to open the data table can now continue processing.
5. CICS issues a message `DFHFC0940` to indicate that loading has started. The message is sent to the `CSFL` transient data queue.
6. The transaction that loads the data table reads the source data set sequentially. Under the optional control of the user exit `XDTRD`, the transaction copies the records into the data space. It also constructs an index in address-space storage to facilitate subsequent access to the records.

7. CICS issues a message to indicate the result of the loading. The message number is:

- If loading is successful—DFHFC0941
- If loading fails—DFHFC0942, DFHFC0943, DFHFC0945, DFHFC0946, DFHFC0947, or DFHFC0948.

The message is sent to the CSFL transient data queue. Also, if loading fails, the message is sent to the console. For descriptions of these messages, see the *CICS/ESA Messages and Codes* manual.

8. When loading is complete (whether successful or not), the user exit XDTLC is invoked if it is active. If the loading was not completed successfully, the exit program can request that the data table is closed.

9. For a user-maintained data table, the ACB for the source data set is closed when loading is complete. The data set is deallocated if it was originally dynamically allocated and no other ACBs are open for it.

**Note:** During an emergency restart, any file that requires backout action is reopened. However, if the file is defined as a data table, loading is not initiated at that time; instead, it is initiated by the CSFU transaction at the end of the emergency restart. This gives an opportunity for any user exits that control the copying of records to the data table during loading to be activated at any stage of PLTPI processing.

---

## Closing a data table

A data table is closed in the same way as for any CICS file, by one of the following methods:

- Explicitly, by a CEMT or EXEC CICS request issued by the user.
- Implicitly, when CICS is shutdown normally.

All the rules and options for closing a CICS file also apply to a data table. In particular:

- The rules about the quiescing of current users of the file apply (except that the file can be closed even when a transaction that is running in an AOR is in the middle of a browse sequence).
- For a user-maintained data table, if the data table is defined as recoverable, all units of work that have changed the data table must complete before the data table can be closed.

The data space storage that is used for the data table records, and the address-space storage that is used for the associated table-entry and index storage is freed as part of the close operation. If a file is reopened after it has been closed, the processing is the same as if the file had not been previously opened.

---

## MVS job control

Before using shared data tables, you might need to change some of your JCL statements, modify your operational procedures, or increase the value of the MAXUSER MVS initialization parameter. This is because MVS/ESA does not allow more than one step of a job to act as a shared data table server. If a second job step attempts to act as a shared data table server, CICS issues message DFHFC0405. Also, as job steps following the server step would also be unable to use cross-memory services with MRO, it is recommended that none of the job steps following the server step are another execution of CICS.

If a job that includes a shared data tables server step ends before all requester job steps that connected to this server have ended, the server address space is terminated by MVS/ESA. If the SDT server is running under the control of a batch initiator rather than as a started task, a new initiator must be started when this situation occurs.

MVS/ESA terminates the batch initiator with the message IEF355A INITIATOR TERMINATED, RESTART INITIATOR because, otherwise, for integrity reasons, MVS would have to restrict the functions that could be used by the next job that runs under that initiator, which might cause the job to fail. MVS does not allow an SDT server's ASID to be re-used until after all requester job steps that connected to the server have ended.

---

## Interpreting data table statistics

This section describes the statistics information that is produced by CICS to help you monitor the activity on your data tables.

You can use the information contained in a CICS statistics report to evaluate the benefits of using data table services. The information that is recorded for a data table is shown in Table 10.

*Table 10. Data tables statistics*

Heading	Description
Close Type	Type of data table close (only appears in the statistics collected when a data table is closed)
Read Requests	Number of attempts to read records from data table or, in the AOR statistics, the number of read requests that had to be function shipped
Recs - in Table	Number of attempts to read records from source data set because the record was not found in data table. (For a user-maintained data table, this happens only during loading.)
Adds from Reads	Number of attempts to copy records from source data set to data table during loading process (including read requests from applications during loading)
Add Requests	Number of attempts to write records to data table
Adds rejected - Exit	Number of records suppressed by XDTRD and XDTAD user exits
Adds rejected - Table Full	Number of records that were not included because the table was full
Rewrite Requests	Number of attempts to rewrite records in data table
Delete Requests	Number of attempts to delete records from data table
Highest Table Size	Peak number of records held in data table
Storage Alloc(K)	Number of KB allocated to data table

The statistics for data tables are included in the FILES section of the statistics report. Four sections of FILE information are produced:

- The “FILES - Resource Information” shows information such as the filename, source data set name, data set type (which will always be K for a data table, because the source must be a VSAM KSDS) and a DT indicator that is explained below.
- The “FILES - Requests Information” shows the statistics for accesses to the source data set. For a loaded user-maintained data table, this contains only zeros.
- The “FILES - Data Table Requests Information” shows statistics for accesses to the data table. The meanings of the column headings are given in Table 10 on page 51.
- The “FILES - Performance Information” shows the use of VSAM strings and buffers, and is only of interest for a data table in that it relates to the source data set.

A request to a data table that is owned by another region is recorded in the statistics report for both the requesting and file-owning regions.

The DT Indicator is a single character that can have the values:

- |          |   |
|----------|---|
| <b>T</b> | The statistics report contains data table information because the file has been opened as a data table.   |
| <b>R</b> | The statistics report contains information for a remote file that has accessed a data table via shared data tables cross-memory services.   |
| <b>S</b> | The statistics report contains data table information for the accesses made by this file to an associated data table (which has the same source data set).                                  |
| <b>X</b> | The statistics report contains information for an alternate index file, and reports data table information for the update of the data table associated with a file in the same upgrade set. |

The field is blank if data table statistics are not present for the file.

The Close Type is a single character that appears in the statistics only for the closing of a data table. The possible values indicate the type of close:

- |          |   |
|----------|---|
| <b>C</b> | Close of a CICS-maintained data table.  |
| <b>P</b> | Partial close of a CICS-maintained data table. There are still other files using the data table, so the data table itself has not been closed, only the file. |
| <b>S</b> | Close of the source data set for a user-maintained data table. This happens at the end of loading of the data table.  |
| <b>U</b> | Close of a user-maintained data table.  |

A total value for the storage allocated is not included in the TOTALS line.

For information on how to obtain a statistics report, using the statistics utility program DFHSTUP, see the *CICS/ESA Operations and Utilities Guide*. For a description of the data contained in a statistics report, see the *CICS/ESA Performance Guide*.

## Sample data table statistics

Figure 8, Figure 9 on page 54, and Figure 10 on page 55 show extracts from the file statistics for a hypothetical configuration similar to that discussed in “Using statistics to select data tables” on page 18, following conversion of the files into data tables. These figures are used to discuss how to interpret the information about data tables which is provided by the CICS statistics.

Requested Statistics Report										Collection Date-Time 12/25/99-23:55:41		Last Reset 21:00:00		Applid CICFOR		Jobname SDTGSTF1			
FILES - Resource Information																			
File Name	Dataset Name Base Dataset Name (If Applicable)	Dataset Type	DT Indicator	Time Opened	Time Closed	Remote Name	Remote Sysid	Lsrpool ID											
APPLE	CIC01.CICOWN.APPLES	K	T	22:12:04	OPEN			1											
BANANA	CIC01.CICOWN.BANANAS	K	T	22:53:56	OPEN			1											
ORANGE	CIC01.CICOWN.CITRUS	K	T	20:51:25	OPEN			2											
PLUM	CIC01.CICOWN.PLUMS	K	T	21:30:10	OPEN			4											
POTATO	CIC01.CICOWN.POTATOES	K	T	20:30:10	OPEN			8											
VICTORIA	CIC01.CICOWN.PLUMS	K	S	21:56:23	OPEN			4											
*TOTALS*										1088	104988	834882	110195	5697	1893				
FILES - Requests Information																			
File Name	Get Requests	Get Upd Requests	Browse Requests	Update Requests	Add Requests	Delete Requests	VSAM Data	EXCP Index											
APPLE	1	495	760001	495	12	0	14560	1321											
BANANA	0	1803	48603	1803	951	0	8212	481											
ORANGE	1087	102677	104	107897	4709	1880	25180	1947											
PLUM	0	10	2001	0	20	10	580	3											
POTATO	0	0	24173	0	0	0	4513	2											
VICTORIA	0	3	0	0	5	3	5	1											
*TOTALS*										1088	104988	834882	110195	5697	1893				
FILES - Data Table Requests Information																			
File Name	Close Type	Read Requests	Recs in Table	Adds from Reads	Add Requests	Adds rejected - Exit	Adds rejected - Table Full	Rewrite Requests	Delete Requests	Highest Table Size	Storage Alloc(K)								
APPLE		1105241	1	760000	12	0	0	495	0	760012	95104								
BANANA		652195	0	48602	951	2	0	1803	0	49551	11200								
ORANGE		1473	1191	0	4709	0	0	107897	1880	604167	266658								
PLUM		227	1	2000	20	0	0	0	10	2025	256								
POTATO		9670	24173	24165	0	0	24165	0	0	1000	1760								
VICTORIA		3063	1	0	5	0	0	0	3	2025	256								
*TOTALS*										1771869	25367	834767	5697	2	24165	110195	1893	760012	

Figure 8. CICFOR requested file statistics

Requested Statistics Report      Collection Date-Time 12/25/99-23:56:58    Last Reset 21:00:00    Applid CICAOR1    Jobname SDTGSTA1

FILES - Resource Information

File Name	Dataset Name Base Dataset Name (If Applicable)	Dataset Type	DT Indicator	Time Opened	Time Closed	Remote Name	Remote Sysid	Lsrpool ID
APPLE	REMOTE		R	CLOSED	CLOSED	APPLE	CIF1	N
BANANA	REMOTE		R	CLOSED	CLOSED	BANANA	CIF1	N
ORANGE	REMOTE		R	CLOSED	CLOSED	ORANGE	CIF1	N
POTATO	REMOTE		R	CLOSED	CLOSED	POTATO	CIF1	N
ZUCCHINI	REMOTE		R	CLOSED	CLOSED	COURGETT	CIA2	N

Requested Statistics Report      Collection Date-Time 12/25/99-23:56:58    Last Reset 21:00:00    Applid CICAOR1    Jobname SDTGSTA1

FILES - Requests Information

File Name	Get Requests	Get Upd Requests	Browse Requests	Update Requests	Add Requests	Delete Requests	VSAM EXCP Data	Requests Index
APPLE	1	520	0	520	5	0	0	0
BANANA	0	0	0	0	0	0	0	0
ORANGE	214	38735	14	37105	1311	630	0	0
POTATO	0	0	24173	0	0	0	0	0
ZUCCHINI	0	0	0	0	0	0	0	0
*TOTALS*	215	39255	24187	38625	1316	630		

Requested Statistics Report      Collection Date-Time 12/25/99-23:56:58    Last Reset 21:00:00    Applid CICAOR1    Jobname SDTGSTA1

FILES - Data Table Requests Information

File Name	Close Type	Read Requests	Recs in Table	Adds from Reads	Adds rejected - Exit	Adds rejected - Table Full	Rewrite Requests	Delete Requests	Highest Table Size	Storage Alloc(K)
APPLE		1304214	0	0	0	0	0	0	0	0
BANANA		441349	0	0	0	0	0	0	0	0
ORANGE		63384	228	0	0	0	0	0	0	0
POTATO		4835	24173	0	0	0	0	0	0	0
ZUCCHINI		97867	0	0	0	0	0	0	0	0
*TOTALS*		1911649	24401	0	0	0	0	0	0	0

Figure 9. CICAOR1 requested file statistics



FILES - Resource Information

File Name	Dataset Name Base Dataset Name (If Applicable)	Dataset Type	DT Indicator	Time Opened	Time Closed	Remote Name	Remote Sysid	Lsrpool ID
COURGETT LEMON	CIC02.CICOWN.COURGETT REMOTE	K	T R	22:35:02 CLOSED	OPEN CLOSED	ORANGE	CIF1	1 N

FILES - Requests Information

File Name	Get Requests	Get Upd Requests	Browse Requests	Update Requests	Add Requests	Delete Requests	VSAM EXCP Data	Requests Index
COURGETT LEMON	0 946	0 63942	0 299	0 70792	0 3398	0 1250	0 0	0 0
*TOTALS*	946	63942	299	70792	3398	1250		

FILES - Data Table Requests Information

File Name	Close Type	Read Requests	Recs ~ in Table	Adds from Reads	Add Requests	Adds rejected - Exit	Adds rejected - Table Full	Rewrite Requests	Delete Requests	Highest Table Size	Storage Alloc(K)
COURGETT LEMON		27656 3240872	0 1245	0 0	386590 0	0 0	0 0	27656 0	0 0	101232 0	16160
*TOTALS*		3268528	1245	0	386590	0	0	27656	0	101232	

Figure 10. CICAOR2 requested file statistics

The main changes that are seen in the statistics when a file is redefined as a data table are:

1. Data table statistics appear in both the owning and requesting regions showing the use of the data table.
2. Values appear in the owning region showing the activity of loading the data table.
3. Except during loading, the counts of Get Requests and Browse Requests in the "FILES - Requests Information" statistics are much reduced (often to zero) because such requests can now be satisfied from the data table.
4. The Recs ~ in Table figure indicates the degree to which benefit from shared data tables support has been prevented. In an AOR it shows how many retrieval requests have had to be function shipped. In an FOR it shows how many records have had to be fetched from the source data set.

You should use the statistics to get an overall feel for the behavior of your data tables, rather than attempting to explain the individual values.

The examples demonstrate a number of points about the statistics. These points are discussed in the rest of this section.

### Normal loading

Adds from Reads usually shows the number of attempts to add a record during loading<sup>2</sup>. Because the loading process involves browsing the source data set until the end of the file, the number of Browse Requests (in the “FILES - Requests Information”) equals Adds from Reads + 1 if loading completed successfully and there have been no other browses on the source data set. The statistics for APPLE in Figure 8 on page 53 illustrate this point.

In the CICFOR statistics shown, APPLE, BANANA, and PLUM were opened after the last statistics reset, but ORANGE and POTATO were opened before, so the latter do not display the load-time statistics. It is generally better to assess statistics from a time interval that has not been distorted by loading, but you should remember that the loading process incurs an overhead that has to be recovered by the number of data table accesses.

### Optimization of loading

The statistics for BANANA in Figure 8 on page 53 show an example in which only a range of key values from the middle of the source data set is required in the data table. Here, the XDTRD exit has been used to skip over any keys that are not in this range (see “XDTRD user exit” on page 46).

Adds rejected - Exit shows the number of times the exit returned a non-zero return code, and is 2 in this case: 1 for when the first record in the source data set was presented, and the exit requested the load to skip on to the first key in the desired range, and 1 for when the first key beyond this range was presented, and the exit requested the load to skip over all remaining records to the end of the source data set.

In a case like this, you would usually use the XDTAD exit to reject any records that are written with keys outside the desired range. Then, the number of Adds rejected - Exit would include the number of such records that had been written to the file.

