

Cincom

SUPRA SERVER PDM

DBA Utilities User's Guide
(OS/390 & VSE)

P26-6260-63



SUPRA[®] Server PDM DBA Utilities User's Guide (OS/390 & VSE)

Publication Number P26-6260-63

© 1983–1988, 1991, 1992, 1994, 1998, 2000, 2002 Cincom Systems, Inc.
All rights reserved

This document contains unpublished, confidential, and proprietary information of Cincom. No disclosure or use of any portion of the contents of these materials may be made without the express written consent of Cincom.

The following are trademarks, registered trademarks, or service marks of Cincom Systems, Inc.:

AD/Advantage [®]	iD CinDoc [™]	MANTIS [®]
C+A-RE [™]	iD CinDoc Web [™]	Socrates [®]
CINCOM [®]	iD Consulting [™]	Socrates [®] XML
Cincom Encompass [®]	iD Correspondence [™]	SPECTRA [™]
Cincom Smalltalk [™]	iD Correspondence Express [™]	SUPRA [®]
Cincom SupportWeb [®]	iD Environment [™]	SUPRA [®] Server
CINCOM SYSTEMS [®]	iD Solutions [™]	Visual Smalltalk [®]
 gOOj [™]	intelligent Document Solutions [™]	VisualWorks [®]
	Intermax [™]	

UniSQL[™] is a trademark of UniSQL, Inc.
ObjectStudio[®] is a registered trademark of CinMark Systems, Inc.

All other trademarks are trademarks or registered trademarks of their respective companies.

Cincom Systems, Inc.
55 Merchant Street
Cincinnati, Ohio 45246-3732
U.S.A.

PHONE: (513) 612-2300
FAX: (513) 612-2000
WORLD WIDE WEB: <http://www.cincom.com>

Attention:

Some Cincom products, programs, or services referred to in this publication may not be available in all countries in which Cincom does business. Additionally, some Cincom products, programs, or services may not be available for all operating systems or all product releases. Please see your Cincom representative to be certain the items are available to you.

Release information for this manual

The *SUPRA Server PDM DBA Utilities User's Guide (OS/390 & VSE)*, P26-6260-63, is dated January 15, 2002. This document supports Release 2.7 of SUPRA Server PDM in IBM mainframe environments.

We welcome your comments

We encourage critiques concerning the technical content and organization of this manual. Please take the [survey](#) provided with the online documentation at your convenience.

Cincom Technical Support for SUPRA Server PDM

FAX: (513) 612-2000
Attn: SUPRA Server Support

E-mail: helpna@cincom.com

Phone: 1-800-727-3525

Mail: Cincom Systems, Inc.
Attn: SUPRA Server Support
55 Merchant Street
Cincinnati, OH 45246-3732
U.S.A.



Contents

About this book	xi
Using this document.....	xi
Document organization	xi
Revisions to this manual	xiii
Conventions	xiv
SUPRA Server documentation series	xvi
Using the DBA utilities	19
DBA utilities overview	19
Utility functions and applications	21
Non-UCL utilities and applications	22
Executing functions that require UCL.....	23
Using the hierarchical structure of UCL	24
Formatting UCL.....	25
Coding null arguments	26
Coding arguments.....	27
Validating programs	28
Executing utilities that do not require UCL	29
Executing utilities with different types of files	30
Running debug and trace for DBA utilities	31
Using the DEBUG function.....	32
Using the XTRACE function.....	32
Executing the functions	33
Defining files.....	33
Defining files for functions that use UCL in OS/390	33
Defining files for functions that use UCL in VSE	36
Defining files for the Execution Statistics utility	38
File definitions for Execution Statistics in OS/390.....	38
File definitions for Execution Statistics in VSE.....	39
Defining files for the PDM Termination utility	40

Choosing run-time options.....	42
Defining the amount of storage	43
Coding the run-time interface parameters	44
Setting the STACK parameter	46
Inserting exit programs into functions.....	49
Using sort programs	52
Allocating sort memory.....	53
Allocating sort work space.....	54
Handling errors in sort programs.....	56
Coding the control section	57
Coding the UCL for the control section.....	58
Determining control statements for functions	86
Coding the Format function	87
Format function syntax	88
Coding the Sorted-Populate function	91
Coding the UCL for the Sorted-Populate function	92
Writing exit programs.....	97
Selecting exit points.....	97
Summary of data parameters and valid actions	99
Requesting statistics.....	102
Coding the Depopulate function	105
Coding the UCL for the Depopulate function	106
Writing exit programs.....	112
Selecting exit points.....	112
Summary of data parameters and valid actions	114
Requesting statistics.....	117
Coding the Reorganize function	119
Coding the UCL for the Reorganize function	119
Writing exit programs.....	124
Selecting exit points.....	125
Summary of data parameters and valid actions	126
Requesting statistics.....	129

Coding the File Statistics function	131
Coding the UCL for the File Statistics function.....	132
Programming examples	136
Requesting file statistics.....	137
Requesting Basic File Information (BASE)	138
Requesting Current File Size (SIZE).....	139
Requesting Linkpath Statistics (LINK).....	140
Requesting Chain Statistics (CHAIN).....	141
Requesting Record Code Statistics (CODE).....	146
 Coding the Expand function	 147
 Coding the Version 1 Unload and Load functions	 153
Coding the UCL for the Unload function	155
Coding the UCL for the Load function.....	169
Writing exit programs	181
Modifying the data record.....	182
Deleting the current data record.....	182
Adding a new data record	183
Retaining the format of the data file	184
Formatting the run control record.....	185
Formatting the pre-header record	186
Formatting the file header record	187
Formatting the data records	188
Formatting the file trailer record	190
Examples of Unload, Load, and Modify functions	191
Example 1—Unloading and loading all of the Burry's database files.....	200
Example 2—Unloading, changing and loading files.....	213
 Coding the Version 2 Unload, Load, and Insert Linkpath functions	 225
Version 2 overview	225
What to do with linkpaths when you unload and load	228
Coding the CSIPARM file for Unload, Load, and Insert Linkpath.....	230
Coding CSIPARM file and run control statements for PDM files	230
Coding CSIPARM file and run control statements for directory files.....	231
Coding JCL for Unload, Load, and Insert Linkpath functions.....	232
Files you define in OS/390 JCL.....	233
Files you define in VSE JCL.....	234

- Unloading PDM files 235
 - Defining files 235
 - Coding run control statements 242
 - Coding file control statements 254
- Unloading Directory files 264
- Using exit points 265
 - Using exit point 10 267
 - Using exit point 20 279
 - Using exit point 30 286
- Loading PDM files 292
 - Defining files 292
 - Coding run control statements for the Load function 302
 - Coding the file control statements for the Load function 317
- Loading Directory files 322
- Coding the Insert Linkpath function 322
 - Defining files 325
 - Coding control statements 328
- Examples of Unload, Load, and Insert Linkpath functions 333
 - Internal schema of Burry's database 335
 - Internal schema of files before unloading 338
 - Internal schema of files after unloading 340
 - Unloading and loading all of Burry's database files 342
 - Unloading, changing, and loading files 352

Coding the Print function 363

- Coding the UCL for the Print function 363
- Writing exit programs 376
- Print examples 377

Coding the Modify function 381

- Coding the Modify function 381
- Coding the UCL for the Modify function 382
- Writing exit programs 393
- Modify examples 394

Coding the PDM Termination utility 399

- Coding the PDM Termination utility 399
- Coding the input statements for the PDM Termination utility 400
- PDM termination example 402

Coding the Execution Statistics utility for release 2.1.6	403
Coding the Execution Statistics utility for release 2.1.6.....	403
Defining the files.....	404
Arrangement of the statistics report	405
Sample of the Physical Data Manager identification page	406
Sample of the group identification page.....	406
Sample of the system statistics page.....	407
Sample of the file statistics page.....	411
Sample of the file statistics totals for group	413
Sample of the termination page	416
Coding the Execution Statistics utility for release 2.4	417
Coding the Execution Statistics utility for release 2.4.....	417
Defining the files.....	418
Arrangement of the statistics report	419
Sample of the Physical Data Manager identification page	420
Sample of the group identification page.....	420
Sample of the system statistics page.....	421
Sample of the file statistics page.....	426
Sample of the file statistics totals for group	429
Sample of the termination page	432
Coding the Inter-Directory Copy utility	433
Coding the Inter-Directory Copy utility.....	433
Coding the input statements for the Inter-Directory Copy utility	435
Optional input statements	437
Signon input statement	438
Copy table input statement.....	439
Edit mask input statement.....	439
User input statement.....	440
Security group input statement	441
Schema input statement	442
Conceptual schema input statement.....	443
Domain input statement	444
Executing the Inter-Directory Copy utility	445
OS/390	445
VSE	445
Inter-Directory Copy example.....	446

Coding the Recover, Restore, and Log-Print utilities	447
Coding the Recover, Restore, and Log-Print functions	447
Coding the UCL for the Recover and Restore functions	449
Coding the UCL for the Log-Print function.....	456
Writing exit programs.....	461
Selecting exit points	462
Valid actions	469
Examples	471
Recover example.....	471
Restore example	475
Log-Print example.....	481
Coding the Review function	485
Coding the Review function	485
Coding the UCL for the Review function.....	486
Review example	487
Coding the Unlock function	489
Coding the Unlock function.....	489
Index	493

About this book

Using this document

This manual tells how to code utilities to help control and monitor a SUPRA database. It is written for database administrators familiar with the SUPRA Physical Data Manager and Directory concepts.

The first chapter of this manual explains functions available to you. For information about using the utilities with Series 80 and SUPRA 1.x files, refer to the *SUPRA Server PDM Migration Guide (OS/390 & VSE)*, P26-0550.

Document organization

The information in this manual is organized as follows:

Chapter 1—Using the DBA utilities

Describes the utilities available to organize and maintain data and perform a broad range of functions.

Chapter 2—Executing the functions

Describes what is required to execute functions and provides special considerations.

Chapter 3—Coding the control section

Describes how to code a control section, which defines the processing environment for the functions you want to perform.

Chapter 4—Coding the Format function

Describes how to code the format function to format database files.

Chapter 5—Coding the Sorted-Populate function

Describes how to code the Sorted Populate function, a secondary key function to create the secondary key structure on large files.

Chapter 6—Coding the Depopulate function

Describes how to code the Depopulate function to delete secondary keys.

Chapter 7—Coding the Reorganize function

Describes how to code the Reorganize function to correct deterioration of a secondary key structure.

Chapter 8—Coding the File Statistics function

Describes how to code the File Statistics function to get reports on various physical and logical characteristics of database files.

Chapter 9—Coding the Expand function

Describes how to code the Expand function to enlarge database files that are too full for acceptable performance.

Chapter 10—Coding the Version 1 Unload and Load functions

Describes how to code the Version 1 Unload function to extract records from a database file and write them to a sequential output file. Also describes the Load function, which copies records from sequential files to database files.

Chapter 11—Coding the Version 2 Unload, Load, and Insert Linkpath functions

Describes how to code the Version 2 Unload, Load, and Insert Linkpath functions if performance of Unload and Load functions is critical or you are reloading the files in a SUPRA converted or Series 80 format.

Chapter 12—Coding the Print function

Describes how to code the Print function to print records from a database file.

Chapter 13—Coding the Modify function

Describes how to code the Modify function to update records in database files.

Chapter 14—Coding the PDM Termination utility

Describes how to code the PDM Termination utility, used to shut down the PDM by executing a single function.

Chapter 15—Coding the Execution Statistics utility for release 2.1.6

Describes how to code the Execution Statistics utility (CSUXSTAT) to generate a statistics report in release 2.1.6.

Chapter 16—Coding the Execution Statistics Utility for release 2.4

Describes how to code the Execution Statistics utility (CSUXSTAT) to generate a statistics report in release 2.4.

Chapter 17—Coding the Inter-Directory Copy utility

Describes how to code the Inter-Directory Copy utility to copy information from one SUPRA directory to another.

Chapter 18—Coding the Recover, Restore, and Log-Print functions

Describes how to code the Recover, Restore, and Log Print functions to back off updates to the last commit, reapply updates when files are lost or damaged, or print selected information from the System Log File without updating the database.

Chapter 19—Coding the Review function

Describes how to code the Review function to determine whether database files are locked.

Chapter 20—Coding the Unlock function

Describes how to code the Unlock function to examine each field in the lock record of files you specify, show incorrect fields, and update lock records to indicate the file is locked.

Index**Revisions to this manual**

The following changes have been made for this release:

- ◆ Revised the Considerations for the **SECONDARY-KEY parameter** on page 95.
- ◆ Added a Consideration for the **PRESERVE clause** on page 256.
- ◆ Revised **Element List illustrations** on pages 262 and 320.
- ◆ Added a consideration for the **Element List statement** beginning on page 318.
- ◆ Revised information in the section “**Defining the LINKWK02/LNKWRK2 file**” on page 327.
- ◆ Revised the output filename for the Inter-Directory Copy utility to LTRX, under “**VSE**” on page 445.

Conventions

The following table describes the conventions used in this document series:

Convention	Description	Example
Constant width type	Represents screen images and segments of code.	<pre>PUT 'customer.dat' GET 'miller\customer.dat' PUT '\DEV\RMT0'</pre>
Slashed b (<i>b</i>)	Indicates a space (blank). The example indicates that four spaces appear between the keywords.	<pre>BEGNbbbbSERIAL</pre>
Brackets []	Indicate optional selection of parameters. (Do not attempt to enter brackets or to stack parameters.) Brackets indicate one of the following situations: A single item enclosed by brackets indicates that the item is optional and can be omitted. The example indicates that you can optionally enter a WHERE clause.	<pre>[WHERE <i>search-condition</i>]</pre>
	Stacked items enclosed by brackets represent optional alternatives, one of which can be selected. The example indicates that you can optionally enter either WAIT or NOWAIT. (WAIT is underlined to signify that it is the default.)	<pre><u>(WAIT)</u> (NOWAIT)</pre>
Braces { }	Indicate selection of parameters. (Do not attempt to enter braces or to stack parameters.) Braces surrounding stacked items represent alternatives, one of which you must select. The example indicates that you must enter ON or OFF when using the MONITOR statement.	<pre>MONITOR {ON OFF}</pre>

Convention	Description	Example
Underlining (In syntax)	Indicates the default value supplied when you omit a parameter. The example indicates that if you do not choose a parameter, the system defaults to WAIT.	[WAIT] [NOWAIT]
	Underlining also indicates an allowable abbreviation or the shortest truncation allowed. The example indicates that you can enter either STAT or STATISTICS.	<u>STATISTICS</u>
Ellipsis points...	Indicate that the preceding item can be repeated. The example indicates that you can enter multiple host variables and associated indicator variables.	INTO : <i>host-variable</i> [: <i>ind-variable</i>],...
UPPERCASE lowercase	In most operating environments, keywords are not case-sensitive, and they are represented in uppercase. You can enter them in either uppercase or lowercase.	COPY MY_DATA.SEQ HOLD_DATA.SEQ
<i>Italics</i>	Indicate variables you replace with a value, a column name, a file name, and so on. In this example, you must substitute the name of a table.	FROM <i>table-name</i>
Punctuation marks	Indicate required syntax that you must code exactly as presented. () parentheses . period , comma : colon ' ' single quotation marks	(<i>user-id</i> , <i>password</i> , <i>db-name</i>) INFILE 'Cust.Memo' CONTROL LEN4
SMALL CAPS	Represent a required keystroke. Multiple keystrokes are hyphenated.	ALT-TAB
OS/390 VSE	Information specific to a certain operating system is flagged by a symbol in a shadowed box (OS/390) indicating which operating system is being discussed. Skip any information that does not pertain to your environment.	OS/390 See the SUPRA Server procedure library member TIS\$RDM for a list of RDM procedures. VSE See the SUPRA Server RDM sublibrary member TXJ\$INDX for a list of JCL.

SUPRA Server documentation series

SUPRA Server is the advanced relational database management system for high-volume, update-oriented production processing. A number of tools are available with SUPRA Server including Directory Maintenance, DBA utilities, DBAID, SPECTRA, and MANTIS. The following list shows the manuals and tools used to fulfill the data management and retrieval requirements for various tasks. Some of these tools are optional. Therefore, you may not have all the manuals listed. For a brief synopsis of each manual, refer to the *SUPRA Server Digest (OS/390 & VSE)*, P26-9062.

Overview

- ◆ *SUPRA Server Digest (OS/390 & VSE)*, P26-9062

Getting started

- ◆ *SUPRA Server PDM Migration Guide (OS/390 & VSE)*, P26-0550*
- ◆ *SUPRA Server PDM CICS Connector Systems Programming Guide (OS/390 & VSE)*, P26-7452

General use

- ◆ *SUPRA Server PDM Glossary*, P26-0675
- ◆ *SUPRA Server PDM Messages and Codes Reference Manual (RDM/PDM Support for OS/390 & VSE)*, P26-0126

Database administration tasks

- ◆ *SUPRA Server PDM and Directory Administration Guide (OS/390 & VSE)*, P26-2250
- ◆ *SUPRA Server PDM Directory Online User's Guide (OS/390 & VSE)*, P26-1260
- ◆ *SUPRA Server PDM Directory Batch User's Guide (OS/390 & VSE)*, P26-1261
- ◆ *SUPRA Server PDM DBA Utilities User's Guide (OS/390 & VSE)*, P26-6260
- ◆ *SUPRA Server PDM Logging and Recovery (OS/390 & VSE)*, P26-2223
- ◆ *SUPRA Server PDM Tuning Guide (OS/390 & VSE)*, P26-0225
- ◆ *SUPRA Server PDM RDM Administration Guide (OS/390 & VSE)*, P26-8220
- ◆ *SUPRA Server PDM RDM PDM Support Supplement (OS/390 & VSE)*, P26-8221
- ◆ *SUPRA Server PDM RDM VSAM Support Supplement (OS/390 & VSE)*, P26-8222
- ◆ *SUPRA Server PDM Migration Guide (OS/390 & VSE)*, P26-0550*
- ◆ *SUPRA Server PDM Windows Client Support User's Guide*, P26-7500*
- ◆ *SPECTRA Administrator's Guide*, P26-9220

Application programming tasks

- ◆ *SUPRA Server PDM DML Programming Guide (OS/390 & VSE)*, P26-4340
- ◆ *SUPRA Server PDM RDM COBOL Programming Guide (OS/390 & VSE)*, P26-8330
- ◆ *SUPRA Server PDM RDM PL/1 Programming Guide (OS/390 & VSE)*, P26-8331
- ◆ *SUPRA Server PDM Migration Guide (OS/390 & VSE)*, P26-0550*
- ◆ *SUPRA Server PDM Windows Client Support User's Guide*, P26-7500*

Report tasks

- ◆ *SPECTRA User's Guide*, P26-9561



Manuals marked with an asterisk (*) are listed more than once because you use them for multiple tasks.



Educational material is available from your regional Cincom education department.

1

Using the DBA utilities

You can use the DBA utilities to organize and maintain data, and perform a broad range of functions.

DBA utilities overview

Of the 14 DBA utilities, you are likely to begin with the **Format** function to format new files. You can use the **Load** function to add data to the new primary and related files. You can also use the **Load** function to both format and add data to files in one step.

After you load the files, you can use the **Sorted-Populate** function to take information from them to create secondary keys for the index files. You can maintain the index files with the **Depopulate** and **Reorganize** functions. The **Depopulate** function deletes secondary keys, and the **Reorganize** function corrects the deterioration of the secondary key tree structure that can occur with updates. With the **Reorganize** function, you can rebuild a tree structure without accessing its primary or related file.

Once you have created your files, you can monitor their growth and access time with the **File Statistics** and the **Execution Statistics Utility** functions. When the files are too full for acceptable performance, you can use the **Expand** function to enlarge the related files. The **Unload and Load** functions can also be used to enlarge primary and related files. In addition, the **Load** function repairs damaged linkpaths and arranges the files so the PDM can access them more efficiently.

While you can use the Load and Unload functions to format and enlarge files, the main purpose of these functions is to repair and reorganize files. Depending on the format in which you want the files loaded, you can choose one of two different versions of the Unload and Load functions. The **Version 1 Load and Unload** functions automatically change any file's format to that of SUPRA native files.

With the **Version 2 Load and Unload functions**, you can leave files in their original format or change them to any other format. In addition, the Version 2 functions repair damaged linkpaths and run much faster than the Version 1 functions.

The **Version 2 Insert Linkpath function** adds linkpath data to primary files without your reloading them. The Load function also updates the linkpath fields with correct data.

For routine maintenance, you can use the **Print** and **Modify** functions. With the Print function, you can see the records in database files. With the Modify function, you can update records and blank out linkpaths. This enables you to use the **Version 1 Unload and Load functions** to unload and load related files, without unloading and loading their associated primary files.

You execute all the functions listed in “**Utility functions and applications**” on page 21 from a common driver module. To use any of these utilities, code a Utilities Command Language (UCL) program. In the UCL program, you can code any number of different functions.

OS/390

You submit the UCL program by executing the CSUOUTIL load module in OS/390 and VSE. In OS/390, you can also use the general utility cataloged procedure TISUTUTL. For more information, refer to the **SUPRA Server PDM and Directory Administration Guide (OS/390 & VSE)**, P26-2250.

Utility functions and applications

The following table lists utility functions and their applications:

Utility	Applications
Format	Formats a database file. The Format function sets all records in the file to blanks and writes a file control record on the file. Use this function only with SUPRA native files.
Sorted-Populate	Puts secondary key information from primary and related files into index files.
Depopulate	Deletes secondary keys.
Reorganize	Corrects the deterioration of the tree structure that can result from updating the database.
File Statistics	Displays the physical and logical characteristics of database files. Use this function only with SUPRA native files.
Expand	Expands the capacity of an existing related file. Use this function only with SUPRA native files.
Version 1 Unload	Extracts records from a database file and writes them to a sequential output medium. Use this function only with SUPRA native files. You can use the Version 1 Load and Unload functions to convert Series 80 or SUPRA converted files to the SUPRA native format. However, these functions cannot convert SUPRA native files to Series 80 or SUPRA converted format.
Version 1 Load	Formats database files and write data records from a sequential medium to the files. Use this function only with SUPRA native files. You can use the Version 1 Load and Unload functions to convert Series 80 or SUPRA converted files to the SUPRA native format. However, these functions cannot convert SUPRA native files to Series 80 or SUPRA converted format.
Print	Prints records from a database file. Use this function with SUPRA native, SUPRA converted, and Series 80 files.
Modify	Updates records in a database file. Use this function with SUPRA native, SUPRA converted, and Series 80 files.
Recover	Backs off updates to the database after an abend.
Restore	Reapplies updates to the database after an abend.
Log-Print	Prints selected information from the System Log File without updating the database.
Review	Determines whether database files are locked.
Unlock	Unlocks database files after an abend.