The number of Adds from Reads contains the number of records that were loaded into the data table plus the two that were rejected. As for all the file and data table statistics, this figure shows the number of attempted, rather than the number of successful, writes.

### Loading a user-maintained data table

When the loading of a user-maintained data table completes, the source data set is closed and an unsolicited statistics record is written that reports the number of records that are browsed from the source and written to the data table (or rejected by the XDTRD exit). Therefore, these figures do not appear in any later statistics reports, such as that for COURGETT in Figure 10 on page 55. A Close Type of S identifies such statistics.

---

<sup>2</sup> If loading has completed, but the load failed to read to the end of the source data set, then the count for a CICS-maintained data table might also show attempts to add records that have been read from the source data set because they were not originally loaded.

### **Implicit open from the requesting region**

If the file is not open in the FOR when the AOR issues the first read to it, then no connection exists and the read is function shipped. This appears as one file Get Request in the AOR statistics. The implicit open and subsequent loading of the data table is triggered in the FOR.

This first read attempt from the new data table is counted in the Read Requests in the FOR data table statistics, but as the record is not found in the data table at this stage, it is added to the count of Recs  $\rightarrow$  in Table (which records the number of times a record could not be obtained from the data table). The record is fetched from VSAM, so the number of file Get Requests is incremented by 1. The statistics from CICAOR1 and CICFOR for APPLE illustrate this point.

### **Update requests**

All update requests (writes, rewrites, and deletes) are processed by the owning region, which also controls loading and other maintenance of the data table.

Because of this, the data table statistics of Adds from Reads, Add Requests, Adds rejected, Rewrite Requests, Delete Requests, Highest Table Size, and Storage Alloc are always zero on the remote requesting regions.

Updates are always reflected in both the data table and the source data set for a CICS-maintained data table, so matching numbers are often seen in the file statistics and the data table statistics for Add Requests and Delete Requests, and also for Update Requests in the file statistics compared with Rewrite Requests in the data table statistics. The statistics for APPLE and ORANGE illustrate this point.

These numbers might not match if not all records in the source data set are loaded into the data table, or if some error occurs when the source data set is updated. For example, an attempt to write a record with a duplicate key to the source data set is counted in the file Add Requests but no attempt is made to write the record to the data table, so the count of data table Add Requests is less.

In the case of a user-maintained data table, access after loading is complete is always to the data table, so the line of file statistics always contains zeros. The statistics for COURGETT in Figure 10 on page 55 illustrate this point.

### **Data table high water mark**

The Highest Table Size shows the largest number of records that was present in the data table at any one time. For APPLE, from which no records have been deleted, this is the number of records originally loaded, plus the number of data table Add Requests. For BANANA, the XDTRD user exit accepts only records that are of interest during loading, and XDTAD performs the same task when records are written; so the number is given by Adds from Reads plus Add Requests minus Adds rejected - Exit.

### **Total storage allocated**

Storage Alloc gives the number of Kilobytes that have been allocated to the data table. This includes both address space and data space storage.

## Reading and browsing

The number of data table Read Requests includes browses that are satisfied by the data table. Thus for ORANGE in Figure 9 on page 54 and LEMON in Figure 10 on page 55 the numbers of file Browse Requests are very small (and in most cases they would be zero). But the number of data table Read Requests is of a similar magnitude to the total number of Get and Browse requests that were made before conversion of the file to a data table.

## Failure to access records via the data table

The set of figures for ORANGE and LEMON show an effect that is sometimes seen when there is much update activity on a CICS-maintained data table. In this case, some of the read requests from remote regions might find that the record in the data table is being updated, so these requests are function shipped to the FOR.

For example, LEMON shows 1245 Recs  $\rightarrow$  in Table, of which 946 are Get Requests and 299 are Browse Requests. The function-shipped reads and browses attempt to access the data table in CICFOR (as shown by the 1473 Read Requests for ORANGE), by which time some of the reads can be satisfied from the data table, but the remainder use the source data set (as shown by the number under Recs  $\rightarrow$  in Table).

POTATO shows what can happen if an unsuitable choice of candidate is made. Because the data table size specified in the file definition is much less than the number of records in the source data set, only a small part of the file is loaded. However, the file is accessed remotely by an application that browses many records that lie beyond those that were loaded.

All those browse requests have to be function shipped to CICFOR (as shown by the high number of Recs  $\rightarrow$  in Table seen in Figure 9 on page 54 for that file) and then, on CICFOR, the source data set has to be accessed (as shown by the number of Browse Requests in Figure 8 on page 53). An attempt is made to add the records to the data table (see count of Adds from Reads) but as the data table is still at its maximum number of records, they have to be rejected (see Adds rejected - Table Full). Incidentally, the statistics record that contains the loading figures for the file includes 1 under Adds rejected - Table Full for the record that caused the load to terminate because the data table had reached its maximum size.

The values of Read Requests, Recs  $\rightarrow$  in Table, Browse Requests, Adds from Reads, and Adds rejected - Table Full are not all equal (as might have been expected) because some browses reach the end of the source data set (in which case there is no record to attempt to add to the data table) and also because often no attempt is made to access the data table when browsing over a range of records that is known to be missing from the data table.

Although this is an extreme example, it does illustrate the importance in certain situations of having a good understanding of the applications. A user exit should have been used to select the range of records on which most of the browsing occurs.

## Multiple files with a single source

The file VICTORIA in CICFOR is included to show that when a second file is defined with the same source data set as a file that is open as a data table (PLUM) then it can take advantage of the data table for non-update reads and browses (regardless of which file is opened first). In the Resource Information for VICTORIA, a DT Indicator of S means that the line of data table statistics shows how the data table has been used by this associated file; for example, 3063 reads or browses have been satisfied from the data table.

The same Storage Alloc and Highest Table Size statistics are reported for both PLUM and VICTORIA because the data table is associated with both files. Because of this, the data table TOTALS line does not include a value for the total storage allocated. The load-time statistics are reported only for the file that initiated the data table; that is, the file whose open caused the current instance of the data table to be built.

## Additional statistics fields

Some additional statistics about shared data tables are collected when file statistics are gathered, but they are not formatted in a statistics report. You can write a program to extract these additional statistics from the statistics record. For programming information about CICS statistics, see the *CICS/ESA Customization Guide*.

This section describes the fields that are related to shared data tables and are part of the DFHA17 statistics record but are not displayed in a statistics report.

### A17\_DT\_SIZE\_CURRENT

This is a fullword at offset 160 (X'A0'). It contains the current count of records in the data table.

### A17\_DT\_IN\_USE\_TOTAL

This is a fullword at offset 168 (X'A8'). It contains the total amount of storage (in KB) currently in use for the data table.

### A17\_DT\_ALLOC\_ENTRY

This is a fullword at offset 172 (X'AC'). It contains the amount of storage (in KB) currently allocated from the server's address space to hold table entries for this data table.

### A17\_DT\_IN\_USE\_ENTRY

This is a fullword at offset 176 (X'B0'). It contains the amount of storage (in KB) in the server's address space currently being used by table entries for this data table.

### A17\_DT\_ALLOC\_INDEX

This is a fullword at offset 180 (X'B4'). It contains the amount of storage (in KB) currently allocated from the server's address space to the index for this data table.

### A17\_DT\_IN\_USE\_INDEX

This is a fullword at offset 184 (X'B8'). It contains the amount of storage (in KB) in the server's address space currently being used by the index for this data table.

### **A17\_DT\_ALLOC\_DATA**

This is a fullword at offset 188 (X'BC'). It contains the amount of storage (in KB) currently allocated from the data space for the data portion of the records in this data table.

### **A17\_DT\_IN\_USE\_DATA**

This is a fullword at offset 192 (X'C0'). It contains the amount of storage (in KB) in the data space currently being used to hold record data for this data table.

### **A17\_DT\_REREADS**

This is a fullword at offset 196 (X'C4'). It contains a count of the number of times a read from a requesting region has retried a part of the data table section of the request processing because the data table changed in some way after the start of that section.

These fields are also displayed in the X'0B22' exit trace for a statistics call amongst the fourteen statistics fullwords, which are (in the order they appear in the trace):

Adds from Reads during load  
Adds rejected - Exit during load  
Adds rejected - Table Full during load  
Highest Table Size  
Current record count  
Storage Alloc (K) or Total storage allocated  
Total storage in-use  
Entry storage allocated  
Entry storage in-use  
Index storage allocated  
Index storage in-use  
Data storage allocated  
Data storage in-use  
Rereads

Because these are internal fields, the traced values do not always correspond exactly to those in a statistics record.

---

## **Activating user exits**

To activate the data table user exits, you need to perform the following steps:

1. Decide which user exits you want to use. A description of each user exit is included in Chapter 7, "Customization" on page 43.
2. Write the user exit programs. Examples are included in Appendix A, "Sample user exit programs" on page 93.
3. Define the user exit programs to CICS, using the CEDA DEFINE PROGRAM command (see the *CICS/ESA Resource Definition Guide*).
4. Activate the user exits, using the EXEC CICS ENABLE command (for programming information about this command, see the *CICS/ESA System Programming Reference* manual). If required, you can later deactivate the user exits, using the EXEC CICS DISABLE command.

Unless you control the opening of a data table explicitly, with a CEMT or EXEC CICS command, you should probably activate the user exits during CICS startup.

Otherwise loading of the data table might begin before the user exits are activated. To activate the user exits during startup, you need to:

1. Write one or more program list table postinitialization (PLTPI) programs that include the EXEC CICS ENABLE commands to activate the user exits (for programming information about PLTPI programs, see the *CICS/ESA Customization Guide*).
2. Define a program list table (PLT) with an entry for each of those PLTPI programs (see the *CICS/ESA Resource Definition Guide*).
3. Specify the **PLTPI=suffix** parameter for system initialization (see the *CICS/ESA System Definition Guide*). Use the suffix of the PLT that was defined in the previous step. This causes the PLTPI programs to be executed in the second stage of initialization, before any files are opened.

You can use PLT shutdown (PLTSD) programs in a similar way to disable the user exits during CICS shutdown.





---

## Chapter 9. Problem determination

This chapter describes the trace and dump information that is produced by CICS to help you determine the cause of a problem with data tables.

*This chapter contains Diagnosis, Modification, or Tuning Information.*

Explanations of the diagnostic messages and abend codes produced by SDT are contained in the *CICS/ESA Messages and Codes* manual.

---

### Trace information

The trace table produced by CICS helps you to determine the cause of a problem. It shows the flow of control through the CICS modules. This section describes the entries included in the trace table by data table services. For information on the contents of the trace table and how to obtain it, see the *CICS/ESA Diagnosis Reference*.

There are two types of trace points:

- Entry and exit trace points for each of the services provided by SDT. File control level-2 tracing must be enabled to obtain these trace points.
- Exception trace points.

Both of these types are listed separately below.

### Entry and exit trace points

The following entry and exit trace points are provided by SDT:

<b>0B13</b>	Entry to Remote Read service
<b>0B14</b>	Exit from Remote Read service
<b>0B1B</b>	Entry to Initialize Data Table Support service
<b>0B1C</b>	Exit from Initialize Data Table Support service
<b>0B1D</b>	Entry to Logon service
<b>0B1E</b>	Exit from Logon service
<b>0B1F</b>	Entry to Load service
<b>0B20</b>	Exit from Load service
<b>0B21</b>	Entry to Open, Close, Set Enablement and Statistics services
<b>0B22</b>	Exit from Open, Close, Set Enablement and Statistics services
<b>0B23</b>	Entry to local read services
<b>0B24</b>	Exit from local read services
<b>0B25</b>	Entry to update (add record, add, replace, delete) services
<b>0B26</b>	Exit from update services
<b>0B2D</b>	Entry to Connect and Disconnect services
<b>0B2E</b>	Exit from Connect and Disconnect services

The format of each of these trace points is described in the *CICS/ESA Diagnosis Reference* manual.

## Function and qualifier flags

Each entry and exit trace point contains a function field, and most of them contain a qualifier flags field. The function field is a byte that identifies the function that was being performed; the qualifier flags field is a byte that contains flags that qualify some of the functions. The values of these fields are:

<b>Function</b>	<b>Qualifier flags</b>
X'00' Initialize	X'00' as shared data table server X'80' as shared data table requester
X'02' Add entry from source data set to table	X'00' add issued as a result of a read request X'40' add issued by load transaction
X'03' Write entry to table	X'00' completed write X'80' pre-write for CMT
X'04' Rewrite entry in table	X'00' completed rewrite X'80' pre-rewrite for CMT
X'05' Delete entry in table	X'00' completed delete X'80' pre-delete for CMT
X'06' Commit this unit of work	
X'07' Rollback this unit of work	
X'08' Load data table (on exit trace only)	X'00' load OK X'80' source file is empty
X'09' Point at a record	X'80' equal match X'40' greater than match X'20' less than match (the above can be in various combinations) X'10' test if data table is enabled
X'0A' Retrieve record by key	X'80' equal match X'40' greater than match X'20' less than match (the above can be in various combinations) X'10' test if data table is enabled
X'0B' Retrieve record by token (internal fastpath for a sequence of records)	X'80' equal match X'40' greater than match X'20' less than match (the above can be in various combinations) X'10' test if data table is enabled
X'0C' Logon as a server	
X'0E' Open a data table	
X'0F' Close a data table	
X'10' Collect statistics	
X'11' Set enablement state	X'00' enable data table X'80' disable data table X'40' force disablement (always combined with disable)
X'15' Connect to a shared data table	
X'16' Break connection to a shared data table	
X'17' Process the completion of loading	

## Response codes

Each exit trace point contains a two-byte response-code and reason-code field. The first byte is the response code, for which the possible values are:

<b>X'01'</b>	Successful
<b>X'02'</b>	Exception
<b>X'03'</b>	Disaster
<b>X'04'</b>	Invalid
<b>X'06'</b>	Purged

## Reason codes

Each exit trace point contains a two-byte response-code and reason-code field. The second byte is the reason code, for which the possible values are:

<b>X'01'</b>	Record not in data table
<b>X'02'</b>	Duplicate (record already in data table)
<b>X'03'</b>	Data table full (already contains the maximum number of records)
<b>X'04'</b>	Record rejected by user exit
<b>X'05'</b>	Failed to get storage <sup>3</sup>
<b>X'06'</b>	Record not in data table (and table is known to be complete)
<b>X'07'</b>	Data table service failed <sup>3</sup>
<b>X'08'</b>	Not authorized to connect to file <sup>3</sup>
<b>X'09'</b>	Resource is not a data table
<b>X'0A'</b>	Remote system has not logged on as a server
<b>X'0B'</b>	Load request failed <sup>3</sup>
<b>X'0C'</b>	Data table is disabled
<b>X'0D'</b>	Add request (from DASD) deliberately not processed
<b>X'0E'</b>	Record too long
<b>X'0F'</b>	Data table token invalid
<b>X'10'</b>	Record not in data table (but might be in source data set)
<b>X'11'</b>	Data table not closed as other files are still using it
<b>X'12'</b>	Reserved
<b>X'13'</b>	Record is in data table but not currently valid
<b>X'14'</b>	File cannot be closed as it is disabled
<b>X'15'</b>	Protocol error <sup>3</sup>
<b>X'16'</b>	CICS is not an MVS subsystem
<b>X'17'</b>	Not authorized to connect to this file <sup>3</sup>

---

<sup>3</sup> This reason code might have accompanying error code information. The error code is a four-byte field that is also reported in either an error message or an exception trace point. The possible values are described in the *CICS/ESA Messages and Codes* manual, "Analyzing errors from the SVC" on page 67, and "Analyzing errors from cross-memory services" on page 70.