Non-UCL utilities and applications

You do not use UCL with the utilities listed in the following table:

Utility	Applications
Version 2 Unload	Extracts records from Series 80, SUPRA converted, SUPRA native, or SUPRA Directory files at high speed and writes them to a sequential output medium appropriate for the Version 2 Load utility.
Version 2 Load	Formats Series 80, SUPRA converted, or SUPRA native database files and writes data records at high speed from a sequential medium (the output from a Version 2 Unload function) to the files.
Version 2 Insert Linkpath	Inserts linkpath data into Series 80, SUPRA converted, SUPRA native, or SUPRA Directory files without reloading primary files. The input for this utility must be the output of the Version 2 Load utility.
PDM Termination	Shuts down the PDM.
Execution Statistics	Generates system statistics.
Inter-Directory Copy	Copies information from one SUPRA Directory to another.

Executing functions that require UCL

To program the Utility functions in “[Utility functions and applications](#)” on page 21, you must use UCL. UCL is a procedural language that uses statements to identify the functions to be performed. The CONTROL statement initiates the control section, which establishes the processing environment for the Utility functions.

The FUNCTION statement initiates a function such as load or unload. You can code the FUNCTION statement more than once, and you can combine many functions in a single UCL program. However, some combinations of functions are not recommended. These combinations are described in the appropriate chapters. The following example shows how to organize a simple UCL program:

```
CONTROL (BEGIN)
    supporting control statements
FUNCTION (name)
    supporting function statements
FUNCTION (name)
    supporting function statements
CONTROL (END)
```

Using the hierarchical structure of UCL

UCL has a hierarchical structure. After each CONTROL or FUNCTION statement, you code a number of subordinate statements. For example, in the following UCL program, LIST is subordinate to CONTROL, and FILE is subordinate to FUNCTION. This manual shows the subordinate statements by indenting them.

```
CONTROL (BEGIN)
  LIST (name)
  FUNCTION (name)
    FILE (name)
  CONTROL (END)
```

To code a subordinate statement, you must code all of its superordinate statements. For example, you must code a FUNCTION statement before you code a FILE statement. This restriction also applies when subordinate statements have subordinates. In the following UCL example, the CONTROL statement has a subordinate statement, SORT, which has a subordinate statement, MEMORY. You must code the SORT statement before you code the MEMORY statement.

```
CONTROL (BEGIN)
  SORT ( )
    MEMORY (500K)
  FUNCTION (SORTED-POPULATE)
    STATISTICS (ALL)
    FILE (PTMF)
      SECONDARY-KEY (PTMFSK01,PTMFSK02)
      LOAD-DENSITY (80)
      SECONDARY-KEY (PTMFSK03)
      LOAD-DENSITY (90)
    FILE (RANV)
  CONTROL (END)
```

Formatting UCL

UCL has a free-form format that allows almost any coding style. Even though this manual lists only one statement per line and indents to show subordination, you do not need to follow this structure. For example, you can code the following example this way:

```
COMMAND (argument)
  STATEMENT (argument)
    STATEMENT (argument)
      STATEMENT (argument)
```

Or this way:

```
COMMAND (argument)
STATEMENT (argument) STATEMENT (argument) STATEMENT (argument)
```

When you put statements on the same line, you can put any number of spaces between them, or you can leave out spaces. You can also insert comments in your UCL program. You can provide any information you wish about your program in these comments. To show that information is a comment, put an asterisk in column 1. You must place your comments after CONTROL (BEGIN) and before CONTROL (END) as in the following example:

```
CONTROL (BEGIN)
**
* THIS UCL PROGRAM WILL FORMAT
* ALL USER FILES.
**
  ENV-DESC (MYDESC)
    SCHEMA (MYSHEMA)
      FUNCTION (FORMAT)
* CUSTOMER FILE      *
    FILE (CUST)
* PURCHASE ORDER FILE*
    FILE (PORD)
CONTROL (END)
```

Coding null arguments

Default values are provided for many statements so that you need not code all statements. However, whenever you code a subordinate statement to specify an option, you must also code the subordinate statement's superordinate statement even if you code it with a null argument. To code a null argument, put open and close parentheses after the statement. You do not need to include a space between them. The following is an example of a statement with a null argument:

```
LIST ( )  
  HEADER (YES)
```

To code the HEADER statement, you must first code the LIST statement. If you code the HEADER statement without coding the LIST statement, you receive an error message.

Coding arguments

When you code a statement name and an argument, you must enclose the argument in parentheses. You can use spaces or not as you like. For example, the following statement formats are valid:

```
STATEMENT(argument)
STATEMENT (argument)
STATEMENT ( argument )
STATEMENT ()
STATEMENT ( )
STATEMENT()
```

You code an argument differently depending on whether it is one of a set or one of a list of items. In the following example, the format shows a number of options in a set:

```
SUMMARY-DATA ( [ALL][FILE][NONE][FUNCTION][CUMULATIVE] )
```

When the format shows an option is in a set, you can select any number of options from the set. When you code the options, separate them with one or more spaces. For example, if you select FILE and FUNCTION, you can code them like this:

```
SUMMARY-DATA (FILE FUNCTION)
```

If the format shows the argument can be a list, however, it looks like the following example:

```
ELEMENT { { ALL
            }
           element - list }
```

If you code a list of elements instead of ALL, separate the elements with commas, as shown in the following example:

```
ELEMENT(ELEMENT1,ELEMENT2,ELEMENT3)
```

You can also separate the items in a list with any number of spaces, as shown in the following examples:

```
ELEMENT (ELEMENT1, ELEMENT2, ELEMENT3)
ELEMENT (ELEMENT1,  ELEMENT2,  ELEMENT3)
```

Do not embed spaces within an item. For example, ELEMENT1 and ELEMENT 1 do not mean the same thing.

Validating programs

The program listing follows the opening message CSUL0101I: COMMENCING COMMAND VALIDATION. The listing is an image of your input program.

Errors are identified by an error flag (**ERROR**), error pointer (@), and a three-digit number. These three indicators appear immediately after each line in which an error occurs. The following annotated listing identifies errors in lines 18 and 21:

Line number	Line text	Indicator
1	CONTROL (BEGIN)	<i>Valid Command</i>
2	ENV-DESC (MYDESC)	<i>Valid Command</i>
	.	
17	FUNCTION (PRINT)	<i>Valid Command</i>
18	FILE CUST	<i>Invalid Command</i>
ERROR	@001	<i>Inserted Error Information</i> <i>Error Number</i> <i>Error Pointer</i>
19	RECORD (ALL)	<i>Valid Command</i>
20	ELEMENT (ALL)	<i>Valid Command</i>
21	FILE (ABC#)	<i>Invalid Command</i>
ERROR	@015	<i>Inserted Error Information</i> <i>Error Number</i> <i>Error Pointer</i>
	.	
44	CONTROL (END)	<i>Valid Command</i>

The error flag appears in the line number column in the left margin. The pointer identifies the error's exact location in the line. The three-digit number specifies the error's cause or condition. Only one error is reported in each line. For example, in line 18, the number 001 indicates the open parenthesis is missing. The pointer is immediately below the file called CUST where the open parenthesis should be. Since only one error is reported in each line, nothing indicates that the close parenthesis is also missing.



If a UCL programming error occurs, validation continues flagging errors to the end of the program. No function processing is done until you correct all errors and rerun the job.

Executing utilities that do not require UCL

The utilities listed below do not use UCL. See the applicable chapters for the input statements for these utilities:

- ◆ Version 2 Load function (see “Coding the Version 2 Unload, Load, and Insert Linkpath functions” on page 225)
- ◆ Version 2 Unload function (see “Coding the Version 2 Unload, Load, and Insert Linkpath functions” on page 225)
- ◆ Version 2 Insert Linkpath function (see “Coding the Version 2 Unload, Load, and Insert Linkpath functions” on page 225)
- ◆ PDM Termination (see “Coding the PDM Termination utility” on page 399)
- ◆ Execution Statistics utility (see “Coding the Execution Statistics utility for release 2.1.6” on page 403 or “Coding the Execution Statistics utility for release 2.4” on page 417)
- ◆ Inter-Directory Copy (see “Coding the Inter-Directory Copy utility” on page 433)

You execute each of these utilities as separate programs.

Executing utilities with different types of files

You can execute all the utilities with SUPRA native files. However, you cannot execute all of them with Series 80 or SUPRA converted files.

If you use the following utilities with a Series 80 or SUPRA converted file, the results are unpredictable, and you may damage the file:

- ◆ Expand
- ◆ Modify

If you use the following utilities with a Series 80 or SUPRA converted file, the results are unpredictable, but you do not damage the file:

- ◆ File Statistics
- ◆ Print

If you use these utilities with a Series 80 file, the following results:

- ◆ **Format**—creates an empty SUPRA native file (not Series 80 or SUPRA converted file).
- ◆ **Version 1 Load**—creates SUPRA native file. Pre-SUPRA PDMs cannot use the file, and the Unload function cannot convert it back to pre-SUPRA PDM format. To process Series 80 or SUPRA converted files, use the Version 2 Unload and Load functions.

You can run the following utilities with a Series 80 file or a SUPRA converted file:

- ◆ **Version 1 Unload**—the Version 1 Load utility creates only SUPRA native files.
- ◆ Version 2 Load
- ◆ Version 2 Unload
- ◆ Version 2 Insert Linkpath

Running debug and trace for DBA utilities

Debug and trace facilities are available for DBA Utilities. On occasion, you may need debug output to help your technical service center resolve a usage or production problem.

Use Utility Control Language statements to activate debugging and tracing.

To run debug or trace in OS/390 environments, add this statement to your JCL:

```
//OUTPUT DD DSN =*
```

To run debug or trace in VSE environments, add these statements to your JCL for printed output or for disk output:

For printer:

```
//ASSGN OUTPUT,SYSLST
```

For disk:

```
//DLBL OUTPUT,'xxxx',,SD  
//EXTENT SYSnnn,xxxx
```

These statements define the output file for the output produced by debug or trace.

Using the DEBUG function

The DEBUG function causes the utilities to print out debugging information during utility execution. DEBUG produces a substantial amount of output. However, the amount of debugging support and the meaning of the DEBUG options is not consistent across the utilities functions.

The DEBUG parameter must be part of the CONTROL section of a UCL program. Code the parameter as follows:

```
DEBUG ( {  
        ALL  
        DML  
        FUNCTION  
        TRACE  
    } )
```

- ◆ ALL returns all types of debugging information.
- ◆ DML returns a listing of Physical View DML CALL parameters.
- ◆ FUNCTION returns the activities of all function processing.
- ◆ TRACE returns logical calls.

Using the XTRACE function

The XTRACE function enables tracing in the utilities' parser. Each modification of the parsing stack triggers a listing of the stack and other relevant information. This information provides a history of the parsing stack including pushing and popping of tokens according to the grammar rules. You must understand compilers and parsers in order to understand the output.

Code the XTRACE parameter as follows:

```
XTRACE ( {  
        ON  
        OFF  
    } )
```

You may code XTRACE anywhere in a UCL program. You may enable the trace function for the entire UCL program or trace only a few statements in the program.

2

Executing the functions

When you execute functions that require UCL, you must define files and code run-time options. In addition, some functions sort and some functions have exit points where you can insert exit programs. In all cases, there are special considerations.

Defining files

To execute the functions, you must define files for libraries, input, output, work, Directory, and PDM. For the functions that use UCL, you can use the file definitions in [“Defining files for functions that use UCL”](#) on page 33.

To define files for functions that do not use UCL, see [“Coding the Version 2 Unload, Load, and Insert Linkpath functions”](#) on page 225 for the Version 2 Unload, Load, and Insert Linkpath functions, [“Defining files for the Execution Statistics utility”](#) on page 38 for the Execution Statistics utility, and [“Defining files for the PDM Termination utility”](#) on page 40 for the PDM Termination utility.

Defining files for functions that use UCL in OS/390

You define different files in OS/390 and VSE. Differences are noted where they occur.

OS/390

To execute functions in OS/390, you must define the files listed in [“File definitions in OS/390”](#) on page 34. The figure in [“Files you define for functions that do not sort in OS/390”](#) on page 35 illustrates the configuration of the files needed for functions that do not sort. The figure in [“Files you define for functions that sort in OS/390”](#) on page 35 shows the additional files needed for functions that sort.

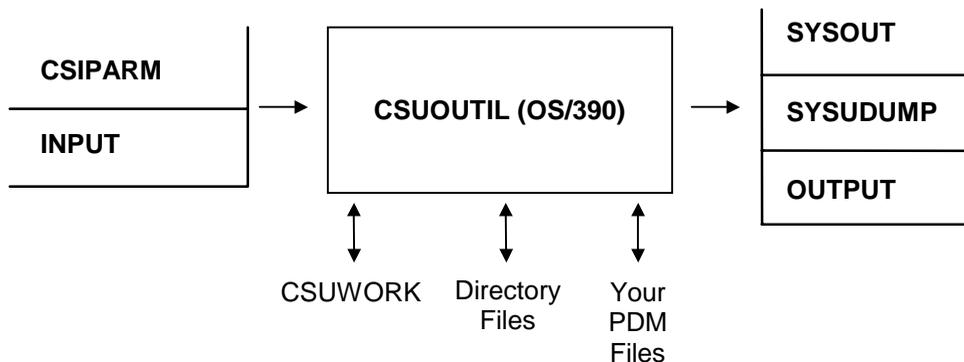
In OS/390, you can use the file definitions in the cataloged procedure TISUTUTL. You do not need to read further unless you want more details about defining files.

File definitions in OS/390

Type of file	Name of file	Use of file
Libraries in OS/390	STEPLIB	In this data set concatenation, you declare the libraries where the system looks for the function's load module, the single-task PDM load module, and any exit programs you code.
	SORTLIB	In this data set concatenation, you declare the libraries where the system looks for the sort program.
Input	INPUT	You code the UCL in this data set.
	CSIPARM	In this data set and in the SCHEMA and ENV-DESC statements in the UCL control section, you code the input to the single-task PDM.
Output Data Sets	OUTPUT	You define this data set to hold the output that the functions create.
	SYSOUT	You define this data set to hold output that the operating system creates.
	SYSUDUMP	You define this data set to hold output from a dump, if you request one.
Work Files	CSUWORK	You define this data set for the functions to use as a work file as they interpret and execute the UCL.
	CSI#WK0n	(where $n=1, 2, 3,$ and $4.$) You define these data sets for sort work space. You need these data sets only for functions that sort.
Other Files	Directory	You define these files for all functions except Log-Print.
	PDM	You define these files only when the function needs them. For example, if you are executing the File Statistics utility against only the Directory files, you do not need to define.

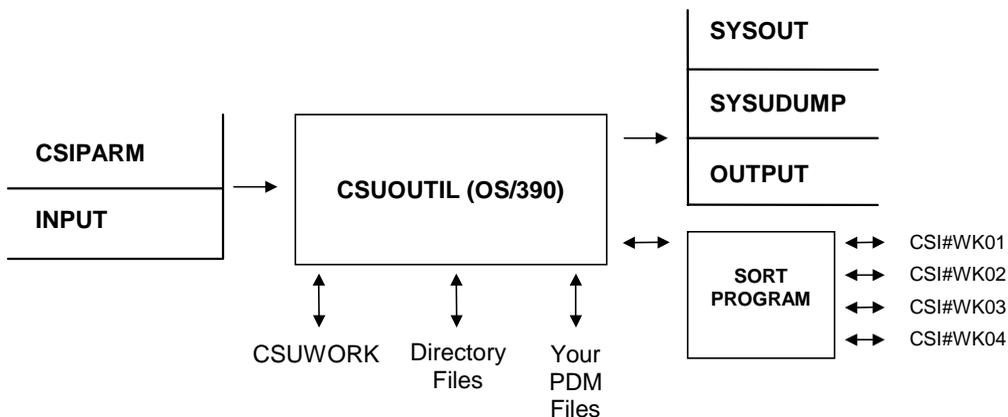
Files you define for functions that do not sort in OS/390

The following figure illustrates the configuration of the files needed for functions that do not sort.



Files you define for functions that sort in OS/390

The following figure shows the additional files needed for functions that sort.



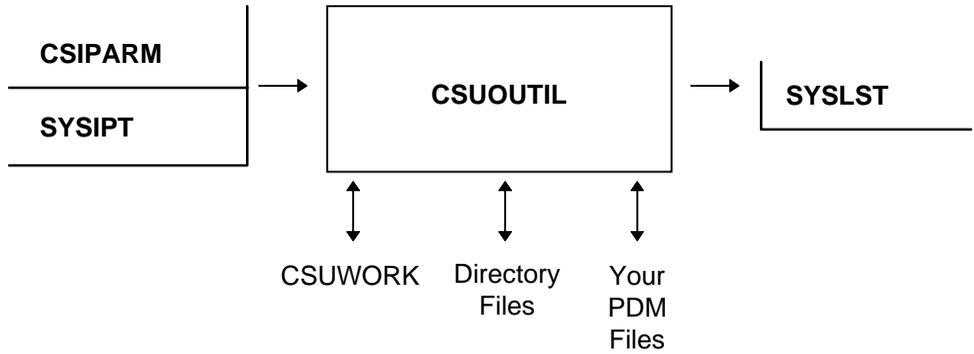
Defining files for functions that use UCL in VSE

VSE To execute functions in VSE, you must define the files listed in the following table. “Files you define for functions that do not sort in VSE” on page 37 illustrates the files you must define for functions that do not sort; “Files you define for functions that sort in VSE” on page 37 shows the files for functions that sort.

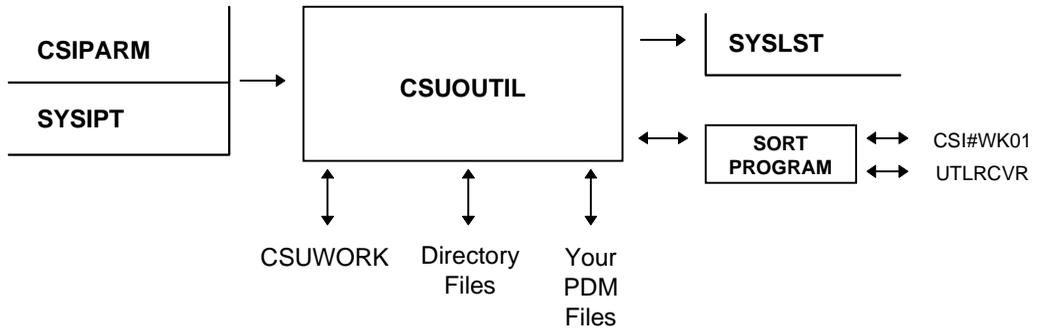
File definitions in VSE

Type of file	Name of file	Use of file
Libraries	LIBDEF	In the library definition search chain, you declare the libraries and sublibraries where the system looks for the function's program, the PDM program, the sort program, and any exit programs you code.
Input	SYSIPT	In this system logical unit, you code the UCL.
	CSIPARM	In this data set and in the SCHEMA and ENV-DESC statements in the UCL control section, you code the input to the single-task PDM. (You do not need this file for the Log-Print function.)
Output	SYSLST	You define this system logical unit for the output from the functions, the operating system, and any dump you request.
Work Files	CSUWORK	You define this data set for the functions to use as a work file as they interpret and execute the UCL.
	CSI#WK1	You define this data set for sort work space. You need this data set only for functions that sort.
Other Files	Directory	You define these files for all functions.
	PDM	You define these files only when the function needs them. For example, if you run File Statistics with only the Directory files, you do not need to define PDM files. Define your PDM files as direct access (does not apply to VSAM). In the Format, Version 1 Load, and Expand functions, also define the same files as sequential access and prefix their names with a Z. For example, DLBL PQRSTUV becomes DLBL ZPQRSTU.

Files you define for functions that do not sort in VSE



Files you define for functions that sort in VSE



Defining files for the Execution Statistics utility

The Execution Statistics utility does not require UCL. The files you define for this utility are not quite the same as for the functions that require UCL.

The following table lists the files you define in OS/390. “[File definitions for Execution Statistics in VSE](#)” on page 39 lists the files you define for VSE. “[Files you define for Execution Statistics in OS/390](#)” on page 38 shows the configuration of the OS/390 files. “[Files you define for Execution Statistics in VSE](#)” on page 39 shows the configuration of the VSE files.

In OS/390 and VSE, see the sample JCL member TXJPSTAT. You do not need to read further unless you want more details about defining files.

File definitions for Execution Statistics in OS/390

Type of file	Name of file	Use of file
Libraries	STEPLIB	In this data set concatenation, you declare in OS/390 the libraries where the system looks for the function's load module, the single-task PDM load module, and any exit programs you code.
Input Data Sets	INPUT	In this data set, you code the record size and block size of the STATS file.
	STATS	You define this data set to hold execution statistics records from the PDM.
Output Data Sets	OUTPUT	You define this data set to hold the output from the function.
	SYSUDUMP	You define this data set to hold output from a dump, if you request one.

Files you define for Execution Statistics in OS/390

The Execution Statistics utility uses no work, Directory, or PDM files.



File definitions for Execution Statistics in VSE

In VSE, you define the files listed below. The figure following the table shows their configuration.

Type of file	Name of file	Use of file
Libraries	LIBDEF	In the library definition search chain, you declare the libraries and sublibraries where the system looks for the function's program, the PDM program, the sort program, and any exit programs you code.
Input	SYSIPT	In this system logical unit, you code the record size and block size of the STATS file.
	STATS	You define this data set to hold execution statistics records from the PDM.
Output	SYSLST	You define this system logical unit for the output from the functions, the operating system, and any dump you request.

Files you define for Execution Statistics in VSE

The Execution Statistics utility uses no work, Directory, or PDM files.



Defining files for the PDM Termination utility

The PDM Termination utility does not require UCL. The files you define for this utility are not quite the same as for the functions that require UCL.

The following table lists the files you define in OS/390. “[File definitions for PDM Termination in VSE](#)” on page 41 lists the files you define for VSE. “[Files you define for PDM Termination in OS/390](#)” on page 41 shows the configuration of the OS/390 files. “[Files you define for PDM Termination in VSE](#)” on page 41 shows the configuration of the VSE files.

In OS/390, you can use the file definitions in the cataloged procedure TISDBTMC.