- X'18' CICS cannot use cross-memory services
- X'19' Interface parameter block format not recognized

### UMT and other flags

This flag byte is included in the entry trace point on OPEN. The significant bits at open time are:

- B'1.....' CICS-maintained data table
- B'01.....' Recoverable user-maintained data table
- B'00.....' Nonrecoverable user-maintained data table

## Exception trace points

The following exception trace points are provided by SDT:

- AP 0B0A Unrecognized function on call to DFHDTRF
- AP 0B0B Unrecognized function on call to DFHDTRR
- AP 0B0C Unrecognized function on call to DFHDTRP
- AP 0B0D Unrecognized function on call to DFHDTRT
- AP 0B0E Unrecognized function on call to DFHDTRSS
- AP 0B0F Unrecognized function on call to DFHDTRC
- AP 0B10 Error on initializing record management
- AP 0B11 Error on record manager OPEN
- AP 0B12 Error on record manager CLOSE
- AP 0B15 Unexpected error on call to retrieval PC
- AP 0B19 Error calling data tables SVC
- AP 0B1A Error calling data tables SVC
- AP 0B27 CLOSE could not find table block
- AP 0B28 CLOSE could not find file block
- AP 0B29 Error calling data tables SVC
- AP 0B2A Error calling data table SVC
- AP 0B2B XDTRD exit returned invalid record length (that is, it changed the length for a CMT, or increased the length for a UMT)
- AP 0B2C Connect index exceeds maximum supported size
- AP 0B2F Disastrous error when acquiring storage to pass parameters to loading transaction

The format of each of these trace points is described in the *CICS/ESA Diagnosis Reference* manual. The following two sections contain guidance on interpreting some of the information that is traced.

---

## Analyzing errors from the SVC

Following an error from a call to the data tables SVC, an exception trace point is always made, which includes an error code field identifying the reason for the error. There are three categories of SVC error:

1. Conditions that are expected to occur, such as the remote file on a connect attempt not being a data table, or the remote system not having logged on as a shared data tables server. CICS takes the appropriate action for such conditions, and no diagnostic information is needed.
2. Errors that could be due to problems in the environment, which it might be possible to correct. For these errors, a message is issued in which the reason code for the error is reported. The explanation of the reason code is included in the explanation of the message (see the *CICS/ESA Messages and Codes* manual).
3. Errors that indicate some sort of logic problem, or a misuse of the routines, possibly in an attempt to circumvent integrity or security checks. These errors are treated by CICS file control as disastrous errors, resulting in a system dump (if you have enabled such dumping) and, in most cases, in the transaction being abended with an AFCZ ABEND. For these, the value of the response and reason field is normally X'0215'.

## Error codes

This section explains the error codes for the third category of errors that is described above. These error codes are seen only in the exception trace entry. The format of the error code is X'*ffaaaaaa*', in which *ff* identifies the type of failure, and *aaaaaa* is additional information provided for some of the failures. The possible values of *ff* for each trace point are described below.

### Values for all trace points

The following error codes can occur for the 0B12, 0B19, 0B1A, 0B29, and 0B2A exception trace points:

- |              |   |
|--------------|---|
| <b>X'01'</b> | A function was specified that requires the caller to be authorized via the CICS AFCB (authorized function control block), but the caller was not authorized.                      |
| <b>X'0A'</b> | The caller passed an invalid function code.   |
| <b>X'0B'</b> | The caller specified an invalid format of SVC call.   |
| <b>X'0C'</b> | An invalid parameter list address was passed to the SVC.  |
| <b>X'0D'</b> | A function was specified that requires the value passed in register 1 to be 0, but it was not. The additional information contains the low-order three bytes of the value passed. |
| <b>X'12'</b> | A function was specified that requires the caller to be in Key 0 supervisor state, but the caller was not.  |

### Values for 0B12 trace point

The 0B12 exception trace point is issued if an error is returned by the SVC on adding or deleting an access list entry when a shared data table is being closed. In addition to the errors that can occur at all trace points, the following are possible:

- X'02'** The CICS region has not yet performed SDT initialization (an anchor block for the region has not been created).
- X'0E'** The specified data space STOKEN is invalid or the caller is not authorized to use it.
- X'0F'** The CICS region has not completed initialization as a server.
- X'13'** An attempt to delete an access list entry failed because the specified entry was not created by the data tables SVC.

All other errors result in a message being issued that contains the error code.

### Values for 0B19 trace point

The 0B19 exception trace point is issued if an error is returned by the SVC on initializing as a shared data table server. In addition to the errors that can occur at all trace points, the following are possible:

- X'02'** An attempt was being made to add an access list entry before the CICS region had performed SDT initialization (an anchor block for the region had not yet been created).
- X'0E'** The specified data space STOKEN is invalid or the caller is not authorized to use it.
- X'0F'** An attempt was being made to add an access list entry before the CICS region had completed server initialization.

All other errors result in a message being issued that contains the error code.

### Values for 0B1A trace point

The 0B1A exception trace point is issued if an error is returned by the SVC on initializing as a shared data table requester. In addition to the errors that can occur at all trace points, the following are possible:

- X'05'** The CICS region has already initialized as a shared data table requester, but is now running under a different request block from that under which it originally initialized.

All other errors result in a message being issued that contains the error code.

### Values for 0B29 trace point

The 0B29 exception trace point is issued if an error is returned by the SVC on logging on as a shared data table server. In addition to the errors that can occur at all trace points, the following are possible:

- X'02'** The CICS region that is attempting to register (logon) as a server has not yet been initialized (an anchor block for the region has not been created).
- X'04'** This CICS region has already registered (logged on) as a shared data tables server.
- X'0F'** The CICS region has not completed server initialization.

- X'14'** The AFCS anchor block does not exist.
- X'15'** The CICS security block does not exist.
- X'16'** Either the caller is not running in a user protection key (its PSW key is less than 8), or the caller's TCB does not normally execute in a user protection key (TCBPKF is less than 8).

All other errors result in a message being issued that contains the error code.

### Values for 0B2A trace point

If the function code field contains X'15' then the 0B2A exception trace point indicates an error on CONNECT (that is, on attempting to establish a connection to a remote file). In addition to the errors that can occur at all trace points, the following are possible:

- X'02'** The requesting region has not performed SDT initialization (an anchor block for the region has not been created).
- X'03'** The requesting region has not completed initialization as a shared data tables requester.
- X'05'** The CICS region is running under a different request block (RB) from when it initialized as a data table requester. The additional information part of the error code contains the RB address under which the call was made.
- X'72'** The LINK to the user-replaceable DFHACEE module to find the home address space's security userid has failed. The additional information part of the error code contains two bytes of the ABEND code from the LINK. The response and reason field accompanying this error is X'020B'.

All other errors result in a message being issued that contains the error code.

If the function code field contains X'16' then the 0B2A exception trace point indicates an error on DISCONNECT (that is, on attempting to break the connection to a remote file). In addition to the errors that can occur at all trace points, the following are possible:

- X'02'** The requesting region has not performed SDT initialization (an anchor block for the region has not been created).
- X'03'** The requesting region has not completed initialization as a shared data tables requester.
- X'05'** The CICS region is running under a different request block (RB) from when it initialized as a data table requester. The additional information part of the error code contains the RB address under which the call was made.
- X'07'** The caller has supplied an invalid index into the vector of file connections. The additional information part of the error code contains the low-order three bytes of the caller's index.
- X'10'** The specified connection was broken previously and no longer exists. The additional information part of the error code contains the low-order three bytes of the caller's index into the vector of file connections.

All other errors result in a message being issued that contains the error code.

---

## Analyzing errors from cross-memory services

Following an unexpected error from data tables cross-memory services, an **X'0B15'** exception trace entry is made. It includes the response and reason codes and an error code field identifying the cause of the error. Such errors are all caused either by a corruption of the routines or of the system, or by a possible misuse of the routines.

For a response and reason of X'0215', the format of the error code is X'ffaaaaaa', in which *ff* identifies the type of failure and *aaaaaa* is additional information provided for some of the failures. The possible values of *ff* are:

- X'01'** An attempt to locate the CICS AFEB (authorized function control block) made by either the cross-memory retrieval routine or the connect vector lookup routine has failed.
- X'02'** The requesting CICS region has not yet performed SDT initialization (an anchor block for the region has not yet been created and set up).
- X'03'** The requesting region has not completed initialization as a shared data tables requester.
- X'05'** The retrieval request was issued under a request block different from the one that performed initialization as an SDT requester.
- X'06'** The connect vector entry for the remote file does not contain the correct linkage index.
- X'07'** The index of the connect vector entry for the remote file is beyond the end of the connect vector.
- X'08'** The connect vector entry for the remote file is not marked as being in use.
- X'09'** The cross-memory retrieval routine has not been called via the correct mechanism.

A response and reason of X'0400' means that the function code passed to the record management code running in the server region was an unrecognized value.

---

## Dump information

Information relevant to data tables is included in a CICS system dump to help you determine the cause of a problem. For information on the contents of dumps and how to obtain them, see the *CICS/ESA Problem Determination Guide*.

The major control blocks that are used by SDT are included in the FILE CONTROL area of a formatted dump of the file-owning region. These control blocks are named:

- Data Table Global Area.  
This is also known as the SDT Header Block, so it uses the eye-catcher DFHDTHDTER.
- Data Table Base Area.  
This is also known as the SDT Table Block, so it uses the eye-catcher DFHDTTABLE.



- Data Table Path Area.  
This is also known as the SDT File Block, so it uses the eye-catcher DFHDTFILE.

The data table contents are not included in the CICS system dump because the data space in which the data table resides is not part of the CICS address space. If you want to see the contents of the data table, ask the system operator to use the MVS DUMP command to request a dump of the data space DFHDT001 owned by the appropriate CICS startup job.

The operator command DISPLAY J,CICS-startup-jobname shows information about a CICS job, including the DSPNAMEs of data spaces that it owns. To dump the contents of the data space, you can use the MVS DUMP command as follows:

1. Enter

```
DUMP COMM=(title for your dump)
```

2. This generates an MVS console message

```
* id IEE094D SPECIFY OPERAND(S) FOR DUMP COMMAND
```

3. Reply to the message with

```
REPLY id, DSPNAME='jobname'.DFHDT001
```

and the data space is dumped.

4. Use DISPLAY DUMP,TITLE to see which SYS1.DUMPnn data set has been used.



---

## # Chapter 10. Using Shared Data Tables support in a sysplex

# This chapter discusses the use of Shared Data Tables support in a sysplex  
# environment. It should be of particular interest to you if you are currently using  
# user-maintained data tables in a single-MVS environment, but are planning to move  
# to a sysplex. It might also be helpful if you already have a sysplex, because it can  
# show you how to exploit shared data tables support in that environment.

# A shared data table can exploit shared data tables support only within a single  
# MVS image.<sup>4</sup> However, you can extend the use of shared data tables to a sysplex  
# environment for an application that requires only read access to a shared  
# user-maintained data table, or for one that does not require that changes are seen  
# immediately.

# You can replicate a user-maintained data table across the sysplex, with one data  
# table per MVS. You must have one Shared Data Tables (SDT) server region in  
# each MVS image, each owning a user-maintained data table which can be  
# accessed using SDT sharing by any of the other CICS regions within that MVS.  
# These other regions require remote file definitions, which refer to the  
# user-maintained data table in their server region. Each user-maintained data table  
# (UMT) must have the same source data set, and this data set must be readable by  
# all of the SDT server regions. If the access is read-only, with the data never being  
# updated, this will in effect provide a shared user-maintained data table in a sysplex.

# If, as is probably more likely, the underlying data changes from time to time, but it  
# is not necessary to reflect such changes immediately in the UMTs, you may  
# periodically perform some processing to refresh the contents of the UMTs so that  
# they are updated to match the underlying data without the need to close and then  
# reload the UMTs. Changes are applied to the source data set, rather than to the  
# user-maintained data table, using CICS applications that refer to the data set by a  
# non-data table file definition, or using batch programs. An example application  
# program (see Figure 11 on page 79) illustrates how the UMTs can be refreshed to  
# reflect the current contents of the source data set. The program would run on each  
# MVS image and update the UMT in that image. Such a program could be run at  
# regular times during the day, or at user request. It would be most efficient to run it  
# in the SDT server regions, to avoid function shipping updates to the UMT.

# If it is critical that the CICS regions in all the MVS images in the sysplex are  
# synchronised in their view of the data, the transactions that read the data must be  
# stopped while the refresh programs run, and restarted only after the programs have  
# completed on all MVS systems.

# This technique is appropriate only for user-maintained data tables because:

- # • Where read-only access is required, a user-maintained data table is the usual  
# choice.
- # • It would not be possible with a CICS-maintained data table to apply updates to  
# the source data set while leaving the tables unaffected.
- # • Any update made to the source data set would be reflected only in the table on  
# the system on which the update was made.

---

# <sup>4</sup> But note that a shared data table can be shared using function shipping across MVS images.

# Figure 11 on page 79 shows an example COBOL application program that  
# demonstrates how you can refresh the replicated UMTs.

---

## # How to refresh replicated UMTs

# The following steps describe how to set up an environment to refresh replicated  
# UMTs. In practice, you may already have some of this in place. For example, you  
# may already have files defined as data tables. The steps described here assume  
# that you already have a sysplex environment.

# 1. Select a file that is appropriate.

# As an illustration, consider an application that checks credit card numbers  
# against a list of stolen credit cards, and requires rapid access to this list. The  
# list is updated periodically with new batches of numbers of stolen cards. The  
# application accesses records in a VSAM KSDS data set named  
# PRODN.SOURCEDS using a filename UMTNAME. The application runs in a  
# sysplex consisting of two MVS images. CICS regions CICS1A, CICS1B, and  
# CICS1C run in the first image, and CICS2A, CICS2B and CICS2C run in the  
# second.

# 2. Set up file definitions:

# • In each MVS in the sysplex, select a CICS region to be the SDT server for  
# this file. Within this region, define the filename by which your applications  
# read the data as a user-maintained data table, with the data set name as  
# that containing the source data.

# In this illustration, CICS1A and CICS2A are set up as the server regions,  
# and files are defined to them called UMTNAME. The file definitions specify  
# DSNAMES as PRODN.SOURCEDS, TABLE as USER, and allowed  
# operations of YES for READ, BROWSE, ADD, DELETE and UPDATE  
# (because this file definition will be used both for reading the data and for  
# updating the UMT when it is refreshed).