File definitions for PDM Termination in OS/390

Type of file	Name of file	Use of file
Library	STEPLIB	In this data set concatenation, you declare the libraries where the system looks for the function's load module, the single-task PDM load module, and any exit programs you code.
Input Data Set	INPUT	In this data set, you code the input to the PDM Termination utility.
	CSIPARM	You code this data set to hold the same CSIPARM information that you used to initialize your PDM.
Output Data Set	OUTPUT	You define this data set to hold the output that the functions create.
	SYSUDUMP	You define this data set to hold a dump if you request one.

Files you define for PDM Termination in OS/390

The PDM Termination utility uses no work, Directory, or PDM files.



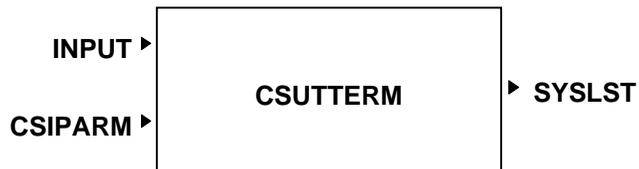
File definitions for PDM Termination in VSE

The following table lists the files you define for VSE:

Type of file	Name of file	Use of file
Library	LIBDEF	In the library definition search chain, you declare the libraries and sublibraries where the system looks for the function's program, the PDM program, the sort program, and any exit programs you code.
Input	SYSIPT	In this system logical unit, you code the input to the PDM termination utility.
	CSIPARM	You code this data set to hold the same CSIPARM information that you used to initialize your PDM.
Output	SYSLST	You define this system logical unit for the output from the functions and the operating system, and for any dump you request.

Files you define for PDM Termination in VSE

The PDM Termination utility uses no work, Directory, or PDM files.



Choosing run-time options

Several parameters affect the run-time environment. You code these parameters when you execute the **Execution Statistics** utility or any functions that require UCL. Do not code these parameters when you execute the **Version 2 Unload, Load, and Insert Linkpath** functions.

To set the run-time environment, you must code the size of the region or partition, the stack, and possibly the I/O buffers. You must also indicate whether you want to print a dump or just messages. Your code would look like one of the following examples, depending on your operating system.

OS/390

In OS/390, code the following:

```
//stepname EXEC yourpgm,PARM='[your run-time  
parameters to the operating system]/  
[your function's run-time parameters]'
```

An example would look like this:

```
//FUNCTION EXEC CSUOUTIL,PARM='/STACK=300K,NOSPIE,NODUMP'
```

VSE

In VSE, code the following:

```
// EXEC yourpgm,SIZE=[your size parameter],  
PARM='[your run-time parameters to the operating system]/  
[your function's run-time parameters]'
```

An example would look like this:

```
// EXEC CSUOUTIL,SIZE=(AUTO,150K),PARM='/STACK=300K,NOSPIE,NODUMP'
```

Defining the amount of storage

You must have sufficient storage available to hold the following:

- ◆ The function's program
- ◆ The size of the stack/heap you code in the STACK parameter in the JCL
- ◆ The single-task PDM
- ◆ The sort work space you code in the MEMORY statement of the UCL (including space for the sort program, if the function uses one)
- ◆ The exit program if you code one in the STANDARD-EXIT statement of the UCL

OS/390

In OS/390, code the size of the address space in the REGION parameter of the JOB statement in the JCL.

VSE

In VSE, you must set the size of your partition when you initialize VSE. When you execute the functions, set the SIZE parameter on the EXEC statement in the JCL to the size of your function. For example, you can code SIZE=AUTO or SIZE=xxxK, where xxx is larger than the function.

In VSE, if your function has a sort program, add the size of the sort work space to the size of the function. For example, code SIZE=(AUTO,xxxK) where xxx is larger than the memory you allocated in the MEMORY statement of the UCL, or code SIZE=yyyK where yyy is larger than the function and the sort work space added together.

Coding the run-time interface parameters

After you set the amount of memory, code the following run-time parameters in any order. The last two interact with each other. To see the results of their possible combinations, see “[Results of different combinations of SPIE and DUMP](#)” on page 45. Follow the coding recommendations to avoid difficulty.

IOBUF=yyyK

OS/390 This parameter indicates the amount of storage to return to OS/390 for I/O buffers and control blocks after the STACK value is allocated. (Not valid in VSE.) Since little of the function's I/O uses this storage, the 36K default is sufficient.



We recommend not coding this parameter.

STACK=xxxK

This parameter sets the amount of storage for the run-time stack/heap. The functions need at least 8K. If you do not code this parameter, the default is the entire region.



You will want to leave room in your region for loading other programs, such as the PDM or an exit program, so we recommend always coding this parameter.

For further information and recommended values, see “[Setting the STACK parameter](#)” on page 46.

SPIE/NOSPIE

This parameter indicates whether the run-time interface should intercept program checks by the operating system.



The default is SPIE; however, we recommend coding NOSPIE (not intercept them).

DUMP/NODUMP

You can use this parameter to indicate whether you want a dump of the address space in OS/390 or the partition in VSE. This parameter determines whether you get a dump when:

- ◆ The run-time interface intercepts a program check. If you coded NOSPIE, the run-time interface does not catch a program check and the operating system creates a dump.
- ◆ A run-time error occurs. If a run-time error occurs, you do not need a dump. The default is DUMP; however, we recommend coding NODUMP.

Results of different combinations of SPIE and DUMP

Run-time options	Action on abend generated by run-time system (2506, a stack/heap collision)	Action on abend generated by operating system (SOC1, an operation exception)
SPIE, DUMP	Run-time system generates abend	Run-time system catches abend
	Run-time system prints messages	Run-time system may print messages
	Run-time system returns abend code (2506)	Run-time system returns abend code (2531)
	Run-time system prints dump	Run-time system prints dump
SPIE, NODUMP	Run-time system generates abend	Run-time system catches abend
	Run-time system prints messages	Run-time system may print messages
	Run-time system returns abend code (2506)	Run-time system returns abend code (2531)
	Run-time system prints no dump	Run-time system prints no dump
NOSPIE, DUMP	Run-time system generates abend	Run-time system does not catch abend
	Run-time system prints messages	Operating system performs abend logic
	Run-time system returns abend code (2506)	Operating system returns abend code (50C1)
	Run-time system prints dump	Operating system may print dump
NOSPIE, NODUMP (recommended)	Run-time system generates abend	Run-time system does not catch abend
	Run-time system prints messages	Operating system performs abend logic
	Run-time system returns abend code (2506)	Operating system returns abend code (50C1)
	Run-time system prints no dump	Operating system may print dump

Setting the STACK parameter

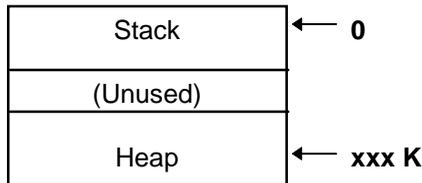
When you code the STACK parameter, you set aside an area of memory called the stack/heap for the functions to use. The stack/heap has two areas, the Procedure Call Stack and the Dynamic Memory Heap, as shown in the following figure.



As the figure shows, the stack starts at the beginning of the area. It stores some run-time information and all global variables. In addition, when each procedure starts, it allocates an area on the stack to store its local variables, and call and return information.

As the procedure calls more nested procedures, the stack grows larger. As the nested procedures return, the stack grows smaller. Thus, the size of the stack depends on the levels of nesting in the procedure calls, and the number and size of the procedures' parameters and local variables. As the stack gets larger, it allocates space toward the heap area.

The heap is the area that functions dynamically allocate at run-time. In it they store internal context information, that is, variables that are not stored in the stack area. When the functions no longer need the space, they free it. As an area is freed, it can be reallocated. Thus, this area varies in size. When a procedure allocates the first heap area, it starts at the end of the space. When a procedure allocates a new area, it takes the new area from the space closer to the stack. As the following figure shows, when the stack and heap get larger, the unused space between them gets smaller.



Estimating the size of the stack/heap

Because the size of the stack/heap varies, it is difficult to provide reliable estimates or formulas to determine the size. It is best to use the STACK value in the JCL example for each function. If this number proves to be insufficient, use [“File definitions for PDM Termination in VSE”](#) on page 41 to estimate how much larger to make it.

You know when the size is too small because you receive an error message and an abend code at run-time. The message states that the stack and heap have collided; that is, they are out of memory.

To estimate how much larger to make the stack, consider the level of complexity in the UCL program. Complex tasks require more space. For example, if you code the LIST option for the Unload function, more procedures are called and you need more stack space.

Because the heap area is allocated dynamically, its size varies more than the stack and is harder to predict. In functions that require UCL, the heap gets larger as the UCL gets more complex. The size of the heap depends on the number of control blocks the functions create for internal use.

To help you determine which functions may require more space, The following table shows the functions that use the stack/heap and on what their size depends.

Function	Variation in stack size	Resulting from
Execution Statistics	Very Little	
PDM Termination	Very Little	
Expand	Average	
Format	Little	
Review	Little	
Unlock	Little	
Print	Average	
Modify	Average	
File Statistics	Much	Complexity of most complex database file.
Log-Print	Very Much	Number of tasks and database files it must analyze.
Recover	Very Much	Number of tasks and database files it must analyze and the size of the largest log record if more than 32K.
Restore	Very Much	Number of tasks and database files it must analyze and the size of the largest log record if more than 32K.
Version 1 Unload	Much	Complexity of most complex database file.
Version 1 Load	Much	Complexity of most complex database file.
Sorted-Populate	Much	The number and complexity of the index files and secondary keys, and the number of record codes for the most complex file.
Depopulate	Much	The number and complexity of the index files and secondary keys, and the number of record codes for the most complex file.
Reorganize	Much	The number and complexity of the index files and secondary keys, and the number of record codes for the most complex file.

Inserting exit programs into functions

At an exit point in a function's code, the function passes control to an exit program you have written. In some cases, the function passes information to your exit program and the exit program passes information back to the function. In other cases, the exit program performs a task.

These functions have exit points where you can insert an exit program:

- ◆ Print
- ◆ Modify
- ◆ Version 1 Unload and Load
- ◆ Sorted-Populate
- ◆ Depopulate
- ◆ Reorganize
- ◆ Recover
- ◆ Restore
- ◆ Log-Print

For information on the exit points in the [Version 2 Unload](#) function, see [“Using exit points”](#) on page 265.

To use an exit program in any of these functions, code the name of the exit program in the STANDARD-EXIT statement of the UCL for the function. Before the function begins executing, your exit program is dynamically loaded. If you coded an exit program for a previous function in the same UCL program, your old exit program is deleted before the new one is loaded.

You can write your exit program in any language that supports the IBM Subroutine Calling Conventions. You must save and restore registers.

If you are writing an exit program in COBOL:

- ◆ **OS/390** Under OS/390, you must issue a call to the COBOL routine ILBOSTP0 before calling the COBOL subroutine.
- ◆ **VSE** Under VSE, issue the call to ILBDSET0.

For more information on the ILBOSTP0 or ILBDSET0 routines, refer to the appropriate IBM COBOL manual.

On entry to your own exit program, follow these conventions:

- ◆ Register 0—Unpredictable
- ◆ Register 1—Parameter list address
- ◆ Register 2–12—Unpredictable
- ◆ Register 13—Address of an area of 18 fullwords that can be used by an exit routine
- ◆ Register 14—Return address
- ◆ Register 15—Entry point address of your exit program

Your exit program does not need to be re-entrant, and you can call your exit program whatever you want.

There is only one exit point in the **Print**, **Modify**, and **Version 1 Unload and Load** functions. For these functions, the parameter list looks like the one in the following table.

Parameter list addresses and contents for a single exit point

Parameter	Data type	Contents before exit (passed to exit program)	Contents after exit (passed from exit program)
Record	<i>n</i> bytes of data	Data record	Must be unchanged
Function name	8 bytes character	Name of function, like PRINT	Same data or changed data if permitted

There are several exit points in the **Sorted-Populate**, **Depopulate**, and **Reorganize** functions. When you have more than one exit point, you must code all the exit logic in one module. All the exit points branch to the single entry point in the module. On the basis of the exit point number in the second parameter, your exit program must determine the exit point to which control is being passed. As the following table shows, different parameters are used when there are multiple exit points.

Parameter list addresses and contents for multiple exit points

Parameter	Data type	Contents before exit (passed to exit program)	Contents after exit (passed from exit program)
Function name	16 bytes character	Name of function, like DEPOPULATE	Must be unchanged
Exit point	4 bytes integer	Exit point number, like 1	Must be unchanged
Action indicator	8 bytes character	bbbbbbbb	bbbbbbbb or other valid values
Data	Variable	Data associated with exit point data	Same data or changed data if permitted

Using sort programs

The [Version 1 Load](#), [Sorted-Populate](#), [File Statistics](#), [Version 2 Load](#), and [Version 2 Unload](#) functions use a sort program. The File Statistics function sorts only if you code STATISTICS (CHAIN) or STATISTICS (LINK) for a coded related file. For information on the Version 2 Unload and Load functions, see “[Coding the Version 2 Unload, Load, and Insert Linkpath functions](#)” on page 225.

For the [Version 1 Load](#), [Sorted-Populate](#), and [File Statistics](#) functions, code the name of the sort program you want to use in the SORT statement in the UCL control section. Depending on your sort program and the amount of data to be sorted, you may need to allocate more than the default amount of sort memory. You may also need to allocate sort work files.



Use of a SORTCNTL file to alter normal EBCDIC sort ordering is not supported. Use of SORTCNTL to alter the standard collating sequence may cause the utilities to fail and/or corrupt the database.

Allocating sort memory

To allocate memory, code the amount in the MEMORY statement in the UCL control section. Generally, the program sorts faster if you allocate more memory.

Depending on your operating system, the memory you allocate may be virtual instead of real memory. Be sure there is enough real memory to support the virtual memory you allocate. If the operating system restricts the sort program to a smaller amount of real memory, it can slow performance. A sort memory value of approximately 300K less than the amount of real memory is recommended, but the best value will depend upon the nature of your individual sort program.

For example, assume you are executing the **Sorted-Populate** function and you allocate 900K of sort memory. As you check the function while it executes, you see that it is using an average of 700K, and the paging rate is high. This indicates that the operating system is allowing the utilities and the sort program only 700K of real memory, and the utilities and sort are trying to use 900K of virtual memory. If you reduce the memory to 400K, your sort program uses 200K, the utilities use about 300K, and the operating system allows about 700K. The paging rate decreases, and your program sorts faster.

Allocating sort work space

Each sort program has its own method of calculating the amount of sort work space it needs to execute. However, all calculations are based on the amount of data to be sorted, which you must determine.

In each function, the sort program sorts only one file at a time and uses the same sort work space for each file. Therefore, you must decide which file has the most data to be sorted and calculate the amount of data for that file. If you are not sure which file needs the most space, make the appropriate calculation for all the files and pick the largest.

You calculate the work sort space differently for each function. See the following sections for each calculation.

The [Version 1 Load](#) function sorts variable-length records. The [Sorted-Populate](#) and [File Statistics](#) functions sort fixed-length records.

For the Version 1 Load function

To calculate the amount of data for the work space in the [Load](#) function, multiply the length of the records by the number of records. To calculate the length, add the lengths of all the parts shown in “[Formatting the data records](#)” on page 188 and multiply that total by the number of records in the file.

For the Sorted-Populate function

To calculate the amount of data for the [Sorted-Populate](#) function, multiply the length of the records by the number of records. To calculate the length, add 17 bytes to the longest secondary key. To calculate the number of records, multiply the number of secondary keys by the number of active records in the file.

If you are populating secondary keys for key-sequenced data sets (KSDS), you calculate the length of the record differently: add 13 bytes to the longest secondary key and then add the length of the KSDS primary file key.

If you are populating secondary keys for coded, related files, the function sorts fewer records if some keys are not defined for all the record codes. In that case, you would need less space for sorting.

For the File Statistics Function

In the **File Statistics** function, the length of the sort records depends on the type of statistics you requested and the type of file for which you requested statistics.

- ◆ If you request chain statistics on a primary file, the length of the sort record is 23 bytes. If you request chain statistics on a related file, the length is 19 bytes plus the size of the largest key. Choose the largest key from all the linkpaths on which you requested statistics. The record lengths are the same if you request both link and chain statistics.
- ◆ If you request link statistics and do not request chain statistics on a coded related file, the length of the sort record is 11 bytes. If you request link statistics and do not request chain statistics on a file that is not a coded related file, there is no sort.

Once you have the length of the sort record, calculate the number of records. Again, the number depends on the type of statistics and the type of file. In addition, there are two types of sort records: type one for link statistics and type two for chain statistics.

- ◆ If you requested link statistics for a coded related file, the number of type one sort records is the same as the number of records in the file. If the file is primary or non-coded related, there are no type one records. These numbers are not affected by whether you request chain statistics in addition to link statistics.
- ◆ If you requested chain statistics, you must figure the number of type two sort records. This number depends on the type of file. If you request statistics on a primary file, the number of sort records is the same as the number of records in the primary file. If you request statistics on a related file, multiply the number of records in the file by the number of linkpaths on which the File Statistics function is collecting statistics. These numbers are the same if you also request link statistics.

Once you have the number of type one and type two sort records, add the numbers to get the total number of records and multiply the total by the length of the longest sort record.

Handling errors in sort programs

In most cases, when the sort program abends or passes a bad status back to the function, the function prints an error message and stops processing. You should correct the error indicated by the message.

However, if you coded RC16=ABE when you installed your sort program, the function cannot print a message when the sort program fails. You have indicated that if the sort program has problems, it should abend and return a code of 16. It does not dump and may not print out a message. If the sort program has problems, you simply receive a return code of U0016 when you execute any function that sorts, that is, the **Sorted-Populate**, **File Statistics**, or **Version 1 Load** functions.

3

Coding the control section

For utilities that require UCL, you must begin by coding a control section. The control section defines the processing environment for the functions you want to perform and can contain the following parts of the environment:

- ◆ The names of the database schema and environment description
- ◆ The content and format of the output listing
- ◆ Options for the sort program
- ◆ Description of the output data file
- ◆ Description of the log file

The sample UCL program on the following page shows the names of the schema, environment description, sort program, and output data file.

Within the control section, you can code any number of FUNCTION statements to perform the tasks you need. You begin the function statements with FUNCTION commands. (In this example, the function sections have been abbreviated for clarity.) After the control and function sections, you code CONTROL (END).

CONTROL (BEGIN)	<i>Initiates UCL program.</i>
ENV-DESC (MYDESC)	<i>Names environment description.</i>
SCHEMA (MYSCHEMA)	<i>Names schema.</i>
SORT (SORTPROG)	<i>Names sort program.</i>
DATA-FILE (OUTFILE)	<i>Names data file.</i>
LABEL (YES)	<i>Denotes labeled data file.</i>
FUNCTION (name)	<i>Invokes a function.</i>
CONTROL (END)	<i>Terminates program.</i>

For information on how to code the function sections, see the remaining chapters in this manual.

Coding the UCL for the control section

The following format and format descriptions show how to code a control section. You must code CONTROL and FUNCTION statements for all functions. Most statements have supplied defaults; however, you must supply values for the SCHEMA and ENV-DESC statements.

CONTROL ({ **BEGIN** }
 { **END** })

ENV-DESC (*environment-description-name*)

SCHEMA (*schema-name*)

[**FORMAT** ([**NO**]
 [**YES**])]

[**DIAGNOSTICS** ([**ABEND**]
 [**SIMPLE**])
 [**EXTENDED**]]

```

LIST ([ALL] [NONE] [AFTER] [BEFORE] [BLOCK] [SYSTEM]
      [FUNCTION] [DESCRIPTION] [APPLIED-IMAGES])
      [
        HEADER ( [NO]
                 [YES]
                 [EXTENSION ('string')]
                 [SUPPRESS ( [ELEMENT] [SPACE] [REFER])] )
      ]

      [LINES ( [1]
               [nnn] )]

      [DATA-FORMAT ( [HEX]
                    [CHAR] )]

```

```

[ SORT ( [SORT]
         [program - name] )
      ]

[ MEMORY ( [120k]
           [nnnnk] ) ]

```

```

[ CONSOLE ( [NO]
            [YES] )
          ]

[ { [NOTIFY]
    [REPLY] } ('operator - msg - text') ]

```

```
DATA - FILE ( [ CSUDATA ]  
              [ ddname ] )  
  
          [ LABEL ( [ NO ]  
                  [ YES ] ) ]  
  
          [ RECORD - FORMAT ( [ F ]  
                              [ V ]  
                              [ FB ]  
                              [ VB ] ) ]  
  
          [ RECORD - SIZE ( [ b ]  
                           [ nnnnn ] ) ]  
  
          [ BLOCK - SIZE ( [ b ]  
                           [ nnnnn ] ) ]  
  
          [ DEVICE ( [ DISK ]  
                   [ TAPE ] ) ]  
  
[ SUMMARY - DATA ( [ ALL ] [ FILE ] [ NONE ] [ FUNCTION ]  
                   [ CUMULATIVE ] ) ]
```

```

LOG - FILE ( [ ddname ]
             [ LOGFILE ] )

             [ ACCESS - METHOD ( [ BSAM ]
                                 [ BDAM ]
                                 [ ESDS ] ) ]

             [ DEVICE ( [ DISK ]
                       [ TAPE ]
                       [ VSAM ] ) ]

             [ DEVICE - ADDRESS ( [ SYS010 ]
                                  [ SYSnnn ] ) ]

             [ BLOCK - SIZE ( [ b
                               [ nnnn ] ) ]

             [ SEQ - ERROR ( [ EOF
                             ERROR
                             IGNORE
                             WARNING
                             INFORMATION ] ) ]

             [ PDM - ID - ERROR ( [ EOF
                                  ERROR
                                  IGNORE
                                  WARNING
                                  INFORMATION ] ) ]

             [ LOG - ID - ERROR ( [ EOF
                                   ERROR
                                   IGNORE
                                   WARNING
                                   INFORMATION ] ) ]

```

```
FUNCTION ( {  
    FORMAT  
    SORTED - POPULATE  
    DEPOPULATE  
    REORGANIZE  
    FILE - STATS  
    EXPAND  
    UNLOAD  
    LOAD  
    RECOVER  
    RESTORE  
    LOG - PRINT  
    REVIEW  
    UNLOCK  
    PRINT  
    MODIFY  
} ) ...
```

CONTROL ({
 BEGIN
 END
})

Description *Required.* Marks the beginning and end of a utility control program.

Options BEGIN
 END

Considerations

- ◆ CONTROL (BEGIN) must be the first statement and CONTROL (END) must be the last statement in every program.
- ◆ If you code any statements before CONTROL (BEGIN), they cause an error in your program. If you code any after CONTROL (END), they are ignored.
- ◆ If you do not code an argument for CONTROL, you receive unpredictable results.

ENV-DESC (*environment-description-name*)

Description *Required.* Identifies the environment description and requests sign-on to the database.

Format 1–8 alphanumeric characters. The first character must be alphabetic.

Considerations

- ◆ Use this statement for all functions described in this manual.
- ◆ The schema and environment description combine to define the database to the utilities and the PDM.
- ◆ You can operate on only Directory files by naming a bootstrap schema and environment description. In that case, do not code either the DIRECTORY or REALM parameter in your CSIPARM file.
- ◆ You can operate on your own files by naming your own schema and environment description. In that case, you must code a DIRECTORY parameter in the CSIPARM file. In the DIRECTORY parameter, name the bootstrap schema and environment description used when the PDM initializes. Do not code the REALM parameter in the CSIPARM file.
- ◆ Do not code a bootstrap environment description with your own schema or vice-versa.
- ◆ Code all database files with a file open mode of NONE, an access option of UPDATE, and an OPENX option of PROCESS. For more information, refer to the *SUPRA Server PDM and Directory Administration Guide (OS/390 & VSE)*, P26-2250.
- ◆ Do not code ENV-DESC when executing the Recover or Restore functions. Refer to the *SUPRA Server PDM Logging and Recovery Guide (OS/390& VSE)*,P26-2223, for their special CSIPARM file considerations.
- ◆ For the special requirements of the Expand function's CSIPARM file, see “*Coding the Expand function*” on page 147.

SCHEMA (schema-name)

Description *Required.* Identifies the schema you want used for the functions you named.

Format 1–8 alphanumeric characters. The first character must be alphabetic.

Considerations Use this statement for all functions described in this manual.

- ◆ The schema and environment description combine to describe the database to the utilities and the PDM.
- ◆ You can operate on only Directory files by naming a bootstrap schema and an environment description. In that case, do not code either DIRECTORY or REALM parameters in your CSIPARM file.
- ◆ You can operate on your own files by naming your own schema and environment description. In that case, you must code a DIRECTORY parameter in the CSIPARM file. In the DIRECTORY parameter, code the bootstrap schema and environment description used when the PDM initializes. Do not code the REALM parameter in the CSIPARM file.
- ◆ Do not code a bootstrap schema with your own environment description or vice-versa.
- ◆ Do not code SCHEMA when executing the Recover or Restore functions. Refer to the *SUPRA Server PDM Logging and Recovery Guide (OS/390 & VSE)*, P26-2223, for their special CSIPARM file considerations.
- ◆ For the special requirements of the Expand function's CSIPARM file, see “Coding the Expand function” on page 147.

```
FORMAT( [ NO ] )
```

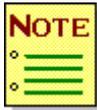
Restriction Use this statement only for the [Version 1 Load](#) function.

Description *Optional.* Determines whether the Load function should format any files not found on the input file but that you coded in the control section.

Default NO

Considerations

- ◆ We recommend that you do not code FORMAT (YES). If you want to format files for which you supplied no data on the input data file, wait until you receive the output listing from the Load function. Then you can see which files the Load function did not load with data, and you can format them with the Format function. For information on the Format function, see [“Coding the Format function”](#) on page 87.



Caution: Exercise extreme caution when you code FORMAT (YES) because it can lose your data. If you list the wrong file or misspell a file's name, the Load function will not find it in the input file. When it formats the file, it changes the data to blank records.

- ◆ If you use FORMAT (YES), the environment description must have a file open mode of NONE and an access mode of UPDATE. For more information, refer to the [SUPRA Server PDM and Directory Administration Guide \(OS/390 & VSE\)](#), P26-2250.
- ◆ When you code FILE (ALL), only your own database files (i.e., your primary, related and index files) are formatted. Task Log, System Log, Statistics, and Directory files are not formatted.

DIAGNOSTICS (

ABEND
SIMPLE
EXTENDED

)

Description *Optional.* Determines the type of diagnostic report to be provided on abnormal termination. You can use this statement with all functions.

Default EXTENDED

Options

ABEND	You receive simple and extended diagnostics and a system dump when necessary.
SIMPLE	You receive simple diagnostics normally of only one line.
EXTENDED	You receive simple and extended diagnostics.

**LIST ([ALL] [NONE] [AFTER] [BEFORE] [BLOCK] [SYSTEM] [FUNCTION]
[DESCRIPTION] [APPLIED-IMAGES])**

Restriction	Use this statement only with the Print, Modify, Version 1 Unload and Load, Recover, Restore, and Log-Print functions.	
Description	<i>Optional.</i> Indicates the content and format of the output listing.	
Default	For the Modify, Version 1 Unload and Load, Recover, Restore, and Log-Print functions, the default is NONE . For the Print function, the default is ALL .	
Options	ALL	Implements all applicable options.
	NONE	Implements none of the options.
	AFTER	Lists after images for Recover, Restore, and Log-Print . For Modify , it lists a record after it is changed.
	BEFORE	Lists before images for Recover, Restore, and Log-Print . For Modify , it lists a record before it is changed.
	BLOCK	Lists log file block headers. (For Recover, Restore, and Log-Print only.)
	SYSTEM	Lists system log file records. (For Recover, Restore, and Log-Print only.)
	FUNCTION	Lists function image records for the log file. (For Recover, Restore, and Log-Print only.)
	DESCRIPTION	Lists log record data formats. (For Recover, Restore, and Log-Print only.)
	APPLIED-IMAGES	Lists log images applied to the database for the Recover and Restore functions.

Considerations

- ◆ If the LIST options are not applicable to a particular function, they are ignored. For example, for the Modify function, LIST(BLOCK) is ignored.
- ◆ If you code LIST(ALL) for the Recover, Restore, and Log-Print functions, you receive a list of all records for all files. Therefore, if you select certain files and records and use LIST(ALL), records for unselected files and records are printed as well.
- ◆ The Print function prints a current image of each file and record that you request, regardless of the arguments you code for this statement. The Print function examines only the DATA-FORMAT statement.
- ◆ If you code multiple options, leave one space between each option.

HEADER (

NO
YES

)

- Restriction** You can only code this statement following a list statement.
- Description** *Optional.* Used for all functions to indicate whether to print the standard utility header on the output listing.
- Default** YES

Considerations

- ◆ The standard title, "DATABASE ADMINISTRATOR UTILITIES," is always printed.
- ◆ If you code HEADER (NO), the second header line is not printed.
- ◆ If you code HEADER (YES) or HEADER () without the EXTENSION statement, the second header line, "FUNCTION= ... FILE= ..." is printed.
- ◆ If you code HEADER (YES) or HEADER () with the EXTENSION statement, the second header line contains the extension. "FUNCTION= ... FILE=..." is not printed.

EXTENSION ('string')

- Restriction** You can only code this statement following HEADER (YES) or HEADER ().
- Description** *Optional.* Used with all functions to print your own header after the standard title.
- Format** 1–120 alphanumeric characters enclosed in single quotes.
- Consideration** If the message text contains a single quote, you must code it as two consecutive single quotes. For example, if the actual message is ABCD'EFG'H, you must code it as 'ABCD"EFG"H'. When you figure the length of the message, two single quotes count as one character.

SUPPRESS ([ELEMENT] [SPACE] [REFER])
Restrictions

- ◆ Use this statement only for the **Print** and **Modify** functions.
- ◆ You can only code this statement following a **HEADER** statement.

Description *Optional.* Lists the heading options you want suppressed on the output listing.

Options	ELEMENT	Do not print element names above the corresponding data element.
	SPACE	Do not insert spaces between data elements.
	REFER	Do not print the relative record number (RRN) of each record above the record.

Consideration The **ELEMENT** and **SPACE** options apply only when you supply a list of element names in the function part of the UCL. If you code **ELEMENT (ALL)**, no names are printed regardless of what you code in the **SUPPRESS** statement.

LINES ($\left[\begin{array}{c} 1 \\ nnn \end{array} \right]$)
Restrictions

- ◆ Use this statement only for the **Recover**, **Restore**, and **Log-Print** functions.
- ◆ You can only code this statement following a **LIST** statement.

Description *Optional.* Indicates the number of output lines you want printed for each data record. Depending on the record length and data format, an output line may constitute several physical lines of output.

Default 1

Format 1–3 numeric characters

Options 1 Prints one logical line of output.

nnn Prints the specified number of output lines.

**DATA - FORMAT ([HEX
[CHAR])**
Restrictions

- ◆ Use this statement for the **Print, Modify, Load, Unload, Recover, Restore,** and **Log-Print** functions.
- ◆ You can only code this statement following the LIST statement.

Description *Optional.* Indicates the format of the data records in the output listing.

Default CHAR

Options HEX Records are printed in over/under hexadecimal format.

CHAR Records are printed in character format.

**SORT ([SORT
[*program-name*])**

Restriction Use this statement only for the **Sorted-Populate, Version 1 Load,** and **File Statistics** functions.

Description *Optional.* Identifies the sort program to be used.

Default SORT

Format 1–8 alphanumeric characters

Consideration The sort name you code must be available in your execution library.

MEMORY ($\left[\begin{array}{c} 120k \\ nnnnnK \end{array} \right]$)

Restrictions

- ◆ Use this statement only for the **Sorted-Populate**, **Version 1 Load**, and **File Statistics** functions.
- ◆ You can only code this statement following a SORT statement.

Description *Optional.* Indicates the amount of memory you want allocated for the sort program.

Default 120K

Format 1–5 numeric characters followed by K

Considerations

- ◆ More memory results in better performance.
- ◆ The amount of memory you code must be available in the region or partition in which you are executing the utilities. For more details on execution-time parameters, see “**Executing the functions**” on page 33.

CONSOLE ($\left[\begin{array}{c} NO \\ YES \end{array} \right]$)

Restriction Use this statement only for the **Recover**, **Restore**, and **Log-Print** functions.

Description *Optional.* Indicates whether the system console is to display messages.

Default NO

{NOTIFY}
{REPLY} ('*operator-msg-text*')

Description *Optional.* Specifies the text of the notification message displayed by the system console

Format 1–50 alphanumeric characters enclosed in single quotes

Options

NOTIFY	Notification message only
REPLY	Notification message and operator response

Considerations

- ◆ If you code the REPLY statement, the operator must reply to the console message.
- ◆ If the message contains single quotes, code each as two consecutive single quotes. For example, if the message is ABCD'EFG'H, code it as 'ABCD"EFG"H'. When you figure the length of the message, count two single quotes as one character.
- ◆ If you do not code CONSOLE (YES), this statement is ignored.

DATA - FILE (**[CSUDATA]**
[ddname])

Restriction Use this statement only for the [Version 1 Unload and Load](#) functions.

Description *Optional.* Identifies the ddname that refers to the Unload/Load function's data file.

Default **[OS/390]** ddname of CSUDATA
[VSE] DLBL name of CSUDATA and logical device name of SYS021

Format 1–7 alphanumeric characters

Consideration In your JCL, the data file must have the ddname specified in this statement.

LABEL (

NO
YES

)

Restrictions

- ◆ Use this statement only for the **Version 1 Unload and Load** functions.
- ◆ You can only use this statement following a DATA-FILE statement.

Description *Optional.* Indicates whether the data file on tape contains standard labels.

Default YES

Options NO The data file is on an unlabeled tape.
 YES The data file is on a labeled tape.

Consideration This statement is ignored if you do not code DEVICE (TAPE).

RECORD - FORMAT (

F
V
FB
VB

)

Restrictions

- ◆ Use this statement only for the **Version 1 Unload** function.
- ◆ You can only use this statement following a DATA-FILE statement.

Description *Optional.* Indicates the format of the data file records.

Default VB

Options F Fixed format
 V Variable format
 FB Fixed blocked format
 VB Variable blocked format

Consideration Only the Unload function uses this statement. The Load function takes its record format from the run control record. If you code this statement for the Load function, it is ignored.

RECORD - SIZE ($\left[\begin{array}{c} \mathit{b} \\ \mathit{nnnnn} \end{array} \right]$)

Restrictions

- ◆ Use this statement only for the [Version 1 Unload](#) function.
- ◆ You can only use this statement following a DATA-FILE statement.

Description *Optional.* Indicates the logical record length of the data file records.

Default *b*

Format 1–5 numeric characters

Considerations

- ◆ You must make the record size large enough to handle all of the data fields you want to unload plus the unloaded record prefix.
- ◆ If you code V (variable) or VB (variable blocked) in the RECORD-FORMAT statement, you must add the four bytes for the record length descriptor when you calculate the size of the record.
- ◆ The smallest record size is 48 bytes. The largest record size is 32,500 bytes or the track size of the disk drives, whichever is smaller. You must code a value that is at least equal to the sum of the following items:
 - Length of the data (the logical record length of the largest file)
 - Length of the control key
 - Length of the control information (14 bytes)

For the structure of the unloaded record, see [“Coding the Version 1 Unload and Load functions”](#) on page 153.
- ◆ If you do not code a record size, the system selects an appropriate default value.
- ◆ Only the Unload function uses this statement. The Load function takes its record size from the run control record and ignores any value you code in this statement.

BLOCK - SIZE ($\left[\begin{array}{c} \text{b} \\ \text{nnnnn} \end{array} \right]$)

Restrictions

- ◆ Use this statement only for the **Version 1 Unload** function.
- ◆ You can only use this statement following a DATA-FILE statement.

Description *Optional.* Selects the block size for the data file.

Default b

Format 1–5 numeric characters

Considerations

- ◆ If the option for the RECORD-FORMAT statement is
 - F The BLOCK-SIZE must equal the RECORD-SIZE
 - V The BLOCK-SIZE must equal the RECORD-SIZE plus 4
 - FB The BLOCK-SIZE must be an even multiple of the RECORD-SIZE
 - VB The BLOCK-SIZE must be the number of records per block times the RECORD-SIZE plus 4
- ◆ If you do not code a block size, the system selects an appropriate value.
- ◆ The Load function takes the record format from the run control record and ignores any value you code here.

DEVICE ($\left[\begin{array}{c} \text{DISK} \\ \text{TAPE} \end{array} \right]$)

Restrictions

- ◆ Use this statement only for the **Version 1 Unload and Load** functions.
- ◆ You can only use this statement following a DATA-FILE statement.

Description *Optional.* Indicates the device type for the data file.

Default DISK

SUMMARY-DATA ([ALL] [FILE] [NONE] [FUNCTION] [CUMULATIVE])

Description *Optional.* Identifies the intervals at which you want summary data reported. The utilities accumulate and present this data on the output listing.

Default CUMULATIVE

Options

ALL	The utilities accumulate and list summary data for the FILE, FUNCTION, and CUMULATIVE intervals.
FILE	The utilities list summary data after processing each completed file for the Sorted-Populate, Version 1 Unload and Load, Print, Modify, and File Statistics functions.
NONE	The utilities provide no data.
FUNCTION	The utilities list summary data after each Sorted-Populate, Version 1 Unload and Load, Print, Modify, and File Statistics function. They accumulate the data for all files the function processed.
CUMULATIVE	The utilities list summary data after completing the UCL program. They accumulate the data for all functions and files processed by the UCL program. The utilities print summary data only if you executed at least one of the following functions: Sorted-Populate, Version 1 Unload or Load, Print, Modify, or File Statistics .

Consideration You may code any combination of listing intervals. However, you cannot code ALL or NONE in combination with any other argument.

LOG - FILE (*ddname*
LOGFILE)

Restriction Use this statement only for the **Recover, Restore, and Log-Print** functions.

Description *Optional.* Identifies the ddname that refers to the System Log File.

Default LOGFILE

Format 1–7 alphanumeric characters

Considerations

- ◆ The Log-Print, Recover, and Restore functions can process a System Log File of only one data set. For information on processing a System Log File of several data sets, refer to the *SUPRA Server PDM Logging and Recovery Guide (OS/390 & VSE)*, P26-2223.
- ◆ In your JCL, you must define the System Log File with the ddname you indicate on this statement. If you use the default for this statement, you must use the ddname LOGFILE.
- ◆ If a standard exit reads the System Log File, the exit ignores the file name you code in this statement.

ACCESS-METHOD (

BSAM
BDAM
ESDS

)

Restrictions

- ◆ Use only for **Recover, Restore, and Log-Print** functions.
- ◆ You can only use this statement following a LOG-FILE statement.

Description *Optional.* Indicates the access method used to create the System Log File.

Default BSAM

Options

BSAM	Basic Sequential Access Method
BDAM	Basic Direct Access Method
ESDS	Entry Sequenced Data Set (VSAM)

Considerations

- ◆ If you code DEVICE (TAPE), you cannot code ACCESS-METHOD (BDAM) or ACCESS-METHOD (ESDS).
- ◆ If you code DEVICE (VSAM) under LOG-FILE, you must code ACCESS-METHOD (ESDS). No other methods are allowed with a VSAM device, including the default, BSAM.
- ◆ If a standard exit reads the System Log File, the exit ignores the access method you code.
- ◆ SUPRA supports FBA devices with VSAM (ESDS/KSDS) or as BSAM. Direct access or BDAM access is not supported. FBA users need to recover/restore from tape or ESDS log file. The system log tape must be labeled for the Restore function. Unless the exits are implemented, RESTORE/RECOVER/LOG-PRINT do not process tape labels. Therefore, the tape must be positioned on the data file (MTC FSF, 181,1).

DEVICE ($\left[\begin{array}{c} \text{DISK} \\ \text{TAPE} \\ \text{VSAM} \end{array} \right]$)

Restrictions

- ◆ Use only for **Recover, Restore, and Log-Print** functions.
- ◆ You can only use this statement following a LOG-FILE statement.

Description *Optional.* Indicates the type of storage device for the System Log File.

Default DISK

Considerations

- ◆ When you code DEVICE (VSAM), you must code ACCESS-METHOD (ESDS).
- ◆ If a standard exit reads the System Log File, the exit ignores the device you code in this statement.

DEVICE - ADDRESS ($\left[\begin{array}{c} \text{SYS010} \\ \text{SYS}nnn \end{array} \right]$)

Restrictions

- ◆ VSE only.
- ◆ Use only for **Recover, Restore, and Log-Print** functions.
- ◆ You can only use this statement following a LOG-FILE statement.

Description *Optional.* Indicates the logical unit address of the System Log File.

Default SYS010

Options SYS010 Logical device name is SYS010.

SYSnnn Logical device name is SYS followed by the three-digit number you specify.

Consideration If a standard exit reads the System Log File, the exit ignores the device address you code in this statement.

VSE **BLOCK - SIZE** ($\left[\begin{array}{c} \underline{b} \\ nnnnn \end{array} \right]$)

Restrictions

- ◆ VSE only.
- ◆ Use only for **Recover, Restore, and Log-Print** functions.
- ◆ You can only use this statement following a LOG-FILE statement.

Description *Optional.* Indicates the size of the log block.

Default *b*

Format 1–5 numeric characters

SEQ - ERROR ($\left[\begin{array}{c} \underline{EOF} \\ \underline{ERROR} \\ \underline{IGNORE} \\ \underline{WARNING} \\ \underline{INFORMATION} \end{array} \right]$)

Restriction You can only use this statement following a LOG-FILE statement.

Description *Optional.* Presents a message stating the action you want taken if a block sequence error occurs on the System Log File.

Default EOF

Options	EOF	Treat the exception as an end-of-file and continue processing.
	ERROR	Present an error message and terminate processing.
	IGNORE	Take no action and continue processing.
	WARNING	Present a warning message and continue processing.
	INFORMATION	Present an informational message and continue processing.

```
PDM - ID - ERROR ( [ EOF  
                   [ ERROR  
                   [ IGNORE  
                   [ WARNING  
                   [ INFORMATION ]  
                   ] )
```

Restrictions

- ◆ Use only for the **Recover, Restore, and Log-Print** functions.
- ◆ You can only use this statement following a LOG-FILE statement.

Description *Optional.* Presents a message stating the action required after a PDM-ID error.

Default EOF

Options

EOF	Treat the exception as an end-of-file and continue processing.
ERROR	Present an error message and terminate processing.
IGNORE	Take no action and continue processing.
WARNING	Present a warning message and continue processing.
INFORMATION	Present an informational message and continue processing.

```
LOG - ID - ERROR ( [ EOF
                   ERROR
                   IGNORE
                   WARNING
                   INFORMATION ] )
```

Restrictions

- ◆ Use only for the **Recover, Restore, and Log-Print** functions.
- ◆ You can only use this statement following a LOG-FILE statement.

Description *Optional.* Presents an action message after a log identifier error.

Default EOF

Options	EOF	Treat the exception as an end-of-file and continue processing.
	ERROR	Present an error message and terminate processing.
	IGNORE	Take no action and continue processing for the function.
	WARNING	Present a warning message and continue processing.
	INFORMATION	Present an informational message and continue processing.

```

FUNCTION ( {
  FORMAT
  SORTED - POPULATE
  DEPOPULATE
  REORGANIZE
  FILE - STATS
  EXPAND
  UNLOAD
  LOAD
  RECOVER
  RESTORE
  LOG - PRINT
  REVIEW
  UNLOCK
  PRINT
  MODIFY
} )

```

Description *Required.* Selects the function. Except for the **Expand** and **Unload** functions, you may code functions more than once in a UCL program. Some combinations of functions are not valid.

- Options**
- FORMAT Formats a database file into SUPRA native format. (See “**Coding the Format function**” on page 87.)
 - SORTED-POPULATE Creates the secondary key's tree structure. (See “**Coding the Sorted-Populate function**” on page 91.)
 - DEPOPULATE Deletes secondary keys. (See “**Coding the Depopulate function**” on page 105.)
 - REORGANIZE Rebuilds the tree structure after you have updated it. (See “**Coding the Reorganize function**” on page 119.)
 - FILE-STATS Reports the physical and logical characteristics of the primary or related SUPRA native format files you are examining. (See “**Coding the File Statistics function**” on page 131.)
 - EXPAND Expands the capacity of existing related SUPRA native format files. (See “**Coding the Expand function**” on page 147).