# • For all the other regions in the sysplex, define the filename by which the  
# applications read the data as a remote file with the REMOTESYSTEM as  
# the SDT server region in the same MVS, and the REMOTENAME as the  
# name of the UMT in that region.

# So, in this illustration, files called UMTNAME would be defined in CICS1B  
# and CICS1C with the REMOTESYSTEM as the sysid for CICS1A and the  
# REMOTENAME as UMTNAME, this time with READ and BROWSE as the  
# only allowed operations, because there is no need for the UMT to be  
# updated through these remote definitions. Similar file definitions are set up  
# in CICS2B and CICS2C, but for these CICS2A is the remote system.

# • In each SDT server region, set up a file definition that can be used to read  
# the source data set when the UMTs are refreshed.

# In this illustration, files named SOURCEDS are defined to CICS1A and  
# CICS2A, with the DSNAMES as PRODN.SOURCEDS, TABLE as NO, and  
# allowing only READ and BROWSE operations.

# • In one region in the sysplex (which has access to the source data set),  
# define a file that is used to apply updates to the source. The file definition  
# could be the same as that used by the refresh program to read the source  
# data set, but in this case it would need to allow both reading and updating

```

#           operations. You might, if you prefer, decide to update the data set using a
#           batch program, in which case this CICS file definition would not be needed.
#
#           This illustration uses the same file definition as is used in refreshing the
#           UMTs. In this case, one of the regions would need to define SOURCEDS
#           as allowing all file operations.
#
# 3. Set up the source data set so that it can be accessed by all applications that
#     need to read or update it.
#
#     You can set up the data set SHAREOPTIONS so that it can be updated by the
#     program that applies updates to the source, and read by all others.
#     Alternatively, you can set up the data set so that it can be updated only when it
#     is not being read, and ensure that its opening is serialised. For the
#     shareoptions to operate throughout the sysplex, you must use GRS (Global
#     Resource Serialisation).
#
#     In this illustration, define PRODN.SOURCEDS either with:
#
#     • SHR(2), so that it can be updated by the region that runs the program that
#       applies changes to the data set and at the same time read by all the
#       refresh programs,
#
#     or
#
#     • SHR(1), and normally have it open to the program that applies changes;
#       then, when it is to be refreshed, close that access to it, and, on each server
#       region in turn, open it, run the refresh program, and close disable it to allow
#       the next region to open it.
#
# 4. Modify the example program so that it names your files for the UMT and the
#     source data set, and so that the data definitions match the layout of your
#     records. Define the program and transaction in your server regions.
#
#     The file names in the illustration are the same as those in the program
#     (UMTNAME and SOURCEDS). Define the program and a transaction to run it
#     in CICS1A and CICS2A.
#
# 5. You should now be ready to start using the replicated UMTs.
#
# 6. Prime the source data set with its initial contents.
#
# 7. Open the UMTs in the SDT server regions, to cause the contents of the source
#     data set to be loaded into each one.
#
# 8. Start up the applications in all regions in the sysplex. They will all be able to
#     access the data using SDT sharing.
#
#     The applications running in MVS 1 will access the data through the UMT in
#     CICS1A, and those running in MVS 2 will access it through the UMT in
#     CICS2A.
#
# 9. When new data arrives, update the source data set.
#
#     In this illustration, the data is updated by file SOURCEDS.
#
# 10. When you want the applications to access the new data, run the transactions in
#     each server region that will read the source data set and the UMT, and refresh
#     the latter to be in step with the former. Providing your applications are not
#     invalidated if the data seen on one MVS is slightly different from that seen on
#     another, you do not have to stop them running while you do the refresh.

```

---

## # Example program for refreshing a UMT

# To help you write your own program, Figure 11 on page 79 shows an example of  
# a COBOL program that demonstrates how to refresh a UMT while it is still open, to  
# match the source data set.

# If updates are applied frequently to the source data set, and could be applied while  
# the refresh program is running, this could mean that the source data set is never  
# exactly reflected by the UMT, because the record being processed or records  
# already processed could be changed. This means that the program has to be  
# tolerant to the possibility of the records changing. The program is also written to  
# allow for the possibility that the UMT itself is updated by other programs, although  
# you are not recommended to operate in this way (that is, the only program that  
# updates the UMT should be the refresh program).

## # Setting up and executing the example program

# Edit the program, according to the comments in the example, to match the format  
# of the records being updated. 'UMTNAME' and 'SOURCEDDS' should be renamed  
# to match your file definitions.

# Translate, compile and link the program using a COBOL compiler.

# Define the program to CICS, and define a transaction to the program. Define the  
# file (UMTNAME) to point to the UMT, and give it a source data set from which to  
# load when first opened. Define the other file (SOURCEDDS) to point directly to the  
# source data set from which the UMT is defined to load.

# Each sysplex should have one CICS region where the UMT that is to be refreshed  
# resides. In these regions, the definitions needed to run the refresh transaction  
# must be installed. In all other regions in the sysplex, the UMT should be defined as  
# a remote file, pointing to the UMT in the UMT-owning region. It is not necessary to  
# run the refresh transaction on the regions that have the UMT defined as remote.

# The update strategy used will depend on the way the source data set is set up. If  
# the data set has the SHAREOPTIONS set so that it can be read by multiple  
# systems at any one time, all UMTs can be refreshed at the same time, and any  
# updates to the source data set can also be applied. Otherwise, when the source  
# data set is updated, the file that is used to read the source data set for refreshing  
# would need to be closed and disabled on each system for the duration of the  
# update. If all the UMTs are refreshed serially, the source data set could be opened  
# and closed to each UMT-owning region in turn, and when needed for update.

## # How the example program operates

# First, the environment is initialized. A check is made that the UMT file is local and  
# is already open. If the UMT file is remote, the program issues a message and  
# ends. If the UMT file is not open, the program opens it and ends (because opening  
# the UMT will load the latest data from the source data set without the need to  
# perform any more processing). A check is also made that the source file is local; if  
# it is remote, the program issues a message and ends. The file that directly  
# accesses the UMT's source data set is opened. Start browse operations are then  
# performed on both files to allow the program to step through them both  
# sequentially.

```

# If the environment is set up without error, the update of the UMT starts. This
# involves the retrieval and comparison of pairs of records, one from the UMT and
# one from the base data set.

# The records retrieved are compared:
#
# • If the records are equal, the flags are set to read the next record from the UMT
#   and the data set.
#
# • If the UMT has a greater key than the data set, there is a record in the data set
#   that must be added to the UMT.
#
# • If the data set has a greater key than the UMT, there is an extra record in the
#   UMT that needs to be removed.
#
# • If the keys are equal, but the records are different, the UMT should be updated
#   with the record in the data set.

# If a record must be added to the UMT, a write operation is performed.
#
# • If the write operation succeeds, the program goes on to process the next pair
#   of records.
#
# • If the write operation fails because of a record that has been inserted by
#   another transaction between the read and the write operation performed by the
#   program, an attempt is made to delete the record and write it again.
#
# • If the second attempt fails, the program processes the next pair of records.
#
# • When the next pair of records is processed, the current UMT record is
#   compared with the next record in the data set to check for further UMT record
#   omissions.

# If a record needs to be deleted from the UMT, a delete operation is performed.
#
# • If the delete operation succeeds, the program goes on to process the next pair
#   of records.
#
# • If the delete operation fails because the record has already been deleted
#   between the read and delete operations, the program continues to process the
#   next pair of records.
#
# • When the next pair of records is processed, the current data set record is
#   compared with the next record in the UMT to check for further records that
#   should not be in the UMT.

# If a record must be updated in the UMT, a read for update operation is performed,
# to get a lock on the record.
#
# • If this is a success, the updated record is rewritten to the UMT, and the
#   program goes on to process the next pair of records.
#
# • If the operation fails because another transaction has deleted the record, a
#   write operation is performed to put it back in.
#
# • If the write operation fails, the program continues to process the next pair of
#   records.
#
# • When the next pair of records is processed, new records are read from both
#   the UMT and the data set.

```

```
#      When the end of both files has been reached, and there are no more records left to
#      process, the program performs end browses on both the data set and the UMT and
#      returns. Note that the example does not close the file that directly accesses the
#      data set. If the data set cannot operate for update in a shared environment, the file
#      that accesses it should be set to CLOSED DISABLED, to allow it to be updated.
```

```
#      The program traps any unexpected errors and issues an error message on the
#      screen. Only the first operation on the UMT is checked (either the delete, write or
#      read/rewrite operations). If that fails with a return code that could be caused by a
#      record being changed after it was originally read, one final attempt is made to
#      correct the record, but this attempt is not checked. This is to prevent the program
#      entering a loop state.
```

```
#      There are further comments in the code.
```



## # Source code for the example program

```
#          CBL XOPTS(SP)
#          *****
#          *
#          * PROGRAM NAME : UMTUPDT COBOL
#          *
#          * DESCRIPTIVE NAME : CICS application to dynamically update a
#          *                   UMT with the current contents of a dataset.
#          *
#          *-----*
#          *
#          * OVERVIEW
#          *
#          * This program demonstrates how to update a user maintained
#          * table (UMT) to match the data in the source dataset it was
#          * loaded from when opened, whilst it remains in use by one
#          * (or more) CICS systems. It can be used to update a UMT that
#          * is replicated in different sysplexes so that they all match
#          * the source dataset. It should be run on the FOR.
#          *
#          *-----*
#          *
#          * REQUIREMENTS
#          *
#          * This program should be translated, compiled and linked as a
#          * CICS COBOL program, and defined to CICS. A transaction name
#          * should be defined to this program. A UMT file, currently
#          * called UMTNAME, is used to access the UMT, and a source
#          * dataset file, currently called SOURCEDDS, is used to directly
#          * access the dataset the UMT is loaded from. These definitions
#          * must be installed only in the region in which the UMT resides
#          * (the FOR). Any regions in the same sysplex that use the UMT
#          * remotely do not need to run any update process.
#          *
#          *-----*
#          *
#          * DESCRIPTION
#          *
#          * The program will first initialise the two files that are
#          * needed, and start browsing them from the beginning.
#          * Opening the UMT will cause it to be loaded if it isn't open.
#          * If it is not open and the UMT is loaded, the operation of the
#          * program is effectively redundant and the update code will
#          * not be run. The program will also check for a remote system
#          * name. If one is present for either file, then the program
#          * will not run. This is to prevent function shipping occurring
#          * which would obviously degrade performance.
#          *
#          * The program will continuously read a pair of records from the
#          * two files and compare them, adding, deleting or updating any
#          * records in the UMT that don't match the source dataset.
#          *
#          *
```

Figure 11 (Part 1 of 13). Example program to refresh a replicated UMT

```

# * The keys of the pair of records are compared. If the key to *
# * the UMT and the key to the source dataset are equal, and the *
# * records match, then no update is required. If both keys are *
# * equal, but the records are different, then the record in the *
# * source dataset is used to update the UMT. If the key in the *
# * UMT is greater than the key in the source dataset, then the *
# * record(s) in the source dataset are written to the UMT until *
# * the keys become equal or the UMT key becomes less than the *
# * source dataset key. If the UMT key is less than the source *
# * dataset key, then the record(s) in the UMT are removed until *
# * the keys become equal, or the UMT key is greater than the *
# * source dataset. This continues until the end of both files *
# * is reached, or an unexpected error occurs. *
# * *
# * Any errors that are unexpected are reported to the screen, *
# * and operation of the program stops. Some errors are trapped, *
# * and a further attempt will be made to update the UMT. If *
# * this attempt fails, no further action is taken for those *
# * records, and the program will continue to process the next *
# * pair. *
# * *
# *-----*
# * *
# * MODIFYING THE PROGRAM *
# * *
# * This program may not work as is. The record structure it *
# * uses assumes that a 4 character key is used to access a 40 *
# * character record. The following changes will need to be made *
# * to allow this program to work with different record types. *
# * *
# * The key that accesses the UMT and source dataset should be *
# * changed. The variables that store the key are UMT-KEY and *
# * DS-KEY. *
# * *
# * The length of the records are held in UMT-LEN and DS-LEN. *
# * *
# * The UMT and source dataset record variables should be changed.*
# * The variables that store these are UMT-REC (which contains *
# * UMT-REC-KEY and UMT-REC-TEXT), and DS-REC (which contains *
# * DS-REC-KEY and DS-REC-TEXT). Additional fields can obviously *
# * be added as needed. *
# * *
# * The filename of the UMT is set as UMTNAME. This can be *
# * changed to match any UMT already defined. The source dataset *
# * file is set as SOURCEDS, and can also be changed. *
# * *
# *
# *****
#
#

```

Figure 11 (Part 2 of 13). Example program to refresh a replicated UMT

```

#          IDENTIFICATION DIVISION.
#          PROGRAM-ID. UMTUPDT.

#          ENVIRONMENT DIVISION.
#          EJECT.

#          DATA DIVISION.

#          WORKING-STORAGE SECTION.

#          * Declare the UMT and DS record variables
#          77 UMT-KEY          PIC X(4)  VALUE '0000'.
#          77 UMT-LEN          PIC 9(2)  VALUE 40.
#          01 UMT-REC.
#             03 UMT-REC-KEY   PIC X(4)  VALUE SPACES.
#             03 UMT-REC-TEXT  PIC X(36) VALUE SPACES.

#          77 DS-KEY          PIC X(4)  VALUE '0000'.
#          77 DS-LEN          PIC 9(2)  VALUE 40.
#          01 DS-REC.
#             03 DS-REC-KEY   PIC X(4)  VALUE SPACES.
#             03 DS-REC-TEXT  PIC X(36) VALUE SPACES.

#          * Declare other work variables
#          * Screen output strings
#          01 MESSAGE-OUTPUT  PIC X(26) VALUE 'UMT SUCCESSFULLY REFRESHED'.
#          01 REMOTE-OUTPUT   PIC X(25) VALUE 'FILE RESOURCE NOT LOCAL'.
#          01 ERROR-OUTPUT.
#             03 ERROR-OPNAME PIC X(8)  VALUE SPACES.
#             03 FILLER       PIC X(15) VALUE ' RETURNED RESP '.
#             03 ERROR-RESP   PIC X(8)  VALUE SPACES.
#             03 FILLER       PIC X(7)  VALUE ' RESP2 '.
#             03 ERROR-RESP2  PIC X(8)  VALUE SPACES.
#             03 FILLER       PIC X(10) VALUE ' FOR FILE '.
#             03 ERROR-FILE   PIC X(8)  VALUE SPACES.

#          * End of file flags
#          77 UMT-EOF          PIC 9(1)  VALUE 0.
#          77 DS-EOF          PIC 9(1)  VALUE 0.

#          * Record retrieval flags
#          77 GET-NEXT-UMT    PIC 9(1)  VALUE 1.
#          77 GET-NEXT-DS    PIC 9(1)  VALUE 1.

#          * File inquire variables
#          77 REM-SYS-NAME    PIC X(4)  VALUE SPACES.
#          77 OPEN-STAT      PIC S9(8)  BINARY.

#          * Program operation flags
#          77 PROCESS-FILES   PIC 9(1)  VALUE 1.
#          77 REM-FILE        PIC 9(1)  VALUE 0.
#          77 UMT-STARTBR    PIC 9(1)  VALUE 0.
#          77 DS-STARTBR    PIC 9(1)  VALUE 0.
#