UNLOAD	Extracts records from database files and writes them to a sequential output medium. You can use the resulting data file only to reload the files into SUPRA native format. (See “ Coding the Version 1 Unload and Load functions ” on page 153.)
LOAD	Formats database files into SUPRA native format and writes data records to the files from a sequential medium. (See “ Coding the Version 1 Unload and Load functions ” on page 153.)
RECOVER	Recovers PDM database files from a SUPRA PDM System Log File. (See “ Coding the Recover, Restore, and Log-Print utilities ” on page 447.)
RESTORE	Restores PDM database files from a SUPRA PDM System Log File. (See “ Coding the Recover, Restore, and Log-Print utilities ” on page 447.)
LOG-PRINT	Prints the contents of the SUPRA PDM System Log File and reports statistics for the log file and PDM files. (See “ Coding the Recover, Restore, and Log-Print utilities ” on page 447.)
REVIEW	Examines SUPRA native and converted files to see if they are locked and prints an appropriate message. (See “ Coding the Review function ” on page 485.)
UNLOCK	Resets the lock field in a database file that did not go through the normal PDM close logic due to an abend or system failure. Use Unlock with extreme caution. If you use the Unlock function instead of recovery procedures, you may corrupt your database. (See “ Coding the Unlock function ” on page 489.)
PRINT	Prints records from a database file. (See “ Coding the Print function ” on page 363.)
MODIFY	Updates records in a database file. (See “ Coding the Modify function ” on page 381.)

Determining control statements for functions

Once you are familiar with the information in the preceding section, use the following figure for quick reference to the control statements needed for each function. The control statements are indented to show the hierarchical structure.

CONTROL STATEMENTS	FUNCTION															
	Format	Sorted-Populate	Depopulate	Reorganize	File Statistics	Expand	Unload	Load	Print	Modify	Recover	Restore	Log-Print	Review	Unlock	
CONTROL	R	R	R	R	R	R	R	R	R	R	R	R	R	R	R	
ENV-DESC	R	R	R	R	R	R	R	R	R	R	R	R	R	R	R	
SCHEMA	R	R	R	R	R	R	R	R	R	R	R	R	R	R	R	
FORMAT								O								
DIAGNOSTICS	O	O	O	O	O	O	O	O	O	O	O	O	O	O	O	
LIST							O	O	O	O	O	O	O			
HEADER	O				O	O	O	O	O	O	O	O	O			
EXTENSION	O				O	O	O	O	O	O	O	O	O			
SUPPRESS									O	O						
LINES											O	O	O			
DATA-FORMAT							O	O	O	O	O	O	O			
SORT		O			O	O		O								
MEMORY		O			O			O								
CONSOLE											O	O	O			
NOTIFY											O	O	O			
REPLY											O	O	O			
DATA-FILE							O	O								
LABEL							O	O								
RECORD-FORMAT							O									
RECORD-SIZE							O									
BLOCK-SIZE							O									
DEVICE							O	O								
LOG-FILE											O	O	O			
ACCESS-METHOD											O	O	O			
DEVICE											O	O	O			
DEVICE-ADDRESS											O	O	O			
BLOCK-SIZE											O	O	O			
SEQ-ERROR											O	O	O			
PDM-ID-ERROR											O	O	O			
LOG-ID-ERROR											O	O	O			
SUMMARY-DATA		O			O	O	O	O	O	O						
FUNCTION	R	R	R	R	R	R	R	R	R	R	R	R	R	R	R	

Legend: R Required O Optional (blank) Not used by function

4

Coding the Format function

Before you add any records, you must format the PDM database files. In addition, you must format Directory files, the System Log File, and the Task Log File before you use them. You do not need to format the BSAM files or the Statistics file.

You can use Format to format database files or use the [Version 1 Load](#) function to format and add records in one step. Both functions format only files in SUPRA native format. To create files in compatibility or converted format, you must use the [Version 2 Load](#) utility.

When you use Format, it builds the file control records for index, primary, and related files. Format also sets all records to spaces, except with key-sequenced data sets where it only writes the file control record to the file.



Warning: If you format an existing, non-VSAM file, you set the records to blanks. Therefore, you should not format files that contain data you need. You should format only files that are empty, backed up, or no longer needed. If you format over an index file, you delete the secondary keys. If you want to repopulate the file, you must depopulate and purge it first.

After you code the control section as shown in “[Coding the control section](#)” on page 57, you can code the Format function as shown in the following format.

Format function syntax

FUNCTION (FORMAT)
**FILE ({ ALL
file-name-list }),...**

FUNCTION (FORMAT)

Description *Required.* Invokes the Format function.

**FILE ({ ALL
file-name-list }),...**

Description *Required.* Names the database files you want formatted.

Format File names must be 4 alphanumeric characters. The first character must be alphabetic. Separate names with commas.

Options ALL Formats all index, primary, and related files in the schema.

file-name-list Formats the files you name. You can format database, Directory, System Log, or Task Log files.

Considerations

- ◆ When you use Format you should not use task or system logging, which slow performance and serve no purpose. In addition, if you use logging and abnormally terminate, you cannot restart.
 - To format a System Log File, turn system logging off, that is, code the option in the environment description with *nnnn*. In the control section of the UCL, code the user schema and environment description. All files in the user environment description must have an open mode of NONE. In the CSIPARM file, code the DIRECTORY parameter using the boot schema and environment description.
 - To format a Task Log File, turn task logging off by coding the option in the bootstrap environment description with *n*. In the control section, code the bootstrap schema and environment description. In the CSIPARM file, do not code a DIRECTORY or REALM parameter.
 - To format a Directory file, code the bootstrap schema and environment description in the control section. In the CSIPARM file, do not code a DIRECTORY or REALM parameter.
 - To format database files, code your schema and environment description in the control section of the UCL. In the CSIPARM file, code a bootstrap schema and environment description in the DIRECTORY parameter. Do not code the REALM parameter.
- ◆ If you code FILE (ALL), the function formats only index, primary, and related files. It does not format the Directory, Statistics, System Log, or Task Log files.
- ◆ If you code FILE (ALL), the function formats the files in alphabetical order with index files first, and then primary and related files.
- ◆ You must delete and redefine a VSAM file (ESDS or KSDS) before you format it.
- ◆ To format Directory files, you must code FILE (*file-name*). In the control section, you must code a bootstrap schema and environment description. In the CSIPARM file, do not code a DIRECTORY parameter.
- ◆ If you code FILE(), the function does not format any file.
- ◆ The record length of a key-sequenced data set must be at least the size of the key displacement, plus the key length, plus 25 bytes.
- ◆ You can code the FILE statement one or more times.

Examples

- ◆ When you want to initialize your files for use by the PDM, you must first format them. For example, if you want to add data to the PANM, RANV, and PO01 files that are found in the UTILSCHM schema, code the following:

```
CONTROL (BEGIN)
  ENV-DESC (UTED00US)
  SCHEMA (UTILSCHM)
  FUNCTION (FORMAT)
    FILE (PANM, RANV)
    FILE (PO01)
CONTROL (END)
```

- ◆ This example shows the code and the listing that you receive after the code is validated and executed.

```
CSUL0101I : COMMENCING COMMAND VALIDATION.
1 CONTROL(BEGIN)
2 *****
3 *
4 * FORMAT EXAMPLE #1 DESCRIPTION
5 *
6 * OBJECTIVE: FORMAT THE DATABASE FILES PRIOR TO
7 * USE BY THE PDM.
8 *
9 *
10 *****
11 ENV-DESC(UTED00US)
12 SCHEMA(UTILSCHM)
13 FUNCTION(FORMAT)
14 FILE(PANM,RANV)
15 FILE(PO01)
16 CONTROL(END)
CONTENTS OF SOURCE LINES OUTSIDE 1..72 MARGINS IGNORED.
0 SYNTAX ERRORS DETECTED.
16 COMMAND LINES READ.
1 CONTROL SECTIONS ANALYZED.
1 FUNCTION COMMANDS ANALYZED.

CSUL0102I : COMMENCING COMMAND EXECUTION.
CSUL0301I : COMMENCING CONTROL SECTION USING ENVIRONMENT DESCRIPTION UTED00US AND
SCHEMA UTILSCHM.
CSUL0302I : COMMENCING FORMAT PROCESS.
CSUL0311I : COMMENCING FORMAT AGAINST FILE PANM.
CSUL2800I : FILE PANM IS NOW FORMATTED.
CSUL0321I : FORMAT PROCESSING AGAINST FILE PANM TERMINATING NORMALLY.
CSUL0311I : COMMENCING FORMAT AGAINST FILE RANV.
CSUL2800I : FILE RANV IS NOW FORMATTED.
CSUL0321I : FORMAT PROCESSING AGAINST FILE RANV TERMINATING NORMALLY.
CSUL0311I : COMMENCING FORMAT AGAINST FILE P001.
CSUL2800I : FILE P001 IS NOW FORMATTED.
CSUL0321I : FORMAT PROCESSING AGAINST FILE P001 TERMINATING NORMALLY.
CSUL0303I : FORMAT PROCESS TERMINATING.
CSUL0305I : CONTROL SECTION TERMINATING.
CSUL0307I : ALL CONTROL SECTIONS PROCESSED.
CSUL0103I : DATABASE UTILITIES SUCCESSFUL TERMINATION.
```

5

Coding the Sorted-Populate function

The Sorted-Populate function is the first of three secondary key functions. With the Sorted-Populate function, you can create the secondary key tree structure on large files more quickly than with the Directory Maintenance POPULATE command. When you use the Sorted-Populate function to populate a secondary key, you can request statistics, add your own exit program, and indicate how much of each block you want to hold records.

As its name implies, the Sorted-Populate function's increased efficiency comes from sorting. You need to estimate the amount of sort memory and sort work space required. For information on calculating the amounts, see [“Using sort programs”](#) on page 52. For more information on coding the UCL, see [“Coding the UCL for the Sorted-Populate function”](#) on page 92.

The other secondary key functions, Depopulate and Reorganize, enable you to maintain secondary keys. The Depopulate function deletes secondary keys. The Reorganize function corrects the deterioration of the tree structure that occurs from updating it. These functions are explained in [“Coding the Depopulate function”](#) on page 105 and [“Coding the Reorganize function”](#) on page 119.

Coding the UCL for the Sorted-Populate function

After you code the control section as shown in “Coding the control section” on page 57, you can code the Sorted-Populate function as shown in the following format:

FUNCTION (SORTED - POPULATE)

$$\left[\text{STATISTICS} \left(\begin{array}{c} \text{ALL} \\ \text{BASE} \\ \text{NONE} \end{array} \right) \right]$$

$$\left[\text{STANDARD - EXIT} \left(\textit{exit-name} \right) \right]$$

$$\text{FILE} \left(\left\{ \begin{array}{c} \text{ALL} \\ \textit{file-name-list} \end{array} \right\} \right) , \dots$$

$$\left[\begin{array}{c} \text{SECONDARY - KEY} \left(\begin{array}{c} \text{ALL} \\ \textit{key-name-list} \end{array} \right) \\ \\ \text{LOAD - DENSITY (0 - 99)} \end{array} \right] , \dots$$

FUNCTION (SORTED-POPULATE)

Description *Required.* Executes the Sorted-Populate function.

Consideration Performance is highly sensitive to the number of buffers in the INDEX file's buffer pool. The recommended number of buffers is twice the expected tree height plus three. If you do not know the expected tree height, use 10–15 buffers. Too few buffers will cause buffer thrashing and substantial performance degradation. The number of DATA file buffers is not critical—one or two is sufficient.

STATISTICS ($\left[\begin{array}{l} \text{ALL} \\ \text{BASE} \\ \text{NONE} \end{array} \right]$)

Description	<i>Optional.</i> Indicates the statistics reports you want generated.	
Default	BASE	
Options	ALL	You receive statistics reports on both index files and secondary keys.
	BASE	You receive reports on only the secondary keys.
	NONE	You receive no statistics reports.

Considerations

- ◆ You may request statistics only once.
- ◆ If you request statistics, you must code this statement before the FILE statement(s).

STANDARD-EXIT (*exit-name*)

Description	<i>Optional.</i> Indicates you want to use an exit program with this function.
Default	The function skips the exit points.
Format	1–8 alphanumeric characters. The first character must be alphabetic.

Considerations

- ◆ You may code the exit only once.
- ◆ If you code the exit, you must code it before the FILE statement.
- ◆ If you code the STANDARD-EXIT statement, you must make the exit program available for the function to load. You must put the exit program in your execution library.
- ◆ Sorted-Populate loads the exit program before processing each file you code in each FILE statement. The function deletes the exit program when it completes processing each file.
- ◆ For a description of the exit points in this function, see [“Writing exit programs”](#) on page 97.

FILE ({ **ALL**
 file-name-list }) ,...

Description *Required.* Names the database files you want to populate with secondary keys.

Format File names must be 4 alphanumeric characters. The first character must be alphabetic. Separate names with commas.

Considerations

- ◆ You must code the FILE statement at least once, and you can code it as many times as you like.
- ◆ You must code the FILE statement after all other Sorted-Populate statements.
- ◆ You can code only primary and related files.
- ◆ If you code FILE (ALL), the function populates every secondary key for every primary and related file in the schema and environment description.
- ◆ If you code FILE (ALL), you cannot code any other FILE statements.
- ◆ When the function begins processing each file, it opens the file for exclusive update (EUPD open mode). When the function finishes, it closes the file. If you code a file in more than one FILE statement, the function opens, sweeps, and closes it each time.
- ◆ If you want to populate one data file with several secondary keys, you can code several FILE statements. This method takes more I/O and more CPU time, but less sort work space. However, since this method degrades performance, you should do this only if sort work space is a problem.

SECONDARY - KEY ($\left[\begin{array}{l} \text{ALL} \\ \text{key-name-list} \end{array} \right]$)

Restrictions

- ◆ To code the SECONDARY-KEY statement, you must code the FILE statement.
- ◆ If you coded FILE (ALL) or a file list with more than one file name, you cannot code SECONDARY-KEY (*key-name-list*).

Description *Optional.* Indicates the secondary keys you want populated.

Default ALL

Format Secondary key names must be 8 alphanumeric characters. Separate names with commas.

Considerations

- ◆ You can code the SECONDARY-KEY statement as many times as you like, or not at all.
- ◆ If you code SECONDARY-KEY (ALL), the function populates all secondary keys for the files you code in the FILE statement.
- ◆ If you code SECONDARY-KEY (ALL), you cannot code any other SECONDARY-KEY statements.
- ◆ If a secondary key is damaged or depopulated but not purged, you cannot populate it. You must depopulate and purge it first. Likewise, you cannot use this function on a populated secondary key until you depopulate and purge it.
- ◆ Sorted-Populate handles a maximum of 512 internal elements. If Sorted-Populate generates more than 512 elements, the utility displays a CSUL0502S message containing CODE=3202, and terminates abnormally. If this occurs, you must use the Directory Maintenance utilities for the secondary key(s). This problem will most likely occur on a related file containing many coded records. The utility builds an element list containing all elements for each record code, plus all elements that make up each secondary key.

LOAD-DENSITY (0–99)

Restriction To code the LOAD-DENSITY statement, you must code the SECONDARY-KEY statement.

Description *Optional.* Indicates how full you want the index file blocks.

Default 0

Options 0–99

Considerations

- ◆ You can code the LOAD-DENSITY statement only once for each SECONDARY-KEY statement.
- ◆ If you code LOAD-DENSITY (0), the function loads each secondary key to the load density defined for the secondary key on the Directory.
- ◆ If you code a load density between 1 and 99, the function fills the index file blocks as closely as possible to that percentage, considering the key-data length and block size.

Writing exit programs

With the exit points from the Sorted-Populate function, you can collect data about secondary keys, turn off exit points, abort Sorted-Populate processing, and see statistics.

See “[Inserting exit programs into functions](#)” on page 49 for information on how exit programs are loaded, how they operate, the languages you can use to write them, and the register conventions you must follow. In register 1, for example, you must code the parameter list addresses. The following table describes the parameter list addresses.

Parameter	Data type	Contents before exit (passed to exit program)	Contents after exit (passed from exit program)
Function name	16 bytes character	SORTED-POPULATE \textasciitilde	Must be unchanged
Exit point	4 bytes integer	Exit point number	Must be unchanged
Action indicator	8 bytes character	$\text{\textasciitilde}\text{\textasciitilde}\text{\textasciitilde}\text{\textasciitilde}\text{\textasciitilde}\text{\textasciitilde}\text{\textasciitilde}\text{\textasciitilde}$	$\text{\textasciitilde}\text{\textasciitilde}\text{\textasciitilde}\text{\textasciitilde}\text{\textasciitilde}\text{\textasciitilde}\text{\textasciitilde}$ or other valid values
Data	Variable	Data associated with exit point data	Same data or changed data if permitted

If your exit program changes anything it is not authorized to change, the results are unpredictable.

To use the exit points, see the following sections, which give the exit number, the data associated with an exit, and the valid actions.

Selecting exit points

To use an exit point, you must know when it occurs in the function, what data is passed, and what actions your program can take at that point. The following table shows when an exit point occurs. “[Data parameters](#)” on page 99 shows the data parameters, and “[Valid actions for exit programs in Sorted-Populate](#)” on page 100 shows the actions you can take. An example shows the order in which the function takes the exit points.

When exit points occur

Exit	Phase	When exit occurs
1	Initialization	During initialization just before the function opens the PDM file
2	Termination	After the function has completed all processing
3	Secondary key initialization	Just before the function starts populating the secondary key
4	Secondary key termination	After the function completes populating a secondary key
5	Data file sweep	Just after each RDNXT issued to the PDM that returns asterisks or an END. status
6	Data file sweep	Before passing the secondary key entry record to the SORT program
7	Secondary key population	After the sort program returns the secondary key entry record
8	Secondary key statistics	After the function prints a secondary key statistics line in the base statistics report
9	Index file statistics	After the function prints an index file statistics line in the extended statistics report

At each exit point, your program can indicate whether action should continue or stop. To continue, your program should return blanks. To stop processing, your program should return ABORT.

ABORT has a different scope at different exit points. At exits on the file level (1, 2, 5, 6, 8, and 9), the function stops processing that file and continues with the next file. At exit points on the secondary key level (3, 4, and 7), the function stops processing that secondary key and continues with the next secondary key in the same file.

In addition to continuing or aborting, your program at the first exit point can also change the switches in the data parameter from Y (Yes) to N (No). When your program changes a switch, it turns off the exit point and the function does not take the exit. If your program changes a switch from Y to N, it must also return an action of SET ~~bbbb~~. The programs at the other exit points cannot change the data parameter.

Summary of data parameters and valid actions

Data parameters

The following table shows the data parameters for each exit program.

Exit	Use	Data parameter
1	Initialization	The four-character file name followed by a string of nine Y (yes) or N (no) switches to set exit settings. Each switch turns on or off its exit point.
2	Termination	The four-character file name followed by a fullword integer containing the function return code (0, 4, 8, or 16)
3	Secondary key initialization	The secondary key name
4	Secondary key termination	The secondary key name
5	Data file sweep	The data area from the just completed RDNXT function
6	Data file sweep	The secondary key's entry record just before it is passed to the SORT program
7	Secondary key population	The secondary key's entry record just after the SORT program returns it
8	Secondary key statistics	The statistics in the detail line in the order in which they appear in the statistics reports. All numbers are fullword binary integers.
9	Index key statistics	The statistics in the detail line in the order in which they appear in the statistics reports. All numbers are fullword binary integers.

Valid actions for exit programs in Sorted-Populate

The following table shows the actions you can take with Sorted-Populate.

			Action indicators		
Exit	Use	Data parameter	b	S	A
1	Set exit points	13 bytes containing a four-byte file name and a string of nine Ys. Your program may change the Y to N to turn off the exit point. If your program changes the setting, it must return SET bbbbbb	Y	Y	Y
2	Get return code	The file name followed by the return code: 0 (complete), 4 (warning), 8 (error), 16 (internal error)	Y	N	Y
3	Gather information or add processing	8-character name of secondary key	Y	N	Y
4	Gather information or add processing	8-character name of secondary key	Y	N	Y
5	Gather information or add processing	Data area from the RDNXT command	Y	N	Y
6	Gather information or add processing	The secondary key's entry record before it goes to the sort program	Y	N	Y
7	Gather information or add processing	The secondary key's entry record after the sort program returns it	Y	N	Y
8	Get the secondary key's statistics	The secondary key's statistics	Y	N	Y
9	Get the index file's statistics	The index file's statistics	Y	N	Y

Legend: **b** = **bbbbbbbb** - Action continues
S = **SETbbbbbb** - Data contents changed
A = **ABORTbbbb** - Terminates recovery

Example The exit points are taken in the following order when you are populating two secondary keys, *ffffSK01* and *ffffSK02*, that both reside in index file IX01. In the UCL, you requested all statistics:

	Exit point number	Point in processing
	1	During initialization of the Sorted-populate function
Once per record (Loop)	➤ 5 ← ↖	After RDNXT sweeps the data file for secondary keys
	➤ 6 Loop ↑	Before the function passes <i>ffffSK01</i> records to the SORT program
	➤ 6 → ↗	Before the function passes <i>ffffSK02</i> records to the SORT program.
Once for each <i>ffffSK01</i>	3	Before the function starts populating <i>ffffSK01</i>
	➤ 7	After the SORT program returns each <i>ffffSK01</i> record
	4	After the function finishes populating <i>ffffSK01</i>
Once for each <i>ffffSK02</i>	3	Before the function starts populating <i>ffffSK02</i>
	➤ 7	After the SORT program returns each <i>ffffSK02</i> record
	4	After the function finishes populating <i>ffffSK02</i>
	8	After the function prints a secondary key statistics line in the base statistics report for <i>ffffSK01</i>
	8	After the function prints a secondary key statistics line in the base statistics report for <i>ffffSK02</i>
	9	After the function prints an index file statistics line in the extended statistics report for IX01
	2	Just before the Sorted-Populate function terminates

Requesting statistics

When you request statistics, you receive information on the secondary keys in the files you listed. After populating the file, the function prints the following base statistics for each secondary key:

- ◆ The status code (indicating whether population was successful)
- ◆ The index file name
- ◆ The number of key values (number of data records being indexed)
- ◆ The number of unique key values
- ◆ The number of blocks in the secondary key
- ◆ The number of levels in the secondary key
- ◆ The number of low level blocks in the secondary key

If you requested extended statistics, the function also prints the following information for each ex file:

- ◆ The number of blocks in the file
- ◆ The number of blocks in use before and after population
- ◆ The number of free blocks before and after population

The following two code samples show examples of the base and extended statistics. You receive these statistics when you code the following in the function section of the UCL.

Example 1 This example shows base statistics on the Sorted-Populate function.

```
FUNCTION(SORTED-POPULATE)
  STATISTICS(ALL)
  FILE(PANM)
      SECONDARY-KEY(ALL)
      LOAD-DENSITY(80)
CSUL3376I : BEGINNING OF SORTED-POPULATE STATISTICS (BASE).
```

SECONDARY KEY NAME	STATUS	INDEX FILE	NUMBER OF KEY VALUES	NUMBER OF UNIQUE KEY VALUES	BLOCKS IN SECONDARY KEY	LEVELS IN SECONDARY KEY	LOW LEVEL BLOCKS IN SECONDARY KEY
PANMSK01	****	I001	89,989	89,990	1,727	3	1,698
PANMSK02	****	I001	89,989	89,990	4,270	4	4,091

```
CSUL3381I : END OF SORTED-POPULATE STATISTICS (BASE).
```

Example 2 This example shows extended statistics on the Sorted-Populate function.

CSUL3370I : BEGINNING OF SORTED-POPULATE STATISTICS (EXTENDED).

INDEX FILE	BEFORE SORTED-POPULATE			AFTER SORTED-POPULATE	
	NUMBER OF BLOCKS IN FILE	NUMBER OF BLOCKS IN USE	NUMBER OF FREE BLOCKS	NUMBER OF BLOCKS IN USE	NUMBER OF FREE BLOCKS
I001	7,005	163	6,842	6,160	845

CSUL3375I : END OF SORTED-POPULATE STATISTICS (EXTENDED).

6

Coding the Depopulate function

The Depopulate function is the second of three secondary key functions. With the Depopulate function, you can delete secondary keys. The Depopulate function works like the [Sorted-Populate](#) function that creates secondary keys. That is, the Depopulate function has exclusive access to the file, and opens and closes the file each time you code it in a FILE statement. In addition, you can request some of the same statistics and add your own exit program.

An additional parameter in the Depopulate function enables you to reclaim index file blocks when you depopulate the secondary key. You must reclaim the blocks before you can repopulate the key.

Coding the UCL for the Depopulate function

After you code the control section as shown in “[Coding the control section](#)” on page 57, you can code the Depopulate function as shown in the following format.

FUNCTION (DEPOPULATE)

$$\left[\text{STATISTICS} \left(\begin{array}{c} \text{ALL} \\ \text{BASE} \\ \text{NONE} \end{array} \right) \right]$$
$$[\text{STANDARD - EXIT} (\textit{exit-name})]$$
$$\text{FILE} \left(\left\{ \begin{array}{c} \text{ALL} \\ \textit{file-name-list} \end{array} \right\} \right) \dots$$
$$\left[\begin{array}{l} \text{SECONDARY - KEY} \left(\begin{array}{c} \text{ALL} \\ \textit{key-name-list} \end{array} \right) \\ \\ \text{PURGE} \left(\begin{array}{c} \text{NO} \\ \text{YES} \end{array} \right) \end{array} \right] \dots$$

FUNCTION (DEPOPULATE)

Description *Required.* Executes the Depopulate function.

STATISTICS ($\begin{bmatrix} \text{ALL} \\ \text{BASE} \\ \text{NONE} \end{bmatrix}$)

Restriction If you code the STATISTICS statement, you must code it before the FILE statement.

Description *Optional.* Indicates the kinds of statistics reports you want generated.

Default BASE

Options ALL You receive statistics reports on both index files and secondary keys.

 BASE You receive reports on only the secondary keys.

 NONE You receive no statistics reports.

Considerations

- ◆ If you want statistics, you must code the Depopulate function.
- ◆ You may code the STATISTICS statement only once.

STANDARD-EXIT (*exit-name*)

Restriction If you code the STANDARD-EXIT statement, you must code it before the FILE statement.

Description *Optional.* Indicates you want to use an exit program with this function.

Default The function skips the exit points.

Format 1–8 alphanumeric characters. The first character must be alphabetic.

Considerations

- ◆ If you want to take the exit, you must code the Depopulate function.
- ◆ You may code the STANDARD-EXIT statement only once.
- ◆ If you want to use an exit program, you must make it available for the function to load. That is, you must put the exit program in your execution library.
- ◆ “[Writing exit programs](#)” on page 112 describes exit points.

FILE ({ **ALL**
file-name-list }) ...

Description *Required.* Names the data files for which you want to depopulate secondary keys.

Format File names must be 4 alphanumeric characters. The first character must be alphabetic. Separate names with commas.

Considerations

- ◆ You must code the FILE statement at least once.
- ◆ You must code the FILE statement after the other Depopulate statements.
- ◆ You may code only primary and related files.
- ◆ If you code FILE (ALL), the function depopulates all secondary keys for every primary and related file in the schema/environment description.
- ◆ If you code FILE (ALL), you cannot code any other FILE statements.
- ◆ When the function processes each file, it opens and closes the file. If you code a file in more than one FILE statement, the function opens and closes the file each time. You can depopulate several secondary keys in one file by coding one FILE statement and several SECONDARY-KEY statements. This reduces I/O and CPU time, but requires more sort work space.
- ◆ Depopulate opens the file for exclusive update (EUPD open mode). Thus, no other program can use the file while you are deleting secondary keys.

SECONDARY - KEY ($\left[\begin{array}{l} \text{ALL} \\ \text{key-name-list} \end{array} \right]$)

- Restriction** You can code the SECONDARY-KEY statement only after a FILE statement.
- Description** *Optional.* Identifies the secondary keys you want to depopulate.
- Default** ALL
- Format** Secondary key names must be 8 alphanumeric characters. Separate names with commas.

Considerations

- ◆ You may code the SECONDARY-KEY statement as many times as you like, or not at all.
- ◆ If you code SECONDARY-KEY (ALL), the function depopulates all secondary keys for the files you coded in the parent FILE statement.
- ◆ If you code SECONDARY-KEY (ALL), you cannot code any other SECONDARY-KEY statements.
- ◆ SECONDARY-KEY (*key-name-list*) is invalid if you coded FILE (ALL) or a file list with more than one file name.

PURGE (

NO
YES

)

Restriction If you code the PURGE statement, you must code the SECONDARY-KEY statement.

Description *Optional.* Indicates whether you want the index file blocks reclaimed as the key is depopulated.

Default YES

Options NO The index file blocks are not reclaimed.

YES The index file blocks are reclaimed.

Considerations

- ◆ You can code PURGE only once for each SECONDARY-KEY statement.
- ◆ You can code PURGE (NO) only if the secondary keys are populated.
- ◆ You can code PURGE (YES) if the secondary keys are populated, damaged, or depopulated but not purged.
- ◆ To repopulate a secondary key, you must have depopulated and purged it. You cannot repopulate a secondary key that is depopulated, but not purged.

Writing exit programs

With the exit points from the Depopulate function, you can collect data about secondary keys, turn off exit points, terminate depopulate processing, and see statistics.

“[Inserting exit programs into functions](#)” on page 49 discusses how your exit programs are loaded, how they operate, the languages you can use to write them, and the register conventions you must follow. In register 1, for example, you must code the parameter list addresses. For a description of the parameter list addresses, see the following table:

Parameter	Data type	Contents before exit (passed to exit program)	Contents after exit (passed from exit program)
Function name	16 bytes character	DEPOPULATE	Must be unchanged
Exit point	4 bytes integer	Exit point number	Must be unchanged
Action indicator	8 bytes character	bbbbbbbb	bbbbbbbb or other valid values
Data	Variable	Data associated with exit point changed	Same data or changed data if permitted

If your exit program changes anything it is not authorized to change, the results are unpredictable.

To use the exit points, see the following sections, which show the exit number, the data associated with the exit, and the valid actions.

Selecting exit points

To use an exit point, you must know when it occurs in the function, what data is passed, and what actions your program can take at that point. The following table shows when exit points occur. The tables in “[Summary of data parameters and valid actions](#)” on page 114 show the data parameters for each exit program and the actions your exit program can take. Finally, an example shows the order in which the function takes the exit points.

When exit points occur

The following table shows when exit points occur:

Exit	Phase	When exit occurs
1	Initialization	After the function has initialized
2	Termination	After the function has completed all processing
3	Secondary key initialization	Just before the function starts depopulating the secondary key
4	Secondary key termination	After the function completes depopulating a secondary key
8	Secondary key statistics	After the function prints a secondary key statistics line in the base statistics report
9	Index file statistics	After the function prints an index file statistics line in the extended statistics report

At each exit point, your program can indicate whether the function should continue or stop processing. To continue, your program should return blanks. To stop processing, your program should return ABORT.

ABORT has a different scope at different exit points. At exits on the file level (1, 2, 8, and 9), the function stops processing that file and continues with the next file. At exit points on the secondary key level (3 and 4), the function stops processing that secondary key and continues with the next secondary key in the same file.

In addition to continuing or aborting, your program at the first exit point can also change the switches in the data parameter from Y (Yes) to N (No). When your program changes a switch, it turns off the exit point and the function does not take the exit. If your program changes a switch from Y to N, it must also return an action of SET ~~bbbbbb~~. The programs at the other exit points cannot change the data parameter.

Summary of data parameters and valid actions

The following table shows the data parameters for each exit program:

Exit	Use	Data parameter
1	Initialization	The 4-character file name followed by a string of nine Y (YES) or N (NO) switches. Each switch turns on or off its exit point. All switches are initially set to YES.
2	Initialization termination	The 4-character file name followed by a fullword integer containing the function return code (0, 4, 8, or 16)
3	Secondary key initialization	The secondary key name
4	Secondary key termination	The secondary key name
8	Secondary key statistics	The statistics in the detail line in the order they appear in the statistics reports. All numbers are fullword binary integers.
9	Index key statistics	The statistics in the detail line in the order they appear in the statistics reports. All numbers are fullword binary integers.

The following table shows valid actions for exit programs in Depopulate:

			Action indicators		
Exit	Use	Data parameter	b	S	A
1	Set exit points	13 bytes containing a four-byte file name and a string of nine Ys. Your program may change the Y to N to turn off the exit point. If your program changes the setting, it must return SETbbbbbb.	Y	Y	Y
2	Get return code	The file name followed by the return code: 0 (complete), 4 (warning), 8 (error), 16 (internal error)	Y	Y	Y
3	Gather information or add processing	8-character name of secondary key	Y	N	Y
4	Gather information or add processing	8-character name of secondary key	Y	N	Y
5	Does not exist in Depopulate				
6	Does not exist in Depopulate				
7	Does not exist in Depopulate				
8	Get the secondary key's statistics	The secondary key's statistics	Y	N	Y
9	Get the index file's statistics	The index file's statistics	Y	N	Y

Legend: b = bbbbbbbbbb - Action continues
 S = SETbbbbbbb - Data contents changed
 A = ABORTbbbb - Terminates recovery

In this example, the exit points are taken in the following order when you depopulate two secondary keys, *ffffSK01* and *ffffSK02*, that both reside in index file IX01. In the UCL, you requested all statistics.

Exit point number	Point in processing
1	During initialization of the Depopulate function
3	Before the function starts depopulating <i>ffffSK01</i>
4	After the function finishes depopulating <i>ffffSK01</i>
3	Before the function starts depopulating <i>ffffSK02</i>
4	After the function finishes depopulating <i>ffffSK02</i>
8	After the function prints a secondary key statistics line in the base statistics report for <i>ffffSK01</i>
8	After the function prints a secondary key statistics line in the base statistics report for <i>ffffSK02</i>
9	After the function prints an index file statistics line in the extended statistics report for IX01
2	Just before the Depopulate function terminates

Requesting statistics

You receive statistics when you code the following in the function section of the UCL:

```
FUNCTION(DEPOPULATE)  
  STATISTICS(ALL)  
  FILE(PANM)  
    SECONDARY - KEY(ALL)
```

When you request statistics, you receive information on the secondary keys in the files you listed. After depopulating the file, the function prints the following base statistics for each secondary key:

- ◆ The status code (indicating whether depopulation was successful)
- ◆ The index file name
- ◆ The number of blocks in the secondary key before it was depopulated

When you request the extended statistics, the function also prints the following information for each index file:

- ◆ The number of blocks in the file
- ◆ The number of blocks in use before and after depopulation
- ◆ The number of free blocks before and after depopulation

Examples of both the base and extended statistics are in the following two code samples.

Example 1 This example shows base statistics on the Depopulate function.

```

CSUL3759I : BEGINNING OF DEPOPULATE STATISTICS (BASE).
                BLOCKS IN
                SECONDARY
SECONDARY      INDEX KEY BEFORE
KEY NAME  STATUS  FILE  DEPOPULATE
-----
PANMSK01   ****   I001      1,727

PANMSK02   ****   I001      4,270

CSUL3763I : END OF DEPOPULATE STATISTICS (BASE).
    
```

Example 2 This example shows extended statistics on the Depopulate function.

```

CSUL3776I : BEGINNING OF DEPOPULATE STATISTICS (EXTENDED).

                BEFORE DEPOPULATION      AFTER DEPOPULATION
                -----
NUMBER OF      NUMBER OF      NUMBER OF
INDEX  BLOCKS IN  BLOCKS      NUMBER OF  BLOCKS      NUMBER OF
FILE   FILE      IN USE      FREE BLOCKS  IN USE      FREE BLOCKS
-----
I001   7,005      6,160      845        163        6,842

CSUL3781I : END OF DEPOPULATE STATISTICS (EXTENDED).
    
```

7

Coding the Reorganize function

The Reorganize function enables you to correct the deterioration of the secondary key tree structure that can result from updates. The Reorganize function rebuilds the tree structure without accessing the primary or related file from which the secondary key came.

The Reorganize function works like the Sorted-Populate function. That is, the Reorganize function has exclusive access to the file, and opens and closes the file each time you code it in a FILE statement. In addition, you can request the same statistics, add an exit program, and indicate how full you want the index file blocks.

Coding the UCL for the Reorganize function

After you code the control section as shown in “[Coding the control section](#)” on page 57, you can code the Reorganize function as shown in the following format:

```
FUNCTION(REORGANIZE)
  [
    STATISTICS ( [ ALL
                  BASE
                  NONE ] )
    [STANDARD - EXIT (exit - name)]
    FILE ( { ALL
            file - name - list } )...
    [
      SECONDARY - KEY ( [ ALL
                        key - name - list ] )
                        [LOAD - DENSITY (0 - 99)] ] ...
  ]
```

FUNCTION (REORGANIZE)

Description *Required.* Executes the Reorganize function.

Consideration Performance is highly sensitive to the number of buffers in the INDEX file's buffer pool. The recommended number of buffers is twice the expected tree height plus three. If you do not know the expected tree height, 10–15 buffers is recommended. The DATA file buffering is not important because Reorganize does not read the DATA file.

STATISTICS ($\left[\begin{array}{l} \text{ALL} \\ \text{BASE} \\ \text{NONE} \end{array} \right]$)

Restriction If you request statistics, you must code this statement before the FILE statement(s).

Description *Optional.* Indicates the kinds of statistics reports you want generated.

Default BASE

Options ALL You receive statistics reports on both index files and secondary keys.

BASE You receive reports on only the secondary keys.

NONE You do not receive statistics reports.

Consideration You may request statistics only once.

STANDARD-EXIT (*exit-name*)

Restriction If you code the STANDARD-EXIT statement, you must code it before the FILE statement.

Description *Optional.* Indicates you want to use an exit program with this function.

Default The function skips the exit points.

Format 1–8 alphanumeric characters. The first character must be alphabetic.

Considerations

- ◆ You may code this statement only once.
- ◆ If you code the STANDARD-EXIT statement, you must make your exit program available for the function to load. You must put the exit program in your execution library.
- ◆ “[Writing exit programs](#)” on page 124 describes the exit points in this function.

FILE ({ **ALL**
file-name-list }) ...

Description *Required.* Names the database files that need their secondary keys reorganized.

Format File names must be 4 alphanumeric characters. The first character must be alphabetic. Separate names with commas.

Considerations

- ◆ You must code the FILE statement at least once. After that, you can code it as many times as you like.
- ◆ You must code the FILE statement after all the other REORGANIZE statements.
- ◆ You can code only primary and related files.
- ◆ If you code FILE (ALL), the function reorganizes all secondary keys for every primary and related file in the schema and environment description.
- ◆ If you code FILE (ALL), you cannot code any other FILE statements.
- ◆ When the function processes each file, it opens and closes the file. If you code a file in more than one FILE statement, the function opens and closes the file each time.

SECONDARY - KEY ($\left[\begin{array}{l} \text{ALL} \\ \text{key-name-list} \end{array} \right]$)

Restriction To code the SECONDARY-KEY statement, you must code the FILE statement.

Description *Optional.* Indicates the secondary keys you want reorganized.

Default ALL

Format Secondary key names must be 8 alphanumeric characters. Separate names with commas.

Considerations

- ◆ You can code the SECONDARY-KEY statement as many times as you like, or not at all.
- ◆ If you code SECONDARY-KEY (ALL), the function reorganizes all secondary keys for the files you coded in the parent FILE statement.
- ◆ If you code SECONDARY-KEY (ALL), you cannot code any other SECONDARY-KEY statements.
- ◆ You cannot code SECONDARY-KEY (*key-name-list*) if you coded FILE (ALL) or a file list with more than one file name.

LOAD-DENSITY (0–99)

Restriction To code the LOAD-DENSITY statement, you must code the SECONDARY-KEY statement.

Description *Optional.* Indicates how full you want the index file blocks.

Default 0

Options 0–99

Considerations You can code the LOAD-DENSITY statement only once.

- ◆ If you code LOAD-DENSITY (0), the function loads each secondary key to the load density defined for the secondary key on the Directory.
- ◆ If you code a load density between 1 and 99, the function fills the index file blocks as closely as possible to that percentage.

Writing exit programs

With the exit points from the Reorganize function, you can collect data about your secondary keys, turn off exit points, and see statistics.

“[Inserting exit programs into functions](#)” on page 49 explains how your exit programs are loaded, how they operate, the languages you can use to write them, and the register conventions you must follow.

In register 1, for example, you must code the parameter list addresses. The following table describes the parameter list addresses:

Parameter	Data type	Contents before exit (passed to exit program)	Contents after exit (passed from exit program)
Function name	16 bytes character	REORGANIZE	Must be unchanged
Exit point	4 bytes integer	Exit point number	Must be unchanged
Action indicator	8 bytes character	bbbbbbbb	bbbbbbbb or other valid values
Data	Variable	Data associated with exit point data	Same data or changed data if permitted

If your exit program changes anything it is not authorized to change, the results are unpredictable.

To use the exit points, see the following sections, which show the exit number, the data associated with the exit, and the valid actions.

Selecting exit points

To use an exit point, you must know when it occurs in the function, what data is passed, and what actions you can take at that point. The following table shows when exit points occur. “[Summary of data parameters and valid actions](#)” on page 126 shows the data parameters and the actions your program can take. Finally, an example situation shows the order in which the function takes the exit points.

Exit	Phase	When exit occurs
1	Initialization	After the function has initialized
2	Termination	After the function has completed all processing
3	Secondary key initialization	Just before the function starts reorganizing the secondary key
4	Secondary key termination	After the function completes reorganizing a secondary key
8	Secondary key statistics	After the function prints a secondary key statistics line in the base statistics report
9	Index file statistics	After the function prints an index file statistics line in the extended statistics report

At each exit point, your program can indicate whether the function should continue or stop processing. To continue, your program should return blanks. To stop processing, your program should return ABORT.

ABORT has a different scope at different exit points. At exits on the file level (1, 2, 8, and 9), the function stops processing that file and continues with the next file. At exit points on the secondary key level (3 and 4), the function stops processing that secondary key and continues with the next secondary key in the same file.

In addition to continuing or aborting, your program at the first exit point can also change the switches in the data parameter from Y (Yes) to N (No). When your program changes a switch, it turns off the exit point and the function does not take the exit. If your program changes a switch from Y to N, it must also return an action of `SET`. The programs at the other exit points cannot change the data parameter.

Summary of data parameters and valid actions

The following table shows the data parameters for each exit program:

Exit	Use	Data parameter
1	Initialization	The 4-character file name followed by a string of nine Y (yes) or N (no) switches. Each switch turns on or off its exit point. All nine switches are initially set to YES.
2	Termination	The 4-character file name followed by a fullword integer containing the function return code (0, 4, 8, or 16)
3	Secondary key initialization	The secondary key name
4	Secondary key termination	The secondary key name
8	Secondary key statistics	The statistics in the detail line in the order they appear in the statistics reports. All numbers are fullword binary integers.
9	Index key statistics	The statistics in the detail line in the order they appear in the statistics reports. All numbers are fullword binary integers.

The following table shows valid actions for exit programs in Reorganize:

			Action indicators		
Exit	Use	Data parameter	b	S	A
1	Set exit points	13 bytes containing a four-byte file name and a string of nine Ys. Your program may change the Y to N to turn off the exit point. If your program changes the setting, it must return SETbbbbbb.	Y	Y	Y
2	Get return code	The file name followed by the return code: 0 (complete), 4 (warning), 8 (error), 16 (internal error)	Y	N	Y
3	Gather information or add processing	8-character name of secondary key	Y	N	Y
4	Gather information or add processing	8-character name of secondary key	Y	N	Y
5	Does not exist in Reorganize				
6	Does not exist in Reorganize				
7	Does not exist in Reorganize				
8	Get the secondary key's statistics	The secondary key's statistics	Y	N	Y
9	Get the index file's statistics	The index file's statistics	Y	N	Y

Legend: b = bbbbbbbbbb - Action continues
 S = SETbbbbbbb - Data contents changed
 A = ABORTbbbbbb - Terminates recovery

The exit points are taken in the following order when you are reorganizing two secondary keys, *ffffSK01* and *ffffSK02*, that both reside in index file IX01. In the UCL, you requested all statistics.

Exit point number	Point in processing
1	During initialization of the Reorganize function
3	Before the function starts reorganizing <i>ffffSK01</i>
4	After the function finishes reorganizing <i>ffffSK01</i>
3	Before the function starts reorganizing <i>ffffSK02</i>
4	After the function finishes reorganizing <i>ffffSK02</i>
8	After the function prints a secondary key statistics line in the base statistics report for <i>ffffSK01</i>
8	After the function prints a secondary key statistics line in the base statistics report for <i>ffffSK02</i>
9	After the function prints an index file statistics line in the extended statistics report for IX01
2	Just before the Reorganize function terminates

Requesting statistics

You receive statistics when you code the following in the function section of the UCL:

```
FUNCTION(REORGANIZE)  
  STATISTICS(ALL)  
  FILE(PANM)  
    SECONDARY - KEY(ALL)  
    LOAD - DENSITY(80)
```

When you request statistics, you receive information on the secondary keys in the files you listed. After reorganizing the file, the function prints the following base statistics for each secondary key:

- ◆ The status code (whether reorganization was successful)
- ◆ The index file name
- ◆ The number of blocks in the secondary key before it was reorganized
- ◆ The number of key values
- ◆ The number of unique key values
- ◆ The number of blocks in the secondary key before and after reorganization
- ◆ The number of levels in the secondary key before and after reorganization
- ◆ The number of low level blocks in the secondary key before and after reorganization

If you requested extended statistics, the function also prints the following information for each index file:

- ◆ The number of blocks in the file
- ◆ The number of blocks in use before and after reorganization
- ◆ The number of free blocks before and after reorganization

The following two code samples show examples of the base and extended statistics.