```

Figure 11 (Part 3 of 13). Example program to refresh a replicated UMT

```

#          * EXEC CICS response variables
#          77 RESPONSE          PIC S9(8) BINARY.
#          77 RESPONSE2        PIC S9(8) BINARY.

#          COPY DFHAID.
#          COPY DFHBMSCA.

#          LINKAGE SECTION.
#          EJECT.

#          PROCEDURE DIVISION USING DFHEIBLK.

#          *****
#          * Main processing starts here.                               *
#          *****
#          MAIN-PROCESSING SECTION.

#          * Check the UMT and dataset for processing
#          PERFORM FILE-CHECK.

#          * If the file check completed okay, process the UMT
#          IF (PROCESS-FILES = 1)

#          * Ready the UMT and DS for access
#          PERFORM INITIALISE

#          * Call the update routine until the end of both files reached
#          PERFORM UPDATE-UMT UNTIL (DS-EOF = 1 AND UMT-EOF = 1)

#          END-IF.

#          * Exit the program cleanly
#          PERFORM TRAN-FINISH.

#          MAIN-PROCESSING-EXIT.
#          GOBACK.
#          EJECT

#          *****
#          * Procedures start here.                                     *
#          *****

#
#          Figure 11 (Part 4 of 13). Example program to refresh a replicated UMT

```

```

# *****
# * Check the files open status and that they aren't remote *
# *****
# FILE-CHECK SECTION.

#
# * Inquire on the UMT to get remote and open status information
#   MOVE SPACES TO REM-SYS-NAME.
#   EXEC CICS INQUIRE FILE('UMTNAME')
#     OPENSTATUS(OPEN-STAT)
#     REMOTESYSTEM(REM-SYS-NAME)
#     RESP(RESPONSE)
#     RESP2(RESPONSE2)
#   END-EXEC.
# * Output an error if inquire on the UMT failed
#   IF (RESPONSE NOT = DFHRESP(NORMAL))
#     MOVE 'INQUIRE ' TO ERROR-OPNAME
#     MOVE 'UMTNAME ' TO ERROR-FILE
#     PERFORM PROCESS-ERROR
#   END-IF.
# * System name is not blank if the file is defined as remote
# * We don't want to do any processing if the file is remote
#   IF (REM-SYS-NAME NOT = SPACES)
#     MOVE 0 TO PROCESS-FILES
#     MOVE 1 TO REM-FILE
#   ELSE
# * If the UMT is not open, then opening it will update it
#   IF (OPEN-STAT NOT = DFHVALUE(OPEN))
#     EXEC CICS SET FILE('UMTNAME')
#       OPEN
#       RESP(RESPONSE)
#       RESP2(RESPONSE2)
#     END-EXEC
# * Check open of UMT was successful
#   IF (RESPONSE NOT = DFHRESP(NORMAL))
#     MOVE 'OPEN ' TO ERROR-OPNAME
#     MOVE 'UMTNAME ' TO ERROR-FILE
#     PERFORM PROCESS-ERROR
#   ELSE
# * Don't want to do any processing, as open will update UMT
#   MOVE 0 TO PROCESS-FILES
#   END-IF
# END-IF.
# END-IF.

#
# * Inquire on the source dataset to get remote and open status
#   MOVE SPACES TO REM-SYS-NAME.
#   EXEC CICS INQUIRE FILE('SOURCEDS')
#     REMOTESYSTEM(REM-SYS-NAME)
#     OPENSTATUS(OPEN-STAT)
#     RESP(RESPONSE)
#     RESP2(RESPONSE2)
#   END-EXEC.
#

```

Figure 11 (Part 5 of 13). Example program to refresh a replicated UMT



```

#           * Output an error if the start browse for the UMT failed
#           IF (RESPONSE NOT = DFHRESP(NORMAL))
#           MOVE 'STARTBR ' TO ERROR-OPNAME
#           MOVE 'UMTNAME ' TO ERROR-FILE
#           PERFORM PROCESS-ERROR
#           END-IF
#           END-IF.
#           * Set UMT start browse flag
#           MOVE 1 TO UMT-STARTBR.

#           * Start browsing the dataset from the first record
#           EXEC CICS STARTBR FILE('SOURCED')
#           RIDFLD(DS-KEY)
#           GTEQ
#           RESP(RESPONSE)
#           RESP2(RESPONSE2)
#           END-EXEC.
#           * If dataset is empty then treat as end of dataset an empty UMT
#           IF (RESPONSE = DFHRESP(NOTFND))
#           MOVE 1 TO DS-EOF
#           ELSE
#           * Output an error if the start browse for the dataset failed
#           IF (RESPONSE NOT = DFHRESP(NORMAL))
#           MOVE 'STARTBR ' TO ERROR-OPNAME
#           MOVE 'SOURCED' TO ERROR-FILE
#           PERFORM PROCESS-ERROR
#           END-IF
#           END-IF.
#           * Set dataset start browse flag
#           MOVE 1 TO DS-STARTBR.

#           INITIALISE-EXIT.
#           EXIT.
#           EJECT

#           *****
#           * Update the UMT according to the record/key states *
#           *****
#           UPDATE-UMT SECTION.

#           * Get the next records from the UMT and dataset
#           PERFORM READ-FILES.

#           * If both records are the same, move to the next record
#           IF UMT-REC = DS-REC
#           MOVE 1 TO GET-NEXT-UMT
#           MOVE 1 TO GET-NEXT-DS
#           ELSE
#           *
#
#           Figure 11 (Part 7 of 13). Example program to refresh a replicated UMT

```

```

#           * If UMT is behind dataset then extra record in UMT so delete it.
#           * Also delete records from UMT if EOF DS reached before EOF UMT
#             IF (UMT-EOF = 0 AND (UMT-KEY < DS-KEY OR DS-EOF = 1))
#               PERFORM UMT-DELETE
#             END-IF

#           * If UMT ahead of dataset then extra record in DS so add to UMT
#           * Also add records to the UMT if the EOF reached before EOF DS
#             IF (DS-EOF = 0 AND (UMT-KEY > DS-KEY OR UMT-EOF = 1))
#               PERFORM UMT-WRITE
#             END-IF

#           * If both keys equal but record different, update UMT
#             IF ((DS-EOF = 0 AND UMT-EOF = 0) AND UMT-KEY = DS-KEY)
#               PERFORM UMT-UPDATE
#             END-IF

#           END-IF.

#           UPDATE-UMT-EXIT.
#           EXIT.
#           EJECT

#           *****
#           * Read the next record from both files *
#           *****
#           READ-FILES SECTION.

#           * If the flags are set to read the next UMT record, do so
#             IF (GET-NEXT-UMT = 1 AND UMT-EOF = 0)
#               MOVE SPACES TO UMT-REC
#               EXEC CICS READNEXT FILE('UMTNAME')
#                 RIDFLD(UMT-KEY)
#                 INTO(UMT-REC)
#                 RESP(RESPONSE)
#                 RESP2(RESPONSE2)
#             END-EXEC

#           * Set the EOF flag if the end of the UMT has been reached
#             IF (RESPONSE = DFHRESP(ENDFILE))
#               MOVE 1 TO UMT-EOF
#             ELSE

#           * Output an error if the return code from the READ is unexpected
#             IF (RESPONSE NOT = DFHRESP(DUPKEY) AND
#               RESPONSE NOT = DFHRESP(NORMAL))
#               MOVE 'READNEXT' TO ERROR-OPNAME
#               MOVE 'UMTNAME ' TO ERROR-FILE
#               PERFORM PROCESS-ERROR
#             END-IF
#           END-IF
#         END-IF.

#
#           Figure 11 (Part 8 of 13). Example program to refresh a replicated UMT

```



```

#           * If the flags are set to read the next dataset record, do so
#           IF (GET-NEXT-DS = 1 AND DS-EOF = 0)
#             MOVE SPACES TO DS-REC
#             EXEC CICS READNEXT FILE('SOURCEDS')
#               RIDFLD(DS-KEY)
#               INTO(DS-REC)
#               RESP(RESPONSE)
#               RESP2(RESPONSE2)
#             END-EXEC
#           * Set the EOF flag if the end of the dataset has been reached
#           IF (RESPONSE = DFHRESP(ENDFILE))
#             MOVE 1 TO DS-EOF
#           ELSE
#           * Output an error if the return code from the READ is unexpected
#             IF (RESPONSE NOT = DFHRESP(DUPKEY) AND
#                RESPONSE NOT = DFHRESP(NORMAL))
#               MOVE 'READNEXT' TO ERROR-OPNAME
#               MOVE 'SOURCEDS' TO ERROR-FILE
#               PERFORM PROCESS-ERROR
#             END-IF
#           END-IF
#         END-IF.

#       READ-FILES-EXIT.
#       EXIT.
#       EJECT

#
# *****
# * Attempt to delete a record from the UMT *
# *****
#       UMT-DELETE SECTION.

#
# * Delete the last read record in the UMT
#       EXEC CICS DELETE FILE('UMTNAME')
#         RIDFLD(UMT-KEY)
#         RESP(RESPONSE)
#         RESP2(RESPONSE2)
#       END-EXEC.
#
# * Allow NORMAL and NOTFND return codes in case record has been
# * deleted since it was first read, otherwise output an error
#       IF (RESPONSE = DFHRESP(NORMAL) OR
#          RESPONSE = DFHRESP(NOTFND))
#
# * Set flags to get next UMT record, but keep same dataset record
#       MOVE 1 TO GET-NEXT-UMT
#       MOVE 0 TO GET-NEXT-DS
#     ELSE
#       MOVE 'DELETE ' TO ERROR-OPNAME
#       MOVE 'UMTNAME ' TO ERROR-FILE
#       PERFORM PROCESS-ERROR
#     END-IF.

#       UMT-DELETE-EXIT.
#       EXIT.
#       EJECT
#

```

Figure 11 (Part 9 of 13). Example program to refresh a replicated UMT

```

# *****
# * Attempt to write a record to the UMT *
# *****
# UMT-WRITE SECTION.

# * Attempt to write the missing record using the dataset key
# EXEC CICS WRITE FILE('UMTNAME')
# RIDFLD(DS-KEY)
# FROM(DS-REC)
# RESP(RESPONSE)
# RESP2(RESPONSE2)
# END-EXEC.
# * If the UMT has had a record written to this position since the
# * read then delete it and try one last time.
# * If write still unsuccessful, move to the next pair of records
# IF RESPONSE = DFHRESP(DUPREC)
# EXEC CICS DELETE FILE('UMTNAME')
# RIDFLD(DS-KEY)
# RESP(RESPONSE)
# RESP2(RESPONSE2)
# END-EXEC
# EXEC CICS WRITE FILE('UMTNAME')
# RIDFLD(DS-KEY)
# FROM(DS-REC)
# RESP(RESPONSE)
# RESP2(RESPONSE2)
# END-EXEC
# ELSE
# * Output an error if return code from first write was bad
# * (but allow suppression return code by user exit)
# IF (RESPONSE NOT = DFHRESP(NORMAL) AND
# RESPONSE NOT = DFHRESP(SUPPRESSED))
# MOVE 'UMTNAME ' TO ERROR-FILE
# MOVE 'WRITE ' TO ERROR-OPNAME
# PERFORM PROCESS-ERROR
# END-IF
# END-IF.

# * Set flags to keep same UMT record, and get next dataset record
# MOVE 0 TO GET-NEXT-UMT.
# MOVE 1 TO GET-NEXT-DS.

# UMT-WRITE-EXIT.
# EXIT.
# EJECT

```

Figure 11 (Part 10 of 13). Example program to refresh a replicated UMT

```

# *****
# * Attempt to update a record in the UMT to match the DS *
# *****
# UMT-UPDATE SECTION.

#
# * Attempt to get a lock on the record using read for update
# EXEC CICS READ FILE('UMTNAME')
# RIDFLD(UMT-KEY)
# INTO(UMT-REC)
# UPDATE
# RESP(RESPONSE)
# RESP2(RESPONSE2)
# END-EXEC.
# * If record has been deleted since original read, write it.
# * If write is unsuccessful, move to next pair of records
# IF RESPONSE = DFHRESP(NOTFND)
# EXEC CICS WRITE FILE('UMTNAME')
# RIDFLD(UMT-KEY)
# FROM(DS-REC)
# RESP(RESPONSE)
# RESP2(RESPONSE2)
# END-EXEC
# ELSE
# * If read for update was successful, write dataset record to UMT
# IF RESPONSE = DFHRESP(NORMAL)
# EXEC CICS REWRITE FILE('UMTNAME')
# FROM(DS-REC)
# RESP(RESPONSE)
# RESP2(RESPONSE2)
# END-EXEC
# * Output an error if rewrite failed
# IF RESPONSE NOT = DFHRESP(NORMAL)
# MOVE 'REWRITE ' TO ERROR-OPNAME
# MOVE 'UMTNAME ' TO ERROR-FILE
# PERFORM PROCESS-ERROR
# END-IF
# ELSE
# * Output an error if the read for update failed
# MOVE 'READUPDT' TO ERROR-OPNAME
# MOVE 'UMTNAME ' TO ERROR-FILE
# PERFORM PROCESS-ERROR
# END-IF
# END-IF.

#
# * Set flags to get next record for both UMT and dataset
# MOVE 1 TO GET-NEXT-UMT.
# MOVE 1 TO GET-NEXT-DS.

#
# UMT-UPDATE-EXIT.
# EXIT.
# EJECT

```

Figure 11 (Part 11 of 13). Example program to refresh a replicated UMT

```

# *****
# * Exit from the program cleanly *
# *****
#   TRAN-FINISH SECTION.

#   * End the browse operation for the UMT
#     IF (UMT-STARTBR = 1)
#       EXEC CICS ENDBR FILE('UMTNAME')
#         RESP(RESPONSE)
#         RESP2(RESPONSE2)
#       END-EXEC
#     END-IF.

#   * End the browse operation for the dataset
#     IF (DS-STARTBR = 1)
#       EXEC CICS ENDBR FILE('SOURCEDS')
#         RESP(RESPONSE)
#         RESP2(RESPONSE2)
#       END-EXEC
#     END-IF

#   * Output a message to the screen if UMT was updated
#     IF (REM-FILE = 0)
#       EXEC CICS SEND TEXT
#         FROM(MESSAGE-OUTPUT)
#         ERASE
#         RESP(RESPONSE)
#         RESP2(RESPONSE2)
#       END-EXEC
#     ELSE
#   * Output a message if either file was defined as remote
#     EXEC CICS SEND TEXT
#       FROM(REMOTE-OUTPUT)
#       ERASE
#       RESP(RESPONSE)
#       RESP2(RESPONSE2)
#     END-EXEC
#   END-IF.

#   * End the program and return to CICS
#     EXEC CICS RETURN
#     END-EXEC.

#   TRAN-FINISH-EXIT.
#     EXIT.
#     EJECT

```

Figure 11 (Part 12 of 13). Example program to refresh a replicated UMT

```

# *****
# * Display error message on screen and exit program *
# *****
# PROCESS-ERROR SECTION.

# * Copy last return codes into the message
# MOVE RESPONSE TO ERROR-RESP.
# MOVE RESPONSE2 TO ERROR-RESP2.

# * Output message to the screen
# EXEC CICS SEND TEXT
# FROM(ERROR-OUTPUT)
# ERASE
# RESP(RESPONSE)
# RESP2(RESPONSE2)
# END-EXEC.

# * End the program and return to CICS
# EXEC CICS RETURN
# END-EXEC.

# PROCESS-ERROR-EXIT.
# EXIT.
#

```

*Figure 11 (Part 13 of 13). Example program to refresh a replicated UMT*



---

## Appendix A. Sample user exit programs

*This appendix contains Product-sensitive Programming Interface and Associated Guidance Information.*

This appendix describes, by means of samples of coding and data definition sequences, the conventions used in user exit programs that are used with SDT. These samples are intended only as general guidance and do not define a programming interface.