Example 1 This example shows base statistics on the Reorganize function.

CSUL3574I : BEGINNING OF REORGANIZE STATISTICS (BASE)

		BEFORE REORGANIZATION				AFTER REORGANIZATION				
		NUMBER OF		BLOCKS IN	LEVELS IN	BLOCKS IN	LOW LEVEL	LEVELS IN	LOW LEVEL	
SECONDARY	INDEX	NUMBER OF	UNIQUE	SECONDARY	SECONDARY	SECONDARY	SECONDARY	SECONDARY	SECONDARY	
KEY NAME	STATUS	FILE	KEY VALUES	KEY	KEY	KEY	KEY	KEY	KEY	
PANMSK01	****	I001	89,989	89,990	1,727	3	1,698	1,727	3	1,698
PANMSK02	****	I001	89,989	89,990	4,270	4	4,091	4,270	4	4,091

CSUL3578I : END OF REORGANIZE STATISTICS (BASE).

Example 2 This example shows extended statistics on the Reorganize function.

CSUL3591I : BEGINNING OF REORGANIZE STATISTICS (EXTENDED).

		BEFORE REORGANIZATION			AFTER REORGANIZATION	
		NUMBER OF	NUMBER OF	NUMBER OF		
INDEX	BLOCKS IN	BLOCKS	NUMBER OF	BLOCKS	NUMBER OF	
FILE	FILE	IN USE	FREE BLOCKS	IN USE	FREE BLOCKS	
I001	7,005	6,160	845	6,160	845	

CSUL3596I : END OF REORGANIZE STATISTICS (EXTENDED).

8

Coding the File Statistics function

After you have used your database files, you may want to use the File Statistics function. With it, you can get reports on various physical and logical characteristics of the database files that are in the SUPRA native format.



You cannot get meaningful information about index files. For this information, cyclically use the Execution Statistics function and analyze the index file results.

With File Statistics reports, you can monitor file growth and predict expansion needs. You may also verify linkages and monitor the time needed to access records. If you use the File Statistics function on a cyclical basis, you can optimize file performance.



If you request statistics on key-sequenced data sets or entry-sequenced data sets, you may receive no report or only partial reports because the information is not available to the function. For more information on the reports you can receive, see the samples in [“Requesting file statistics”](#) on page 137.

When you request statistics on linkpaths or chains, the File Statistics function sorts the information before printing it. Therefore, you must allocate sort work space for the sort. To calculate the amount of sort work space, see [“Using sort programs”](#) on page 52.

Coding the UCL for the File Statistics function

After you code the control section as shown in “[Coding the control section](#)” on page 57, you can code the File Statistics function according to the following format. For information on work files and JCL, refer to the [SUPRA Server PDM and Directory Administration Guide \(OS/390 & VSE\)](#), P26-2250.

FUNCTION (FILE - STATS)

FILE ({ ALL
file-name }) ...

[CLOSE([NO
YES])]

[LINKPATH ([**b**
linkpath-list])]

[STATISTICS ([ALL] [BASE] [SIZE] [LINK] [CHAIN] [CODE])]

FUNCTION (FILE-STATS)

Description *Required.* Invokes the File Statistics function.

FILE ({ **ALL**
 file-name }) ...

Description *Required.* Names the database file(s) on which you want to see statistics.

Format 4 alphanumeric characters. The first character must be alphabetic.

Options ALL You receive statistics on all the primary and related files in the schema. You cannot get statistics on index files, Task Log Files, or System Log Files.

file-name You receive statistics on the specified file.

Considerations

- ◆ If you code FILE (ALL), you receive the statistics in alphabetical order by file with primary files first and then related files. A message indicating that you do not receive statistics for index files appears before the primary files.
- ◆ If you want statistics on Directory files, you must code Directory files by name. You cannot get statistics on Directory files by coding FILE (ALL).
- ◆ The file names you code must exist in the schema you coded in the control section.
- ◆ Do not code FILE (*file-name-list*).
- ◆ If you name individual files, you must code each one in a separate FILE statement. They are processed in the order in which you list them.
- ◆ You may code only primary or related files.

CLOSE(

NO
YES

)

Description *Optional.* Specifies whether to explicitly close the file after the File Statistics function reads it.

Default YES

LINKPATH (

<u>b</u>
<i>linkpath-list</i>

)

Description *Optional.* Indicates the linkpaths on which you want to gather statistics.

Format Linkpath names must be 8 alphanumeric characters. Separate names with commas.

Default b

Options b Gathers statistics on all linkpaths for primary and noncoded related files. You must use only base linkpaths for coded related files.

linkpath-list Gathers statistics on linkpaths you name.

Consideration You must code at least one of the linkpaths in the overlay portion of the record if you request statistics on a coded, related file that does not contain a linkpath in the base portion of the record.

STATISTICS ([ALL] [BASE] [SIZE] [LINK] [CHAIN] [CODE])

Description	<i>Optional.</i> Indicates the content and format of the statistics report.	
Default	BASE	
Options	ALL	You receive all possible statistics.
	BASE	You receive a report of basic file information. For an example, see “ Requesting Basic File Information (BASE) ” on page 138.
	SIZE	You receive a report on current file size. For an example, see “ Requesting Current File Size (SIZE) ” on page 139.
	LINK	You receive a report of linkpath statistics. For an example, see “ Requesting Linkpath Statistics (LINK) ” on page 140.
	CHAIN	You receive a report on chain length statistics and chain migration statistics for primary or related files, plus a report on synonym statistics for primary files only. For an example, see “ Requesting Chain Statistics (CHAIN) ” on page 141.
	CODE	You receive a report on record code statistics for coded, related files. For an example, see “ Requesting Record Code Statistics (CODE) ” on page 146.

Considerations

- ◆ If you name a noncoded file in the FILE statement, STATISTICS (CODE) is ignored.
- ◆ If you name a primary file which has no linkpath elements, STATISTICS (LINK) is ignored.
- ◆ The File Statistics function prints generated reports in the same order regardless of the order of the options you code in the STATISTICS statement.

Programming examples

Example 1 The code in this example will produce the following reports for all the files in the schema called (CINDIREV): basic file information, current file size, chain length and migration statistics, and linkpath statistics. This code also produces extended diagnostics and uses a sort program named SORT.

```
CONTROL (BEGIN)
    ENV-DESC (CINDIREV)
        SCHEMA (CINDIRSC)
DIAGNOSTICS (EXTENDED)
SORT (SORT)
FUNCTION (FILE-STATS)
FILE (ALL)
    STATISTICS (BASE SIZE CHAIN LINK)
CONTROL (END)
```

Example 2 The following example produces Current File Size and Linkpath Statistics reports for only a few files and linkpaths:

```
CONTROL (BEGIN)
    ENV-DESC (CINDIREV)
        SCHEMA (CINDIRSC)
FUNCTION (FILE-STATS)
    FILE (C$-S)
        LINKPATH (C$-#LKST, C$-#LKHD, C$-#LKWV)
        STATISTICS (SIZE LINK)
    FILE (C$-#)
        LINKPATH (C$-#LKST)
        STATISTICS (SIZE LINK)
CONTROL (END)
```

Requesting file statistics

When you use the File Statistics function, you get a printout containing a title page and the reports you requested. The following figure is a sample title page. The remaining figures in this chapter show examples of the reports you receive.

When you receive the printout, you can check the title page to see whether you received the statistics you wanted and whether coding errors have occurred. The title page shown below indicates that you received the following types of statistics on the C\$-D file: basic, size, linkpath, chain, and record code.

```

FUNCTION = FILE-STATISTICS                                FILE = C$-D
DDDDDDDD  BBBB BBBB      AAA
DDDDDDDD  BBBB BBBB      AAAAA
DD  DD  BB  BB  AA  AA
DD  DD  BB  BB  AA  AA
DD  DD  BBBB BBBB  AA  AA
DD  DD  BBBB BBBB  AAAAAAA
DD  DD  BB  BB  AAAAAAA
DD  DD  BB  BB  AA  AA
DD  DD  BB  BB  AA  AA
DDDDDDDD  BBBB BBBB  AA  AA
DDDDDDDD  BBBB BBBB  AA  AA

SSSSSSS  TTTTTTT  AAA  TTTTTTT  IIIIII  SSSSSSS  TTTTTTT  IIIIII  CCCCCC  SSSSSS
SSSSSSSS TTTTTTTT  AAAAA  TTTTTTT  IIIIII  SSSSSSSS  TTTTTTTT  IIIIII  CCCCCCCC  SSSSSSSS
SS  SS  TT  AA  AA  TT  II  SS  SS  TT  II  CC  CC  SS  SS
SS  TT  AA  AA  TT  II  SS  TT  II  CC  SS
SSSSSSS  TT  AA  AA  TT  II  SSSSSSSS  TT  II  CC  SSSSSSSS
SSSSSSSS TT  AAAAAAAA  TT  II  SSSSSSSS  TT  II  C  SSSSSSSS
SS  SS  TT  AAAAAAAA  TT  II  SS  TT  II  CC  SS
SS  SS  TT  AA  AA  TT  II  SS  SS  TT  II  CC  CC  SS  SS
SSSSSSSS TT  AA  AA  TT  IIIIII  SSSSSSSS  TT  IIIIII  CCCCCCCC  SSSSSSSS
SSSSSSS  TT  AA  AA  TT  IIIIII  SSSSSS  TT  IIIIII  CCCCCC  SSSSSS

FILE NAME          C$-D
ENV-DESC NAME      CINDIREN
SCHEMA NAME        CINDIRSC
STATISTICS          BASE SIZE LINK CHAIN CODE
    
```

Requesting Basic File Information (BASE)

To receive the Basic File Information report, code BASE on the STATISTICS statement in your UCL. This report is a summary of physical and logical characteristics for primary or related files in the schema you indicated (see the following figure).

For key-sequenced data sets, you receive no information on block size or record capacity. For files using the BDAM access method, the control interval size is not available.

```

FUNCTION = FILE-STATISTICS                                FILE = C$-#
CCCCCCC $                                               # #
CCCCCCCCC $$$$$$                                       # #
CC CC $$ $ $$                                         # #
CC      $$ $ $$                                         #####
CC      $$$$$$ ----- # #
CC      $$$$$$ ----- # #
CC      $$ $ $$                                         #####
CC CC  $$ $ $ $                                         # #
CCCCCCCCC $$$$$$                                       # #
CCCCCCC $                                               # #

FILE TYPE = PRIMARY

      B A S I C   F I L E   I N F O R M A T I O N

      SCHEMA NAME                                CINDIRSC
      ACCESS METHOD                                BDAM

      LOGICAL RECORD LENGTH                       374
      BLOCKSIZE                                   4488
      CONTROL INTERVAL SIZE                       N/A
      LOGICAL RECORDS PER BLOCK                   12
      LOGICAL BLOCKS IN FILE                      600

      MAXIMUM DATA RECORDS                       7199
      CONTROL RECORDS                             1
      TOTAL LOGICAL RECORDS                       7200

NOTE - SOME ITEMS MAY NOT BE AVAILABLE DEPENDING ON ACCESS METHOD.
    
```

Requesting Current File Size (SIZE)

To receive the Current File Size report, code SIZE on the STATISTICS statement. The report monitors the growth of primary and related files to determine when you need to expand them. The report prints record and block statistics. The following figure shows statistics on the primary file C\$-#.

The File Statistics function calculates the average data records per block in the entire file and the average data records per block in the blocks with data records. The report shows those averages below the statistics. For key-sequenced data sets, the only size statistics available are the numbers of active data records, control records, and records in use.

```

FUNCTION = FILE-STATISTICS                               FILE = C$-#
CCCCCCC          $                                     # #
CCCCCCCCC      $$$$$$                                # #
CC      CC     $$ $ $$                                # #
CC           $$ $ $$                                #####
CC           $$$$$$ -----                          # #
CC           $$$$$$ -----                          # #
CC           $$ $ $$                                #####
CC      CC     $$ $ $$                                # #
CCCCCCCCC      $$$$$$                                # #
CCCCCCC          $                                     # #
FILE TYPE = PRIMARY

      C U R R E N T   F I L E   S I Z E
                                     ACTUAL          % OF
                                     NUMBER          FILE
                                     CAPACITY

RECORD STATISTICS
  ACTIVE DATA RECORDS              2627            36.486
  CONTROL RECORDS                    1             0.014
  RECORDS IN USE                     2628            36.500
  UNUSED RECORDS                     4572            63.500
  TOTAL LOGICAL RECORDS             7200            100.000

BLOCK STATISTICS
  EMPTY BLOCKS                       4             0.667
  BLOCKS IN USE                      596            99.333
  FULL BLOCKS                         1             0.167
  LOGICAL BLOCKS IN FILE             600            100.000
AVERAGE DATA RECORDS/BLOCK IN ENTIRE FILE          4.378
AVERAGE DATA RECORDS/BLOCK IN BLOCKS WITH DATA RECORDS 4.408
NOTE - SOME ITEMS MAY NOT BE AVAILABLE DEPENDING ON ACCESS METHOD.

```

Requesting Linkpath Statistics (LINK)

To receive the Linkpath Statistics report, code LINK on the STATISTICS statement. You can receive linkpath statistics on both primary and related files.

The following figures are examples of reports for a primary file and a related file. Use this report to verify the accuracy of linkages for the files you select. If you have key-sequenced data sets, you do not receive statistics on file capacity or maximum data records. For a coded related file, you also receive the number of records for each record code on each linkpath.

```

FUNCTION = FILE-STATISTICS                                FILE = C$-#
                CCCCCC      $                                # #
                CCCCCCCC    $$$$$$                        # #
                CC   CC    $$ $ $$                          # #
                CC     $$ $ $$                             #####
                CC     $$$$$$    -----                  # #
                CC     $$$$$$    -----                  # #
                CC     $$ $ $$                             #####
                CC   CC    $$ $ $$                          # #
                CCCCCCCC    $$$$$$                        # #
                CCCCCC      $                                # #

FILE TYPE = PRIMARY

                L I N K P A T H   S T A T I S T I C S
LINKPATH      RECORDS WITH   % OF   % OF   RECORDS WITH   % OF   % OF
                ACTIVE         ACTIVE  FILE   NON-ACTIVE   ACTIVE  FILE
                LINKPATH      DATA RECORDS  CAPACITY  LINKPATH  DATA RECORDS  CAPACITY
C$-#LKHD      1259           47.925   17.489   1368       52.075   19.003
C$-#LKST      1259           47.925   17.489   1368       52.075   19.003
C$-#LKWU      2453           93.376   34.074   174        6.624    2.417
C$-#LKDA      767           29.197   10.654   1860       70.803   25.837
C$-#LKTT      1262           48.040   17.530   1365       51.960   18.961

ACTIVE DATA RECORDS                2627                2627
MAXIMUM DATA RECORDS                7199                7199

NOTE - SOME ITEMS MAY NOT BE AVAILABLE DEPENDING ON ACCESS METHOD.
    
```

The following figure shows a related file:

```

FUNCTION = FILE-STATISTICS                                FILE = C$-S
                CCCCCC      $                                SSSSSSS
                CCCCCCCC    $$$$$$                        SSSSSSSSS
                CC   CC    $$ $ $$                          SS   SS
                CC     $$ $ $$                             SS
                CC     $$$$$$    -----                  SSSSSSS
                CC     $$$$$$    -----                  SSSSSSS
                CC     $$ $ $$                             SS   SS
                CC   CC    $$ $ $$                          SS   SS
                CCCCCCCC    $$$$$$                        SSSSSSSSS
                CCCCCC      $                                SSSSSSS

FILE TYPE = RELATED

                L I N K P A T H   S T A T I S T I C S
T O T A L S   R E C O R D   N U M B E R   % O F   % O F
                FOR         OF           DATA RECORDS  CAPACITY
LINKPATH      CODE      RECORDS
C$- #LKST     TOTAL     7392           100.000       23.072
              DT        6020           81.439        18.790
              HD        1372           18.561        4.282

ACTIVE DATA RECORDS                7392
MAXIMUM DATA RECORDS                32039

NOTE - SOME ITEMS MAY NOT BE AVAILABLE DEPENDING ON ACCESS METHOD.
    
```

Requesting Chain Statistics (CHAIN)

Code CHAIN on the STATISTICS statement to receive the following reports:

- ◆ Chain Length Statistics report (for primary or related files)
- ◆ Chain Migration Statistics report (for primary or related files)
- ◆ Synonym Statistics report (for primary files only)

Requesting Chain Length Statistics on primary files

When you receive the Chain Length Statistics report for primary files, you get the number of records randomized to the same home location. With this report, you can monitor the physical structure of chains and their accessing characteristics. In this report, the number in chain value is the actual number of records chained together. The following figure is an example of the report:

FUNCTION = FILE-STATISTICS		FILE = C\$-N	
CCCCCCC	\$	N	NN
CCCCCCCC	\$\$\$\$\$\$	NN	NN
CC CC	\$ \$ \$ \$	NNN	NN
CC	\$ \$ \$ \$	NNNN	NN
CC	\$\$\$\$\$	NN NN	NN
CC	\$\$\$\$\$	NN NN NN	
CC	\$ \$ \$ \$	NN	NNNN
CC CC	\$ \$ \$ \$	NN	NNNN
CCCCCCCC	\$\$\$\$\$\$	NN	NN
CCCCCC	\$	NN	N
FILE TYPE = PRIMARY			
C H A I N L E N G T H S T A T I S T I C S			
RECORDS RANDOMIZED TO SAME HOME LOCATION			
NUMBER	NUMBER	% OF	
IN	OF	TOTAL	
CHAIN	CHAINS	CHAINS	
1	1879	84.336	
2	305	13.689	
3	40	1.795	
4	3	0.135	
5	1	0.045	
6	0	0.000	
7	0	0.000	
8	0	0.000	
9	0	0.000	
10	0	0.000	
OVER	10	0.000	
	TOTAL CHAINS	2228	
CHAIN LENGTH -	MINIMUM	1	
	MAXIMUM	5	
	AVERAGE	1.179	

Requesting Chain Length Statistics on related files

When you request the Chain Length Statistics report for related files, you get the number of records on linkpath chains. You get one report for each linkpath you select.

The number in chain value is the actual number of records chained together. The chain lengths are reported for ranges that are tailored to your particular linkpaths based on the distribution of chain lengths in the file. At least 80% of all chains fall into the defined range; the rest are under the minimum and over the maximum. For the report to be accurate, you must make the keys to linkpaths on related files 245 bytes or less. The following figure is an example of the report:

```

FUNCTION = FILE-STATISTICS                                FILE = C$-S
CCCCCCC $                SSSSSSS
CCCCCCCCC $$$$$$$$    SSSSSSSSS
CC      CC $$ $ $$      SS      SS
CC      $$ $ $$        SS
CC      $$$$$$    ----- SSSSSSSS
CC      $$$$$$    ----- SSSSSSSS
CC      $$ $ $$      SS      SS
CC      CC $$ $ $$    SS      SS
CCCCCCCCC $$$$$$$$    SSSSSSSSS
CCCCCCC $                SSSSSSS

FILE TYPE = RELATED
      C H A I N   L E N G T H   S T A T I S T I C S
LINKPATH = C$-#LKST
      NUMBER          NUMBER          % OF
      IN             OF              TOTAL
      CHAIN          CHAINS          CHAINS
UNDER
  2 -              2              0              0.000
 44 -              43             1243            98.729
 86 -              85              2              0.159
128 -             127              8              0.635
170 -             169              1              0.079
212 -             211              2              0.159
254 -             253              0              0.000
296 -             295              0              0.000
338 -             337              2              0.159
380 -             379              0              0.000
OVER              421              0              0.000
      TOTAL CHAINS          1259
CHAIN LENGTH -  MINIMUM          2
                MAXIMUM         425
                AVERAGE         5.871
    
```

Requesting Chain Migration Statistics on primary files

When you receive the Chain Migration Statistics report for primary files, you get the number of block boundaries traversed and the number of different blocks encountered. The number of block boundaries traversed may equal or exceed the number of blocks encountered. A block traverses more than one boundary if a block contains many noncontiguous records on the same chain. Statistics are not returned for the number of different blocks encountered.

The following figure is an example of the report for a primary file.

```

FUNCTION = FILE-STATISTICS                                FILE = C$-#
                CCCCCC      $                # #
                CCCCCCCC    $$$$$$          # #
                CC      CC  $$ $ $$         # #
                CC      $$ $ $$           #####
                CC      $$$$$$          # #
                CC      $$$$$$          # #
                CC      $$ $ $$           #####
                CC      CC  $$ $ $$         # #
                CCCCCCCC    $$$$$$          # #
                CCCCCC      $                # #
FILE TYPE = PRIMARY
                C H A I N   M I G R A T I O N   S T A T I S T I C S
NUMBER OF BLOCK BOUNDARIES TRAVERSED      NUMBER OF DIFFERENT BLOCKS ENCOUNTERED
NUMBER OF          NUMBER          % OF          NUMBER          NUMBER          % OF
BLOCK             OF              TOTAL          OF              OF              TOTAL
BOUNDARIES       CHAINS           CHAINS      BLOCKS          CHAINS          CHAINS
                1             2             0.091          1             0             0.000
                2             0             0.000          2             0             0.000
                3             0             0.000          3             0             0.000
                4             0             0.000          4             0             0.000
                5             0             0.000          5             0             0.000
                6             0             0.000          6             0             0.000
                7             0             0.000          7             0             0.000
                8             0             0.000          8             0             0.000
                9             0             0.000          9             0             0.000
                10            0             0.000          10            0             0.000
OVER             10            0             0.000          OVER          10            0             0.000
                TOTAL CHAINS          2193          TOTAL CHAINS          2193
    
```

Requesting Chain Migration Statistics on related files

When you receive the Chain Migration Statistics report for related files, you get the same statistics that you get for primary files. You receive one report for each linkpath you code.

The numbers are reported for ranges that are tailored to your particular linkpaths based on the number of blocks in the linkpath's chains. At least 80% of all chains fall into the defined range; the rest are under the minimum and over the maximum. The following figure shows an example of a report for a related file.