The exit programs are supplied with CICS, in the library SDFHSAMP.

Copybook DFHXDTDS defines the data tables user exit parameter list, which is used in each of the following samples. DFHXDTDS is shown in Figure 7 on page 44.

---

### Sample XDTRD exit program

```
                TITLE 'DFH$DTRD - Sample XDTRD Global User Exit Program'
*****
*                                                       *
*   MODULE NAME = DFH$DTRD                               *
*                                                       *
*   DESCRIPTIVE NAME = CICS/ESA Shared Data Tables Sample XDTRD Exit *
*                                                       *
*   FUNCTION =                                           *
*   The program selects records for inclusion in a data table. *
*                                                       *
* ----- *
*   NOTE that this program is only intended to DEMONSTRATE the use *
*   of the data tables user exit XDTRD, and to show the sort of *
*   information which can be obtained from the exit parameter list. *
*   IT SHOULD BE TAILORED BEFORE BEING USED IN A PRODUCTION ENVIRONMENT *
* ----- *
*                                                       *
*   This global user exit program will be invoked, if enabled, when *
*   a record which has been fetched from the source data set is about *
*   to be added to the data table. *
*                                                       *
*   The program can be used both on CICS systems which are using *
*   Shared Data Tables support and on ones which are not. *
*   It uses a flag that is passed via the data tables parameter list *
*   to determine whether the exit has been invoked by shared data *
*   tables support. *
*                                                       *
*   The purpose of the program is to demonstrate the use of the *
*   option to optimize the data table load by skipping over ranges of *
*   key values which are to be excluded from the table. This option *
```

*Figure 12 (Part 1 of 9). Sample XDTRD user exit program*

```

* is only allowed for shared data tables, but the program also      *
* illustrates that individual records can be rejected when the exit *
* is not invoked by the data table loading transaction, or when     *
* shared data tables support is not in use.                          *
*                                                                      *
* If the program has been invoked by shared data tables support     *
* then it checks whether the source data set name passed to it is  *
* the one defined by the constant EXITDSN. If so, and the exit has  *
* been invoked from the loading transaction, then the skip-during- *
* load option will be used to skip (not attempt to load) any      *
* records except those whose keys start with the two characters in  *
* in EXITKEY.                                                       *
*                                                                      *
* If the program has not been invoked by shared data tables, it    *
* uses the data table name rather than the source DSname to check  *
* whether this is the file from which records are to be rejected.  *
* If the table name matches the constant EXITFILE then only records *
* whose keys start with the two characters in EXITKEY will be      *
* accepted for inclusion in the table.                               *
*                                                                      *
* A number of the actions taken are for illustrative purposes only, *
* rather than being the recommended way in which to code an XDTRD *
* exit program - for example, the program demonstrates how the    *
* keylength passed to the exit can be used to avoid having to know *
* the keylength of the source data set, whereas in practice this   *
* might well be known; and the program chooses to reject any     *
* records which are not presented to it by the loading transaction, *
* whereas it would be more realistic to accept all records in the  *
* desired range of keys.                                           *
*                                                                      *
* It should also be noted that there are other useful things which *
* can be done with the XDTRD exit, such as amending the contents of *
* the records as they are loaded into a user-maintained data table. *
*                                                                      *
* The trace flag passed to the exit is set ON if File Control (FC) *
* level 1 tracing is enabled.                                       *
*                                                                      *
* NOTES :                                                            *
*   DEPENDENCIES = 370/XA                                           *
*   DFH$DTRD, or an exit program which is based on this           *
*   sample, must be defined on the CSD as a program                *
*   (with DATALOCATION(ANY)).                                       *
*   RESTRICTIONS =                                                 *
*   This program is designed to run on CICS/ESA 3.3 or later      *
*   release. It requires the DFHXDTDS copybook to be               *
*   available at assembly time.                                     *
*   REGISTER CONVENTIONS = see code                                 *
*   MODULE TYPE = Executable                                       *
*   PROCESSOR = Assembler                                          *
*   ATTRIBUTES = Read only, AMODE 31, RMODE ANY                    *
*   -----*

```

Figure 12 (Part 2 of 9). Sample XDTRD user exit program



```

*
* ENTRY POINT = DFH$DTRD
*
* PURPOSE =
*   Described above
*
* LINKAGE =
*   Called by the user exit handler
*
* INPUT =
*   Standard user exit parameter list DFHUEPAR,
*   addressed by R1 and containing a pointer to the
*   Data Tables parameter list
*
* OUTPUT =
*   Return code placed in R15
*   When skipping is requested, a skip-key is returned in an
*   area whose address is passed in the parameter list
*
* EXIT-NORMAL =
*   Return code in R15 can be
*   UERCDTAC = accept record (include it in the table)
*   UERCDTRJ = reject record (omit it from the table)
*   UERCDTOP = optimize load by skipping on to specified key
*             (only possible with shared data tables support)
*
* EXIT-ERROR =
*   None
*
*-----*
*
* EXTERNAL REFERENCES :
*   ROUTINES = None
*   DATA AREAS = None
*   CONTROL BLOCKS =
*     User Exit Parameter list for XDTRD: DFHUEPAR
*     Data Tables User Exit Parameter List: DT_UE_PLIST
*   GLOBAL VARIABLES = None
*
* TABLES = None
*
* MACROS =
*   DFHUEXIT to generate the standard user exit parameter list
*             with the extensions for the XDTRD exit point
*   DFHUEXIT to declare the XPI (exit programming interface)
*   DFHTRPTX XPI call to issue a user trace entry
*
*-----*

```

Figure 12 (Part 3 of 9). Sample XDTRD user exit program

```

*
* DESCRIPTION of the program structure:
*
* 1) Standard entry code for a global user exit that uses the XPI:
*     The program sets up any definitions required, then saves
*     the caller's registers, establishes addressability, and
*     addresses the parameter lists.
* 2) Initial section of code:
*     The program gets the key address and length from the data
*     table parameter list, then tests whether the exit was
*     invoked from shared data table code.  If not, it branches
*     to the non-SDT section of code.
* 3a) SDT code - Skipping:
*     If the source data set is the one from which records are to
*     be selected, and if the exit has been called by the data
*     table load, then the program compares the record key with
*     a value which defines the range to be included in the
*     data table, and sets return codes which will cause the
*     data table load to skip over any other keys.
* 3b) SDT code - Tracing:
*     If FC level 1 tracing is enabled, the program issues a user
*     trace point X'0128', then branches to set R15 and return.
* 4a) non-SDT code - choosing whether to accept record:
*     If the file name is the one for which records are to be
*     selected, then the program rejects the record if it does
*     not match the value which defines the range to be included
*     in the data table.
* 4b) non-SDT code - Tracing:
*     If FC level 1 tracing is enabled, the program issues a user
*     trace point X'0118'.
* 5) Set R15 and return:
*     The program sets a return code in R15 and performs
*     standard exit code for a global user exit (restores
*     caller's registers, and returns to the address that was in
*     R14 when the exit program was called).
*
*****
EJECT ,
DFHUEXIT TYPE=EP,ID=XDTRD   Standard UE parameters for XDTRD
DFHUEXIT TYPE=XPIENV       Exit programming interface (XPI)
EJECT ,
COPY DFHXDTDS               Additional data table UE params
EJECT ,
COPY DFHTRPTY               Trace definitions
EJECT ,

```

Figure 12 (Part 4 of 9). Sample XDTRD user exit program

```

*****
* REGISTER USAGE : *
* R0 - *
* R1 - address of DFHUEPAR on input, and used by XPI calls *
* R2 - address of standard user exit parameter list, DFHUEPAR *
* R3 - record length *
* R4 - address of data set name (SDT) or data table name (non-SDT) *
* R5 - address of storage for XPI parameters *
* R6 - address of data tables parameter list, DT_UE_PLIST *
* R7 - final return code to be set in R15 *
* R8 - address of the record key *
* R9 - key length *
* R10- address of the skip-key area (SDT only) *
* R11- base register *
* R12- address of data table flags byte, UEPDTFLG *
* R13- address of kernel stack prior to XPI CALLS *
* R14- used by XPI calls *
* R15- return code and used by XPI calls *
* (The register equates are declared by the DFHUEEXIT call above) *
*****
        SPACE 2
DFH$DTRD CSECT
DFH$DTRD AMODE 31
DFH$DTRD RMODE ANY
        STM  R14,R12,12(R13)      Save callers registers
        LR   R11,R15              Set up base register
        USING DFH$DTRD,R11
        LR   R2,R1                Address standard parameters
        USING DFHUEPAR,R2
        L    R6,UEPDTPL           Address data table parameters
        USING DT_UE_PLIST,R6
        L    R8,UEPDTKA           Key address
        L    R9,UEPDTKL           Key length
*****
* Test whether the exit was invoked from shared data tables support *
*****
        TM   UEPDTFLG,UEPDTSDT    Were we invoked from SDT?
        BZ   NOTSDT                Branch to non-SDT code if not
        EJECT ,
*****
* Invoked from SDT, so can use skip optimization *
*****
        L    R10,UEPDTSKA         Get skip-key area address
*****
* Determine whether the source data set is the one on which *
* optimization by skipping is to be performed. *
* If it is not, just accept all records. *
*****
        CLC  UEPDTDSN,EXITDSN
        BNE  SDTACC

```

Figure 12 (Part 5 of 9). Sample XDTRD user exit program

```

*****
* Only keys in the range defined by the initial two characters in      *
* EXITKEY are to be accepted, any other ranges of key values         *
* are to be skipped.                                                *
* First check whether skipping is valid - skipping can only be used  *
* when the call has been issued by the loading transaction (which is  *
* indicated by the UEPDPTOPT flag being set).                        *
*****
      TM   UEPDTFLG,UEPDPTOPT   Can we skip?
      BZ   SDTREJ               Just reject rec if not (although
                               in a production version it would
                               make more sense to check whether
                               the record key is in the desired
                               range, and accept it if so)
*****
* EXITKEY is currently set to 'AD', so that the effect of           *
* the exit will be:                                                *
* - If the key is less than 'AD....' then skip to a skip-key of    *
*   'AD' padded with 00s.                                          *
* - If it starts with 'AD' then accept it.                          *
* - If it is greater than 'AD' then skip to a skip-key of 'FF's   *
* If the value of the constant EXITKEY is altered, then the program *
* will cause the new range it defines to be selected for inclusion *
* in the table. Note that if a different length of EXITKEY is     *
* needed to define the range to be accepted, then the code will    *
* also require amendment, as it currently assumes a length of 2.  *
*****
      CLC  0(2,R8),EXITKEY     Is key below or above 'AD' ?
      BE  SDTACC               If equal then just accept
      BH  HIGHER               Above so skip to end of file
      SPACE 1
LOWER  DS  0H                  Skip forwards to 'AD...'
      MVC  0(2,R10),EXITKEY    Set skip-key value
      LR  R15,R9               Keylength for padding
      SH  R15,=H'3'           minus 2 for 'AD' and 1 for XC
      EX  R15,XCSKP           Clear out rest of skip key
      LA  R7,UERCDDTOP        Indicate skipping
      B   SDTTR
      SPACE 1
HIGHER DS  0H                  Skip on to end of file
      MVI 0(R10),X'FF'        Set 'FF' in start of skip key
      LR  R15,R9               Get length of skip-key
      BCTR R15,0              Decrement for pad length
      BCTR R15,0              Decrement for MVC
      EX  R15,MVCSKP          Propagate 'FF' through skip-key
      LA  R7,UERCDDTOP        Indicate skipping
      B   SDTTR
      SPACE 1

```

Figure 12 (Part 6 of 9). Sample XDTRD user exit program

```

*****
* Store return code in R7 for accept or reject *
*****
SDTACC  DS   0H
        LA   R7,UERC DTAC      Indicate accept
        B    SDTTR
SDTREJ  DS   0H
        LA   R7,UERC DTRJ      Indicate reject
        EJECT ,
*****
* Tracing when SDT support is in use *
*****
SDTTR   L    R15,UEPTRACE
        TM   0(R15),UEPTRON     Is trace on?
        BZ   NOSDTTR           No - do not issue trace then
        L    R5,UEPXSTOR        Prepare for XPI call
        USING DFHTRPT_ARG,R5
        L    R13,UEPSTACK
*****
* Trace source data set name, key, record length, flags, and skip-key *
* Some of these fields are only available with SDT support *
*****
        LA   R4,UEPD TSN        Point at data set name
        LA   R3,UEPDTRL         Address of record length for trace
        LA   R12,UEPD TFLG       Point at the data table flags
        DFHTRPTX CALL,
        CLEAR,
        IN,
        FUNCTION(TRACE_PUT),
        POINT_ID(RDTRACE2),
        DATA1((R4),UEPD TDSL),
        DATA2((R8),(R9)),
        DATA3((R3),4),
        DATA4((R12),1),
        DATA5((R10),(R9)),
        OUT,
        RESPONSE(*),
        REASON(*)
NOSDTTR DS   0H
        B    FINISH            Go and return from exit
        EJECT ,

```

Figure 12 (Part 7 of 9). Sample XDTRD user exit program

```

*****
*      Exit was NOT invoked by shared data tables support      *
*****
NOTSDT  DS   0H
*****
* Determine whether this data table is the one from which records *
* are to be rejected.  If it is not, just accept all records.   *
*****
          CLC   UEPDTNAM,EXITFILE
          BNE   ACC
          CLC   0(2,R8),EXITKEY      Does key start with 'AD' ?
          BE    ACC                  Yes, so accept it
REJ      DS   0H
          LA   R7,UERCDTRJ          Indicate reject
          B    TRACE                 Go and issue trace
ACC      DS   0H
          LA   R7,UERCDTAC          Indicate accept
          SPACE 2
*****
*      Tracing when SDT support is not in use                    *
*****
TRACE   L    R15,UEPTRACE
          TM   0(R15),UEPTRON       Is trace on?
          BZ   NOTRACE              No - do not issue trace then
          L    R5,UEPXSTOR          Prepare for XPI call
          USING DFHTRPT_ARG,R5
          L    R13,UEPSTACK
*****
* Trace data table name, key, record length, and flags          *
*****
          LA   R4,UEPDTNAM          Point at data table name
          LA   R3,UEPDTRL           Address of reclen for trace
          LA   R12,UEPDFTLG         Point at the data table flags
          DFHTRPTX CALL,
          CLEAR,
          IN,
          FUNCTION(TRACE_PUT),
          POINT_ID(RDTRACE1),
          DATA1((R4),8),
          DATA2((R8),(R9)),
          DATA3((R3),4),
          DATA4((R12),1),
          OUT,
          RESPONSE(*),
          REASON(*)

```

Figure 12 (Part 8 of 9). Sample XDTRD user exit program

```

NOTRACE DS 0H
EJECT ,
*****
* Code to set R15 return code and return control *
*****
FINISH DS 0H
LR R15,R7 Pick up exit return code
L R13,UEPEPSA Standard GLUE ending code
L R14,12(R13)
LM R0,R12,20(R13)
BR R14
SPACE 2
*****
* Constants and executed instructions *
*****
EXITDSN DC CL44'CFV23.CSYSW1.SOURCED'
EXITFILE DC CL8'CICD'
EXITKEY DC C'AD'
SPACE 1
RDTRACE1 DC XL2'118'
RDTRACE2 DC XL2'128'
SPACE 1
MVCSKP MVC 1(*-*,R10),0(R10) Executed to propagate X'FF's
XCCKP XC 2(*-*,R10),2(R10) Executed to clear to X'00's
SPACE 1
END DFH$DTRD

```

Figure 12 (Part 9 of 9). Sample XDTRD user exit program

---

## Sample XDTAD exit program

```
TITLE 'DFH$DTAD - Sample XDTAD Global User Exit Program'
*****
*
*   MODULE NAME = DFH$DTAD
*
*   DESCRIPTIVE NAME = CICS/ESA Shared Data Tables Sample XDTAD Exit
*
*   FUNCTION =
*   The program selects records for inclusion in a shared data table.
*
* -----
*   NOTE that this program is only intended to DEMONSTRATE the use
*   of the data tables user exit XDTAD, and to show the sort of
*   information which can be obtained from the exit parameter list.
*   IT SHOULD BE TAILORED BEFORE BEING USED IN A PRODUCTION ENVIRONMENT
* -----
*
*   This global user exit program will be invoked, if enabled, when
*   a WRITE request is issued to a data table.
*
*   The program can be used both on CICS systems which are using
*   Shared Data Tables support, and on ones which are not.
*   It uses a flag that is passed via the data tables parameter list
*   to determine whether the exit has been invoked by shared data
*   tables support.
*
*   The purpose of the program is to demonstrate the use of the XDTAD
*   global user exit to select only certain records for inclusion in
*   a data table. In this example, the selection is made on the
*   basis of key values. If shared data tables support is being
*   used, and the source data set for the data table is that
*   specified by the constant EXITDSN, then the program will select
*   particular keys for inclusion in the data table, and reject
*   others. For all other source data sets, or if shared data table
*   support is not in use, then all records will be accepted.
*
*   The record selection is made on the basis of the value of the 6th
*   character in the record key. This is for illustrative purposes
*   only, as it is unlikely to be the criterion for selection in a
*   realistic environment. For example, for a shared CICS-maintained
*   data table, it might be desirable to select a group of records
*   which are known to be very frequently read by applications
*   running in other CICS regions in the MVS system.
*
*   The trace flag passed to the exit is set ON if File Control (FC)
*   level 1 tracing is enabled.
*
*
*
```

Figure 13 (Part 1 of 6). Sample XDTAD user exit program



```

* NOTES :
*   DEPENDENCIES = 370/XA
*       DFH$DTAD, or an exit program which is based on this
*       sample, must be defined on the CSD as a program
*       (with DATALOCATION(ANY)).
*   RESTRICTIONS =
*       This program is designed to run on CICS/ESA 3.3 or later
*       release. It requires the DFHXDTDS copybook to be
*       available at assembly time.
*   REGISTER CONVENTIONS = see code
*   MODULE TYPE = Executable
*   PROCESSOR = Assembler
*   ATTRIBUTES = Read only, AMODE 31, RMODE ANY
*
*-----*
*
*   ENTRY POINT = DFH$DTAD
*
*   PURPOSE =
*       Described above
*
*   LINKAGE =
*       Called by the user exit handler
*
*   INPUT =
*       Standard user exit parameter list DFHUEPAR,
*       addressed by R1 and containing a pointer to the
*       Data Tables parameter list
*
*   OUTPUT =
*       Return code placed in R15
*
*   EXIT-NORMAL =
*       Return code in R15 can be
*       UERCDTAC = accept record (include it in the table)
*       UERCDTRJ = reject record (omit it from the table)
*
*   EXIT-ERROR =
*       None
*
*-----*
*
*   EXTERNAL REFERENCES :
*       ROUTINES = None
*       DATA AREAS = None
*       CONTROL BLOCKS =
*           User Exit Parameter list for XDTAD: DFHUEPAR
*           Data Tables User Exit Parameter List: DT_UE_PLIST
*       GLOBAL VARIABLES = None
*
*   TABLES = None
*

```

Figure 13 (Part 2 of 6). Sample XDTAD user exit program

```

* MACROS =
*     DFHUEXIT to generate the standard user exit parameter list
*           with the extensions for the XDTAD exit point
*     DFHUEXIT to declare the XPI (exit programming interface)
*     DFHTRPTX XPI call to issue a user trace entry
*
*-----*
*
* DESCRIPTION of the program structure:
*
* 1) Standard entry code for a global user exit that uses the XPI:
*     The program sets up any definitions required, then saves
*     the caller's registers, establishes addressability, and
*     addresses the parameter lists.
*
* 2) Initial section of code:
*     The program gets the key address and length from the data
*     table parameter list, then tests whether the exit was
*     invoked from shared data table code.  If not, it branches
*     to the non-SDT section of code.
*
* 3a) SDT code - Tracing:
*     If FC level 1 tracing is enabled, the program issues a user
*     trace point X'0129', then branches to choose whether to
*     accept the record for inclusion in the table.
*
* 3b) non-SDT code - Tracing:
*     If FC level 1 tracing is enabled, the program issues a user
*     trace point X'0119', then accepts the record, and exits.
*
* 4) SDT code - choosing whether to accept record:
*     If the source data set is the one from which records are to
*     be selected, then, if the key contains a numeric character
*     in the sixth byte, the program sets the return code so that
*     the record will be accepted.  If the key contains an
*     alphabetic character in the sixth byte then the program
*     sets a return code to reject the record.
*
* 5) Standard exit code for a global user exit that uses the XPI:
*     Restore users registers, and return to the address that was
*     in R14 when the exit program was called.
*
*****
EJECT ,
DFHUEXIT TYPE=EP,ID=XDTAD Standard UE parameters for XDTAD
DFHUEXIT TYPE=XPIENV Exit programming interface (XPI)
EJECT ,
COPY DFHXDTDS Additional data table UE params
EJECT ,
COPY DFHTRPTY Trace definitions
EJECT ,

```

Figure 13 (Part 3 of 6). Sample XDTAD user exit program

```

*****
* REGISTER USAGE : *
* R0 - *
* R1 - address of DFHUEPAR on input, and used by XPI calls *
* R2 - address of standard user exit parameter list, DFHUEPAR *
* R3 - *
* R4 - address of source data set name *
* R5 - address of storage for XPI parameters *
* R6 - address of data tables parameter list, DT_UE_PLIST *
* R7 - address of the trace flag *
* R8 - address of the record key *
* R9 - key length *
* R10- address of the data table name *
* R11- base register *
* R12- address of data table flags byte, UEPDTFLG *
* R13- address of kernel stack prior to XPI CALLS *
* R14- used by XPI calls *
* R15- return code, and used by XPI calls *
* (The register equates are declared by the DFHUEXIT call above) *
*****
        SPACE 2
DFH$DTAD CSECT
DFH$DTAD AMODE 31
DFH$DTAD RMODE ANY
        STM  R14,R12,12(R13)      Save callers registers
        LR   R11,R15              Set up base register
        USING DFH$DTAD,R11
        LR   R2,R1                Address standard parameters
        USING DFHUEPAR,R2
        L    R6,UEPDTPL           Address data table parameters
        USING DT_UE_PLIST,R6
*****
* Save some fields from the parameter list that are to be traced or *
* used in selecting records *
*****
        L    R8,UEPDTKA           Key address
        L    R9,UEPDTKL           Key length
        LA   R10,UEPDTNAM         Data table name
*****
* Test whether the exit was invoked from shared data tables support *
*****
        TM   UEPDTFLG,UEPDTSDT    Were we invoked from SDT?
        BZ   NOTSDT               Branch to non-SDT code if not
        EJECT ,
*****
* Invoked from SDT, so can use SDT fields in parameter list. *
* Issue trace, then check data set name, and select records. *
*****
        L    R7,UEPTRACE          Address of trace flag
        TM   0(R7),UEPTRON        Is trace on?
        BZ   NOSDTTR              No - do not issue trace then
        L    R5,UEPXSTOR          Prepare for XPI call
        USING DFHTRPT_ARG,R5
        L    R13,UEPSTACK

```

Figure 13 (Part 4 of 6). Sample XDTAD user exit program

```

*****
* Trace key, data table name, source data set name, and flags. *
* The last two fields are only meaningful for SDT support. *
*****
      LA   R4,UEPDTDSN          Point at the source data set name
      LA   R12,UEPDTFLG        Point at the data table flags
      DFHTRPTX CALL,
      CLEAR,
      IN,
      FUNCTION(TRACE_PUT),
      POINT_ID(ADTRACE2),
      DATA1((R8),(R9)),
      DATA2((R10),8),
      DATA3((R4),UEPDTDSL),
      DATA4((R12),1),
      OUT,
      RESPONSE(*),
      REASON(*)
NOSDTR  B   CHOOSE              Go and choose whether to accept rec
      EJECT ,
*****
* Exit has not been invoked from SDT *
* Issue trace then accept the record. *
*****
NOTSDT  L   R7,UEPTRACE        Address of trace flag
      TM   0(R7),UEPTRON       Is trace on?
      BZ   NOTRACE             No - do not issue trace then
      L    R5,UEPXSTOR         Prepare for XPI call
      USING DFHTRPT_ARG,R5
      L    R13,UEPSTACK
*****
* Trace key and data table name *
*****
      DFHTRPTX CALL,
      CLEAR,
      IN,
      FUNCTION(TRACE_PUT),
      POINT_ID(ADTRACE1),
      DATA1((R8),(R9)),
      DATA2((R10),8),
      OUT,
      RESPONSE(*),
      REASON(*)
      SPACE 1

```

Figure 13 (Part 5 of 6). Sample XDTAD user exit program

```

NOTRACE DS 0H
        B ACC          Go and accept the record
        EJECT ,
*****
*       Is this the data set from which records are to be selected? *
*       If not, just accept record and end.                         *
*****
CHOOSE  DS 0H
        CLC UEPDTSN,EXITDSN
        BNE ACC
*****
*
* If the sixth character in the key is numeric then accept the
* record, if it is alphabetic then reject the record.
* This assumes that numerics have EBCDIC value >= F0
* and that alphabetics have value < F0
*
*****
        CLI 5(R8),X'F0'      Is sixth character >= F0 ?
        BL REJ              If no, then go and reject record
ACC     LA R15,UERC DTAC    If yes, set RC to ACCEPT
        B GLUEND            and end
REJ     LA R15,UERC DTRJ    Set RC to REJECT
        SPACE 3
GLUEND  DS 0H              Standard GLUE ending code
        L R13,UEPEPSA
        L R14,12(R13)
        LM R0,R12,20(R13)
        BR R14
        SPACE 2
*****
*       Constants
*****
EXITDSN DC CL44'CFV23.CSYSW1.SOURCED'
        SPACE 1
ADTRACE1 DC XL2'119'
ADTRACE2 DC XL2'129'
        SPACE 2
        END DFH$DTAD

```

Figure 13 (Part 6 of 6). Sample XDTAD user exit program

---

## Sample XDTLC exit program

```
                TITLE 'DFH$DTLC - Sample XDTLC Global User Exit Program'
*****
*
*   MODULE NAME = DFH$DTLC
*
*   DESCRIPTIVE NAME = CICS/ESA Shared Data Tables Sample XDTLC Exit
*
*   FUNCTION =
*   The program rejects a data table if its load did not complete OK.
*
*   -----
*   NOTE that this program is only intended to DEMONSTRATE the use
*   of the data tables user exit XDTLC, and to show the sort of
*   information which can be obtained from the exit parameter list.
*   IT SHOULD BE TAILORED BEFORE BEING USED IN A PRODUCTION ENVIRONMENT
*   -----
*
*   This global user exit program will be invoked, if enabled, when
*   the load of a data table has completed.
*
*   The program can be used both on CICS systems which are using
*   Shared Data Tables support, and on ones which are not.
*   It uses a flag that is passed via the data tables parameter list
*   to determine whether the exit has been invoked by shared data
*   tables support.
*
*   The program will issue a user trace entry if tracing is enabled,
*   then check the setting of the load completion indicator.
*   If this shows that loading failed to complete successfully, then
*   the exit program will set a return code that rejects the table
*   by requesting that it be closed.
*
*   The trace flag passed to the exit is set ON if File Control (FC)
*   level 1 tracing is enabled.
*
*   NOTES :
*   DEPENDENCIES = 370/XA
*   DFH$DTLC, or an exit program which is based on this
*   sample, must be defined on the CSD as a program
*   (with DATALOCATION(ANY)).
*   RESTRICTIONS =
*   This program is designed to run on CICS/ESA 3.3 or later
*   release. It requires the DFHXDTDS copybook to be
*   available at assembly time.
*   REGISTER CONVENTIONS = see code
*   MODULE TYPE = Executable
*   PROCESSOR = Assembler
*   ATTRIBUTES = Read only, AMODE 31, RMODE ANY
*
*   -----
```

Figure 14 (Part 1 of 5). Sample XDTLC user exit program

```

*
* ENTRY POINT = DFH$DTLC
*
* PURPOSE =
*   Described above
*
* LINKAGE =
*   Called by the user exit handler
*
* INPUT =
*   Standard user exit parameter list DFHUEPAR,
*   addressed by R1 and containing a pointer to the
*   Data Tables parameter list
*
* OUTPUT =
*   Return code placed in R15
*
* EXIT-NORMAL =
*   Return code in R15 can be
*   UERCDTOK = accept table
*   UERCDTCL = reject table (close it)
*
* EXIT-ERROR =
*   None
*
*-----*
*
* EXTERNAL REFERENCES :
*   ROUTINES = None
*   DATA AREAS = None
*   CONTROL BLOCKS =
*     User Exit Parameter list for XD TLC: DFHUEPAR
*     Data Tables User Exit Parameter List: DT_UE_PLIST
*   GLOBAL VARIABLES = None
*
* TABLES = None
*
* MACROS =
*   DFHUEEXIT to generate the standard user exit parameter list
*     with the extensions for the XD TLC exit point
*   DFHUEEXIT to declare the XPI (exit programming interface)
*   DFHTRPTX XPI call to issue a user trace entry
*
*-----*
*
* DESCRIPTION of the program structure:
*
* 1) Standard entry code for a global user exit that uses the XPI:
*   The program sets up any definitions required, then saves
*   the caller's registers, establishes addressability, and
*   addresses the parameter lists.

```

Figure 14 (Part 2 of 5). Sample XD TLC user exit program

```

* 2) Tracing (only executed if FC level 1 tracing is enabled): *
*   The program tests whether it was invoked from shared data *
*   table code. If so, it issues a user trace point X'0126' *
*   including fields from the data table parameter list which *
*   are only supplied by shared data table support. If not, it *
*   issues a X'0116' trace point, containing parameters which *
*   are supplied by any level of data table support. *
* 3) Choosing whether to accept the table: *
*   The program tests the return code from the load. If the *
*   load failed to complete, then it sets UERC DTCL in R15, *
*   which requests that the table should be closed. If the *
*   load completed successfully, then it sets UERC DTOK in R15 *
*   to keep the table open. *
* 4) Standard exit code for a global user exit that uses the XPI: *
*   Restore users registers, and return to the address that was *
*   in R14 when the exit program was called. *
* *
*****
EJECT ,
DFHUEXIT TYPE=EP,ID=XDTLC standard UE parameters for XDTLC
DFHUEXIT TYPE=XPIENV exit programming interface (XPI)
EJECT ,
COPY DFHXDTDS Additional data table UE params
EJECT ,
COPY DFHTRPTY Trace definitions
EJECT ,
*****
* Register usage : *
* R0 - *
* R1 - address of DFHUEPAR on input, and used by XPI calls *
* R2 - address of standard user exit plist, DFHUEPAR *
* R3 - *
* R4 - address of source data set name *
* R5 - address of storage for XPI parameters *
* R6 - address of data tables parameter list, DT_UE_PLIST *
* R7 - address of the trace flag, UEPTRACE *
* R8 - address of data table name *
* R9 - address of loading completion indicator, UEPDTORC *
* R10- *
* R11- base register *
* R12- address of data table flags byte, UEPDTFLG *
* R13- address of kernel stack prior to XPI calls *
* R14- used by XPI calls *
* R15- return code, and used by XPI calls *
* (The register equates are declared by the DFHUEXIT call above) *
*****

```

Figure 14 (Part 3 of 5). Sample XDTLC user exit program



```

SPACE 2
DFH$DTLC CSECT
DFH$DTLC AMODE 31
DFH$DTLC RMODE ANY
STM R14,R12,12(R13)    Save caller's registers
LR R11,R15             Establish base
USING DFH$DTLC,R11
LR R2,R1               Address standard parameters
USING DFHUEPAR,R2
L R6,UEPDTPL           Address data table parameters
USING DT_UE_PLIST,R6
SPACE 1
LA R8,UEPDTNAM         Address data table name
LA R9,UEPDTORC         Address load return code
*****
* Issue Trace (if tracing is enabled) *
*****
L R7,UEPTRACE          Get trace flag address
TM 0(R7),UEPTRON       Is trace on?
BZ CHOOSE              Skip tracing if not
*****
* Test whether the exit was invoked from shared data tables support *
*****
TM UEPDTFLG,UEPDTSDT  Were we invoked from SDT?
BZ NOTSDT              Branch if not
EJECT ,
*****
* Exit has been invoked from SDT *
*****
L R5,UEPXSTOR          Set up XPI trace call
USING DFHTRPT_ARG,R5
L R13,UEPSTACK
*****
* Trace data table name, load return code, source dsname, and flags. *
* The last two fields are only meaningful for SDT support. *
*****
LA R4,UEPDTDSN         Get source data set name
LA R12,UEPDTFLG        Get data table flags
DFHTRPTX CALL,
CLEAR,
IN,
FUNCTION(TRACE_PUT),
POINT_ID(LCTRACE2),
DATA1((R8),8),
DATA2((R9),1),
DATA3((R4),UEPDTDSL),
DATA4((R12),1),
OUT,
RESPONSE(*),
REASON(*)
B CHOOSE              Go and test if load completed OK
EJECT ,

```

Figure 14 (Part 4 of 5). Sample XD TLC user exit program

```

*****
*      Exit has not been invoked from SDT      *
*****
NOTSDT  L    R5,UEPXSTOR      Set up XPI trace call
        USING DFHRPT_ARG,R5
        L    R13,UEPSTACK
*****
* Trace data table name and load return code *
*****
        DFHRPTX CALL,
            CLEAR,
            IN,
            FUNCTION(TRACE_PUT),
            POINT_ID(LCTRACE1),
            DATA1((R8),8),
            DATA2((R9),1),
            OUT,
            RESPONSE(*),
            REASON(*)
        EJECT ,
*****
*      If load completed successfully, then keep table open *
*      If not, ask for it to be closed *
*****
CHOOSE  DS    0H
        CLI   0(R9),UEPDTLFL   Did load fail?
        BNE  LOADOK
        LA   R15,UERC DTCL     Set RC for table to be closed
        B    GLUEND
LOADOK  LA   R15,UERC DTOK     Set RC to keep table
        SPACE 3
GLUEND  DS    0H              Standard GLUE exit code
        L    R13,UEPEPSA
        L    R14,12(R13)
        LM   R0,R12,20(R13)
        BR   R14
        SPACE 2
*****
* Constant Declarations *
*****
LCTRACE1 DC  XL2'116'
LCTRACE2 DC  XL2'126'
        SPACE 1
        END  DFH$DTLC

```

Figure 14 (Part 5 of 5). Sample XD TLC user exit program

---

# Index

## A

abend codes  
  AFCH 30, 31  
  AFCZ 67  
activation of user exits 60  
AFCH abend code 30, 31  
AFCZ abend code 67  
alternate indexes 3, 9  
AOR (application-owning region)  
  CONNECT operation 6  
  definition 2  
application programming  
  extensions for SDT 1  
  for a CMT  
    description 27  
    overview 9  
  for a UMT  
    description 28  
    overview 11  
automatic journaling 10, 12

## B

benefits  
  of data tables 4, 13  
BIND security 23  
browse requests  
  comparison with function shipping 31  
  comparison with VSAM 33  
  definition xi  
  for a CMT 27  
  for a UMT 28

## C

CEDA DEFINE FILE command  
  description 35  
  example for CMT 38  
  example for UMT 39  
  LOG parameter 37  
  MAXNUMRECS parameter 36  
  OPENTIME parameter 37  
  RECORDFORMAT parameter 37  
  TABLE parameter 36  
CEMT  
  INQUIRE command 40, 41  
  SET command 39, 40  
CICS-maintained data table  
  browse requests 27  
  data integrity 10  
  definition of 35  
  description 9

CICS-maintained data table (*continued*)  
  journaling 10  
  overview 2  
  performance 13  
  read requests 27, 28  
  update requests 28  
  use during loading 28  
closing a data table 31, 50  
communication  
  between CICS and user exits 43  
CONNECT  
  by AOR 6, 30  
  security checking 23  
cross-memory services  
  advantages 1  
  analyzing errors 70  
  commands supported 27  
  comparison with function shipping 4, 31  
  use by application 29  
CSFU transaction 49, 50  
CSSY transaction 49  
customization  
  user exits 43

## D

daisy chaining 30  
data integrity  
  of a CMT 10  
  of a UMT 12  
data space  
  dump of contents 71  
  use by data tables 3, 15  
data tables  
  application programming 27  
  availability 1  
  benefits 4  
  CICS load modules required 25  
  closing 31, 50  
  comparison with VSAM 32  
  concepts 1  
  customization 43  
  disabling 31  
  dump information 70  
  enhancements 1  
  on different releases of CICS 23  
  opening 49  
  operations 49  
  planning 13  
  problem determination 63  
  read requests 1, 27, 28  
  resource definition 35

- data tables (*continued*)
  - selecting files 16
  - sharing 2
  - statistics information 51
  - trace information 63
  - update requests 1, 28, 29
  - use of data space 3
- delete requests
  - comparison with VSAM 33
- DFH\$DTAD sample program 102
- DFH\$DTLC sample program 108
- DFH\$DTRD sample program 93
- DFHDTCV 24
- DFHDTSVC 24
- DFHFCT macro 35
- DFHMVRMS 24
- DFHXDTS copybook 43
- disabling a data table 31
- disconnection
  - of AOR and data table 7, 30
- DSECT
  - for user exit parameter list 43
- dump information for data tables 70
- dynamic transaction backout 12, 37

## E

- EIBRESP2 field 28, 32
- enabling of user exits 60
- exec interface
  - user exits 43

## F

- file
  - used as a data table 1, 16
- file control
  - commands
    - overview for CMT 9, 27
    - overview for UMT 11, 28
    - supported by cross-memory services 27
    - user exits 43
- file management
  - using cross-memory services 4
  - using function shipping 4
- file security 23
- FOR (file-owning region)
  - definition 2
  - LOGON operation 5
- function
  - for trace points 64

## G

- gap 16, 27, 31
  - definition xi

## I

- imprecise keys
  - definition xi
- initial state of data table
  - defining by CEDA 37
- INQUIRE FILE command
  - description 40, 41
  - MAXNUMRECS parameter 40, 41
  - TABLE parameter 40, 41
- installation
  - MVS considerations 23
- installation parameter list 8
- INSTLN parameter 8
- integrity
  - of CMT data 10
  - of UMT data 12
- interface
  - for user exits 43
  - product-sensitive programming 43, 93
- INVREQ condition 28

## J

- journaling 10

## K

- key length
  - comparison with function shipping 32
- KSDS (key-sequenced data set)
  - used as source data set 1, 3
  - with a UMT 12

## L

- load modules
  - required for data tables 25
- loading
  - use of CMT during 28
  - use of UMT during 29
- LOADING condition 29
- local file
  - definition 2
- LOGON
  - by FOR 5
  - security check 23

## M

- messages
  - at end of loading 49
  - at start of loading 49
- multiple files
  - with same source data set 9
- MVS considerations 23, 51

## N

NOSPACE condition 29  
NOTFND condition 28, 29  
notification  
    for CONNECT operations 7

## O

opening a data table 49  
operations for data tables 49

## P

parameter list  
    for user exits 43  
performance  
    benefits of data tables 4, 13  
    of a CMT 13  
    of a UMT 14  
planning  
    for use of data tables 13  
precise keys  
    definition xi  
problem determination for data tables 63  
product-sensitive programming interface 43, 93

## Q

qualifier flags  
    for trace points 64

## R

RACF  
    used as security manager 8  
read requests  
    comparison with function shipping 32  
    comparison with VSAM 32  
    for a CMT 27, 28  
    for a UMT 28  
reason codes  
    in trace points 65  
recovery of data tables  
    defining by CEDA 37  
    during emergency restart 50  
refreshing replicated UMTs 74  
remote file  
    definition 2  
requester  
    definition 2  
resource definition  
    CEDA DEFINE FILE command 35  
    description 35  
    overview for a CMT 9  
    overview for a UMT 11

response codes  
    in trace points 65

## S

SAF (system authorization facility)  
    used for security checking 8  
SDT  
    overview 1  
    replacing existing services 1  
    security 7  
SDT support  
    planning for 23  
security checking  
    at AOR connect 23  
    at FOR logon 23  
    comparison with function shipping 8, 31  
    for data tables 7, 23  
    RACF considerations 8  
    use of SAF 8  
selecting files  
    for use as data tables 16  
server  
    definition 2  
SET FILE command  
    description 39, 40  
    MAXNUMRECS parameter 40  
    TABLE parameter 39, 40  
shareoption, VSAM 10  
sharing  
    CONNECT operation 6  
    LOGON operation 5  
    SDT operations 4, 5  
size of data table  
    defining by CEDA 36  
    defining by SET command 40  
    finding by INQUIRE command 40, 41  
source data set  
    for data tables 1  
    independent of UMT 11  
    must be KSDS 3  
    used with CMT 9  
    with multiple files 9  
statistics  
    additional fields 59  
    samples for data tables 53  
    to evaluate data tables 51  
    to select data tables 18  
storage use  
    description 15  
SUPPRESSED condition 29  
SVC errors 67  
SYSID parameter 29  
sysplex environment  
    example program code 79  
    example program operation 76

sysplex environment (*continued*)  
  example program set up and execution 76  
  purpose of example program 76  
  refreshing replicated UMTs 74  
  using shared data tables in 73  
system dump information 70

## T

trace information  
  entry and exit points 63  
  exception points 66  
  for data tables 63  
  function and qualifier flags 64  
  reason codes 65  
  response codes 65  
transient data queues  
  used for messages 49  
type of data table  
  defining by CEDA 36  
  defining by SET command 39, 40  
  finding by INQUIRE command 40, 41

## U

update requests  
  definition xi  
  for a CMT 28  
  for a UMT 29  
user exits  
  activating 60  
  at end of loading 47  
  communication with CICS 43  
  definition 60  
  description 43  
  DSECT for parameter list 43  
  during loading 46  
  enabling 60  
  exit program samples 93  
  for exec interface 31, 43  
  for file control 31, 43  
  overview 3  
  parameter list 43  
  when adding records 46  
  XDTAD exit 46  
  XDTLC exit 47  
  XDTRD exit 46  
user-maintained data table  
  browse requests 28  
  data integrity 12  
  definition of 35  
  description 11  
  journaling 12  
  overview 2  
  performance 14  
  read requests 28

user-maintained data table (*continued*)  
  replication in a sysplex 73  
  update requests 29  
  use during loading 29

## V

VSAM  
  access method control block 49, 50  
  alternate indexes 3, 9  
  base cluster 9  
  comparison with data tables 32  
  shareoption 10

## X

XDTAD user exit  
  description 46  
  exit program sample 102  
XDTLC user exit  
  description 47  
  exit program sample 108  
XDTRD user exit  
  description 46  
  exit program sample 93





Program Number: 5655-018



Printed in the United States of America  
on recycled paper containing 10%  
recovered post-consumer fiber.

SC33-1186-01





*Spine information:*



CICS/ESA

Shared Data Tables Guide

*Version 4 Release 1*