Statistics are not returned for the number of different blocks encountered:

FILE TYPE = RELATED						
CHAIN MIGRATION STATISTICS						
LINKPATH = C\$-#LKST						
NUMBER OF BLOCK BOUNDARIES TRAVERSED			NUMBER OF DIFFERENT BLOCKS ENCOUNTERED			
NUMBER OF BLOCK BOUNDARIES	NUMBER OF CHAINS	% OF TOTAL CHAINS	NUMBER OF BLOCKS	NUMBER OF CHAINS	% OF TOTAL CHAINS	
1	13	1.033	1	0	0.000	
2	1	0.079	2	0	0.000	
3	3	0.238	3	0	0.000	
4	0	0.000	4	0	0.000	
5	0	0.000	5	0	0.000	
6	1	0.079	6	0	0.000	
7	0	0.000	7	0	0.000	
8	0	0.000	8	0	0.000	
9	0	0.000	9	0	0.000	
10	0	0.000	10	0	0.000	
OVER	10	0.000	OVER	10	0.000	
TOTAL CHAINS		1259	TOTAL CHAINS		1259	
AVERAGE NUMBER OF READS TO TRAVERSE ENTIRE CHAIN			- 1.024			

Requesting Synonym Statistics on primary files

When you request the Synonym Statistics report, you get the actual number of records in a file and the percentage of capacity. With this report, you can monitor the number and length of synonym chains in primary files.

You also receive the average number of physical reads to obtain a record. The average is calculated as follows: (active data records + number of records not in home block)/active data records. For an example, use the numbers in the following figure. Add the active data records, 2626, to the number of records not in home block, 0, and divide by the active data records, 2626. The result is 1.

FUNCTION = FILE-STATISTICS		FILE = C\$-N	
CCCCCCC	\$	N	NN
CCCCCCCC	\$\$\$\$\$\$\$	NN	NN
CC CC	\$ \$ \$ \$	NNN	NN
CC	\$ \$ \$ \$	NNNN	NN
CC	\$\$\$\$\$\$\$	-----	NN NN NN
CC	\$\$\$\$\$\$\$	-----	NN NN NN
CC	\$ \$ \$ \$	NN	NNNN
CC CC	\$ \$ \$ \$	NN	NN
CCCCCCCC	\$\$\$\$\$\$\$	NN	NN
CCCCCCC	\$	NN	N
FILE TYPE = PRIMARY			
SYNONYM STATISTICS			
		ACTUAL	% OF
		NUMBER	FILE
RECORDS AT HOME LOCATION		2228	31.473
RECORDS NOT AT HOME LOCATION		398	5.622
RECORDS IN HOME BLOCK		2626	37.
RECORDS NOT IN HOME BLOCK		0	0.000
ACTIVE DATA RECORDS		2626	
MAXIMUM DATA RECORDS			7079
AVERAGE NUMBER OF PHYSICAL READS TO OBTAIN A RECORD		1.000	

Requesting Record Code Statistics (CODE)

To receive the Record Code Statistics report, include CODE on the STATISTICS statement. You receive statistics on only the coded files. If you included noncoded files in your UCL, the function ignores CODE for those files. With this report, you get the number of coded records, their percentage of active records, and the percentage of total file capacity they take up. The following figure is an example of this report:

FUNCTION = FILE-STATISTICS		FILE = C\$-T	
CCCCCC	\$	TTTTTTT	
CCCCCCCC	\$\$\$\$\$\$	TTTTTTT	
CC	CC \$ \$ \$ \$	TT	
CC	\$ \$ \$ \$ \$	TT	
CC	\$\$\$\$\$	TT	
CC	\$\$\$\$\$	TT	
CC	\$ \$ \$ \$ \$	TT	
CC	CC \$ \$ \$ \$ \$	TT	
CCCCCCCC	\$\$\$\$\$\$	TT	
CCCCCC	\$	TT	
FILE TYPE = RELATED			
	R E C O R D	C O D E	S T A T I S T I C S
RECORD	NUMBER	% OF	% OF
		OF	FILE
CODE	RECORDS	DATA RECORDS	CAPACITY
LT	515	28.981	4.471
ST	1262	71.019	10.956
	ACTIVE DATA RECORDS	1777	
	MAXIMUM DATA RECORDS		11519
	NOTE - SOME ITEMS MAY NOT BE AVAILABLE DEPENDING ON ACCESS METHOD.		

9

Coding the Expand function

When the database files are too full for acceptable performance, you need to enlarge them. To do this, you can choose from several functions depending on the type and format of the file. The Unload and Load functions enlarge both primary and related files. If you want to change the file's format to SUPRA native format, you must use the Version 1 functions described in [“Coding the Version 1 Unload and Load functions”](#) on page 153. If you want to leave the files in the same format, you must use the Version 2 functions described in [“Coding the Version 2 Unload, Load, and Insert Linkpath functions”](#) on page 225.

The Expand function enlarges only related files that are already in SUPRA native format. It copies the file as it currently exists to the new location, adds additional space to the end of the file, and formats the new space to blanks.

Because the new space is added at the end, a disproportionate share of the data is in the front of the file. Since this may affect performance, use the Expand function only when absolutely necessary.



If you use the Expand function with any other function, you should code it before the others

After you code the control section as shown in [“Coding the control section”](#) on page 57, you can code the Expand function as shown in the following format and example.

FUNCTION (EXPAND)

**FILE ({ ALL
 { file-name-list } }) ...**

FUNCTION (EXPAND)

Description *Required.* Indicates you want to expand the capacity of a SUPRA related file.

Considerations

- ◆ You can only add space. You cannot copy a file by coding the same old and new total logical records.
- ◆ You need to use two schemas: one in the REALM parameter in the CSIPARM file and one in the UCL. In the schema in the REALM parameter, you must put a description of the file as it currently exists. In the schema and environment description you include in the UCL, you must put the description of the file after expansion.
- ◆ You need to code two data sets (the old data set and a new data set) with two different ddnames. You must prefix the ddname of the old data set with an O. For example, if you want to expand the CUST file and the ddname for this file is CUSTWXYZ, then you change the ddname for the old data set to OCUSTWXY.

Since the eighth character is dropped when you add the O, you cannot expand in the same job step two files whose ddnames differ only in the eighth character.

- ◆ **VSE** In VSE, indicate that access to BDAM files is sequential direct (SD). (VSAM is automatically sequential direct.)
- ◆ To expand Directory files, you must code the schema and environment description in the REALM parameter in the CSIPARM file. You can create a second bootstrap schema by copying CSTASCHM, renaming it, and then running the Modify Schema utility against the renamed bootstrap schema. You must regenerate the environment descriptions with the new bootstrap schema name in the UCL. You must also regenerate the Valmod. While you code the new bootstrap schema in the UCL, you code the old bootstrap schema in the CSIPARM file.
- ◆ Do not format ESDS files prior to running the Expand function.

FILE ({ **ALL**
file-name-list }) ...

Description *Required.* Names the database files you want expanded.

Format File names must be 4 alphanumeric characters. The first character must be alphabetic. Separate names with commas.

Considerations

- ◆ If you code FILE (ALL), the function expands all related files in alphabetical order.
- ◆ You may code any number of FILE statements following each FUNCTION command.
- ◆ The file names you code must be in the schema you included in the control section.
- ◆ You cannot code FILE (ALL) for Directory files. You must code Directory Files by name.

Example 1 The following example expands the CUST, ACCR, PORT and VEND files. Note that the file names must be in the schema included in the control section.

CONTROL (BEGIN)	<i>Initiates UCL program.</i>
ENV-DESC (MYDESC)	<i>Names environment description.</i>
SCHEMA (MYSHEMA)	<i>Names schema.</i>
FUNCTION (EXPAND)	<i>Invokes the Expand function.</i>
FILE (CUST)	<i>Names files to be expanded.</i>
FILE (ACCR, PORD, VEND)	<i>Names files to be expanded.</i>
CONTROL (END)	<i>Terminates program.</i>

OS/390 In OS/390, the following ddnames would be defined:

ddname	File description
//CUSTWXYZ DD DSN=new file name CUST	<i>Expanded file with name CUST.</i>
//OCUSTWXY DD DSN=old file name CUST	<i>Old file with name CUST.</i>
//ACCRWXYZ DD DSN=new file name ACCR	<i>Expanded file with name ACCR.</i>
//OACCRWXY DD DSN=old file name ACCR	<i>Old file with name ACCR.</i>
//PORDWXYZ DD DSN=new file name PORD	<i>Expanded file with name PORD.</i>
//OPORDWXY DD DSN=old file name PORD	<i>Old file with name PORD.</i>
//VENDWXYZ DD DSN=new file name VEND	<i>Expanded file with name VEND.</i>
//OVENDWXY DD DSN=old file name VEND	<i>Old file with name VEND.</i>

VSE In VSE, you would code the following:

ddname	File description
// DLBL CUSTXYZ, 'DATA_QUAL.CUST', 0, SD	<i>Expanded file with name CUST.</i>
// DLBL OCUSTXY, 'DATA_QUAL.OCUST', 0, SD	<i>Old file with name CUST.</i>
// DLBL ACCRXYZ, 'DATA_QUAL.ACCR', 0, SD	<i>Expanded file with name ACCR.</i>
// DLBL OACCRXY, 'DATA_QUAL.OACCR', 0, SD	<i>Old file with name ACCR.</i>
// DLBL PORDXYZ, 'DATA_QUAL.PORD', 0, SD	<i>Expanded file with name PORD.</i>
// DLBL OPORDXY, 'DATA_QUAL.OPORD', 0, SD	<i>Old file with name PORD.</i>
// DLBL VENDXYZ, 'DATA_QUAL.VEND', 0, SD	<i>Expanded file with name VEND.</i>
// DLBL OVENDXY, 'DATA_QUAL.OVEND', 0, SD	<i>Old file with name VEND.</i>

Example 2 This example shows the code and the listing that you receive after the code is validated and executed.

```

CSUL0101I : COMMENCING COMMAND VALIDATION.
1 CONTROL(BEGIN)
2 *****
3 *
4 * EXPAND EXAMPLE #1 DESCRIPTION
5 *
6 * OBJECTIVE: EXPAND AN EXISTING RELATED FILE.
7 *
8 * NOTES:
9 *
10 * 1. THE SCHEMA AND ENV-DESC SPECIFIED IN THE UCL
11 * CONTAINS THE DESCRIPTION OF THE NEW FILE FOR
12 * EXPAND.
13 *
14 * 2. CSIPARM CONTAINS THE FOLLOWING PARAMETER
15 * WHICH DESCRIBES THE FILE AS IT IS CURRENTLY:
16 *
17 * REALM= (SCHEMA=XXXXXXXX, ENVDESC=XXXXXXXX)
18 *
19 *
20 *****
21 ENV-DESC(UTED00US)
22 SCHEMA(UTILSC)
23 FUNCTION(EXPAND)
24 FILE(R002)
25 CONTROL(END)
CONTENTS OF SOURCE LINES OUTSIDE 1..72 MARGINS IGNORED.
0 SYNTAX ERRORS DETECTED.
25 COMMAND LINES READ.
1 CONTROL SECTIONS ANALYZED.
1 FUNCTION COMMANDS ANALYZED. CSUL0102I : COMMENCING COMMAND EXECUTION.
CSUL0301I : COMMENCING CONTROL SECTION USING ENVIRONMENT DESCRIPTION UTED00US AND SCHEMA UTILSC .
CSUL0302I : COMMENCING EXPAND PROCESS.
CSUL0311I : COMMENCING EXPAND AGAINST FILE R002.
CSUL1102I : FILE R002 IS NOW EXPANDED.
CSUL0321I : EXPAND PROCESSING AGAINST FILE R002 TERMINATING NORMALLY.
CSUL0303I : EXPAND PROCESS TERMINATING.
CSUL0305I : CONTROL SECTION TERMINATING.
CSUL0307I : ALL CONTROL SECTIONS PROCESSED.
CSUL0103I : DATABASE UTILITIES SUCCESSFUL TERMINATION.

```


10

Coding the Version 1 Unload and Load functions

The Version 1 Unload function extracts records from a database file and writes them to a sequential output file. It can extract records from files in Series 80, SUPRA converted, or SUPRA native format. The Unload function then builds the records in the output file in a format compatible with the Load function, so that you can reload them.

The Version 1 Load function copies the records from the sequential file to database files. Before copying the records, the Load function sorts them. Therefore, you need to allocate work space for the sort program. To calculate the amount of work space, see [“Allocating sort work space”](#) on page 54.

The Load function formats records in the SUPRA native format but does not, however, format in Series 80 or SUPRA converted formats. If you want files in these formats, you must use the Version 2 Unload and Load functions described in [“Coding the Version 2 Unload, Load, and Insert Linkpath functions”](#) on page 225.

You can use the Load function to create new files. Since the files do not yet exist, you do not execute the Unload function. Instead, you create a sequential file that looks as if the Unload function created it. For the format of the file, see [“Retaining the format of the data file”](#) on page 184.

If your database files have secondary keys, you must depopulate them either before or after unloading the files. The example in [“Examples of Unload, Load, and Modify functions”](#) on page 191 shows the secondary keys being depopulated before unloading the files. However, if you are unloading files to obtain a backup copy, you must depopulate before you load.

To depopulate the secondary keys, use the Depopulate function in “Coding the Depopulate function” on page 105 or the Directory Maintenance DEPOPULATE command with the Remove parameter. After you have reloaded the files, you can repopulate them with the Sorted-Populate function in “Coding the Sorted-Populate function” on page 91 or the Directory Maintenance POPULATE command. For details on the commands, refer to the *SUPRA Server PDM Directory Online User’s Guide (OS/390 & VSE)*, P26-1260, or the *SUPRA Server PDM Directory Batch User’s Guide (OS/390 & VSE)*, P26-1261.

When you code your JCL to unload and load files with secondary keys, you must include the index files because the Unload and Load functions open the index files when they open the associated data files. However, the functions only open the index files; the functions do not process the index files if they are depopulated.

If you do not depopulate before you load, the Load function reloads a duplicate set of the secondary keys. If there is not enough space in the index file for the duplicate keys, the PDM abends. If there is enough space for the duplicate set, you receive error messages indicating invalid chains. To solve the problem, simply depopulate and repopulate. You do not need to unload and load again.



If performance is critical, use the Version 2, Unload, Load, and Insert Linkpath functions. Version 2 functions are certified for OS/390 and VSE only.

The Version 2 functions are not compatible with these Version 1 functions. The only input you can use with the Version 1 Load function is the output of Version 1 Unload function or a program you code.

Coding the UCL for the Unload function

After you code the control section as shown in “Coding the control section” on page 57, you can code the Unload function as shown in the following format. However, if you have no files and need to load files first, see the Load function’s format in “Coding the UCL for the Load function” on page 169. For UCL examples to unload and load, see “Examples of Unload, Load, and Modify functions” on page 191.

FUNCTION (UNLOAD)

[STANDARD – EXIT (*exit – name*)]

FILE ({ ALL
file – name }) ...

[LINKPATH ({ b
access – linkpath })
 PRESERVE ({ NO
YES })]

[CLEAR – LINKS (*linkpath – list*)]

[RRN – RANGE ({ low – rrn
 – high – rrn
low – rrn – high – rrn })]

[CRITERIA (*element1* [, *element2*, ..., *elementn*]

. *operator*. *datavalue1* [*datavalue2*... *datavalue**n*]

END.)]

[RECORD ({ ALL
record – code })
 ELEMENT ({ ALL
element – list })] ...

FUNCTION (UNLOAD)

Description *Required.* Invokes the Unload function.

Considerations

- ◆ If you code the Unload and Load functions in the same UCL program, you must code the Unload function first.
- ◆ You may code the Unload function only once in any UCL program. However, you may code as many files to be unloaded as you like.
- ◆ You must match the position and length of data in element lists you use for the Unload and Load functions. If the element lengths in the schema you use to load the file do not match those in the Unload function, use `*FILL=nn` to make their lengths equal. For more detailed information and an example of how to use this parameter, see the considerations under the ELEMENT statement.
- ◆ If you want to change the description of a file in OS/390 or VSE, you cannot unload and load in the same UCL program. In OS/390 and VSE, the utilities use only a single schema. You need to code the Load function in another UCL program, so you can describe the file differently in another schema.

STANDARD-EXIT (*exit-name*)

Restriction If you code this statement, it must precede the FILE statements.

Description *Optional.* Indicates you want to invoke the exit program you name while unloading each record. For guidelines on writing exit programs, see [“Writing exit programs”](#) on page 181.

Format 1–8 alphanumeric characters. The first character must be alphabetic.

Considerations

- ◆ You must make your exit program available to be loaded by the Unload function. You must put the exit program in your execution library.
- ◆ Only one exit at a time resides in memory. If you code a new exit name in a subsequent function, the utility deletes the current exit program before loading the new one.
- ◆ If you code an invalid exit program, an error results.

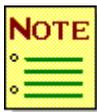
FILE($\left\{ \begin{array}{l} \text{ALL} \\ \text{file - name} \end{array} \right\}$)...

Description *Required.* Names the files you want unloaded.

Format 4 alphanumeric characters. The first character must be alphabetic.

Considerations

- ◆ If you code FILE (ALL), the function unloads only your database files, not the Directory files. If you have index files, you must depopulate them before you unload all of your files.
- ◆ If you code FILE (ALL), the function unloads the primary and then the related files in alphabetical order.
- ◆ To unload Directory files, you must code FILE (*file-name*), not FILE (ALL). In the control section, code the bootstrap schema and environment description. In the CSIPARM file, do not code a DIRECTORY or REALM parameter.
- ◆ Do not code both FILE (ALL) and FILE (*ffff*).
- ◆ The Unload function always opens files for exclusive update. When the Unload function finishes, it explicitly closes them.
- ◆ You cannot code FILE (*file-name-list*).
- ◆ If you are loading a related file, you must load all associated primary files at the same time. If you want to avoid unloading and loading an associated primary file, you must clear all linkpaths that connect it to the related file before loading the related file. The easiest way to clear the linkpaths is to use the Modify function with QUALIFIER (SERIAL). With the Modify function, you can change linkpaths so they contain eight blanks.



Warning: Be careful that you modify the correct linkpath(s), or you will destroy the linkpath and the connection between the files. To recreate a destroyed linkpath, you must unload and load the primary and related files that shared the linkpath.

- ◆ When loading a primary file, you do not need to load all of the associated related files. Instead, you must clear the primary file linkpaths connected to the related files you are loading. However, you must no clear linkpaths connected to related files that you are not loading. You can use the CLEAR-LINKS statement in the Unload or Load function. For an example of loading a primary file without all of its related files, see “[Examples of Unload, Load, and Modify functions](#)” on page 191.

LINKPATH ($\left[\begin{array}{l} \text{b} \\ \text{access} - \text{linkpath} \end{array} \right]$)

Restriction	Use this statement only for related files.
Description	<i>Optional.</i> Indicates the access linkpath to use to unload a related file. The Unload function ignores this statement when you unload primary files.
Default	∅
Format	8 alphanumeric character linkpath name in the format <i>ffffLKxx</i> , where <i>ffff</i> is the name of the primary file, and <i>LKxx</i> is the linkpath.
Options	∅ Use the first linkpath defined for that file. <i>access-linkpath</i> Uses the specified linkpath.

Considerations

- ◆ You must code a linkpath that is in the base portion of the record.
- ◆ If you code FILE (ALL), LINKPATH (*ffffLKxx*) is normally invalid unless your schema specifies the same linkpath in all related files.
- ◆ You should code the same access linkpath to load a file that you used to unload it.
- ◆ If you use the default linkpath value, be careful when you unload from an old schema and load to a new schema. Since the default linkpath is the first defined linkpath in the schema, make sure that the first defined linkpath in both schemas is the same.
- ◆ You must not code LINKPATH (*access-linkpath-list*).

PRESERVE ($\left[\begin{array}{c} \text{NO} \\ \text{YES} \end{array} \right]$)

Restrictions

- ◆ Use this statement only for related files.
- ◆ Use this statement only following the LINKPATH statement.

Description *Optional.* Indicates whether to retain the existing chain sequence when unloading a related file.

Default NO

Considerations

- ◆ Do not use PRESERVE (YES) to unload a file that has corrupted chains or any form of chain damage.
- ◆ The Unload function retains the chain sequence only on the primary access linkpath.
- ◆ When you unload Directory files, code PRESERVE (YES).
- ◆ When reloading your files, do not code a SEQUENCE statement, or you will lose the preserved chain sequence.

CLEAR-LINKS (*linkpath-list*)

- Restriction** Use this statement only for primary files.
- Description** *Optional.* Identifies the linkpaths you want blanked in a primary file.
- Format** Linkpath names must be 4 alphanumeric characters in the format LKxx. The first two characters should be LK, and the last two should represent a linkpath name.

Considerations

- ◆ **IMPORTANT:** When unloading a primary file that has a related file linked to it, this statement must list all primary file linkpaths to the related file.
- ◆ You do not need to use this statement if you intend to clear the linkpaths in the Load function.



We recommend that you clear the linkpaths in the Load function so that you can decide which files to load at that time. If you clear linkpaths when unloading, you must load all of those files

- ◆ If you do not list linkpaths, they are not blanked, but retain their current pointer values. If you are unloading a related file without its primary file, you should not blank the linkpath.
- ◆ If you code the CLEAR-LINKS statement, it must precede the RECORD statements.
- ◆ All linkpaths that you code must be in the primary file that you are unloading.
- ◆ Any linkpath you code in this statement you must also code in the ELEMENT statement by coding ELEMENT (ALL) or explicitly coding the linkpaths in ELEMENT (*element-list*).

$$\text{RRN - RANGE} \left(\begin{array}{l} \textit{low} - \textit{rrn} \\ - \textit{high} - \textit{rrn} \\ \textit{low} - \textit{rrn} - \textit{high} - \textit{rrn} \end{array} \right)$$

Description *Optional.* Indicates a range of relative record numbers that you want retrieved for non-KSDS files. Records outside the range are not retrieved.

Format *rrn* Must be 1–9 numeric characters.

Options

<i>low-rrn</i>	Unloads records having relative record numbers from <i>low-rrn</i> to the end of the file.
- <i>high-rrn</i>	Unloads records having relative record numbers from the beginning of the file to the <i>high-rrn</i> .
<i>low-rrn - high-rrn</i>	Unloads records having relative record numbers from <i>low-rrn</i> through <i>high-rrn</i> .

Considerations

- ◆ If you code the RRN-RANGE statement, it must precede the RECORD statements.
- ◆ If the RRN-RANGE statement you code does not contain a valid data record, no data records are unloaded.
- ◆ The dash (-) is a positional separator in front of *high-rrn* and between the *low* and *high-rrns*. You must code it in those two options. You do not need to code it in the first option (*low-rrn*).

**CRITERIA (*element1*[*element2*,...,*elementn*].*operator.datavalue1*
[*datavalue2*...*datavalue*n]END.)**

Description *Optional.* Establishes an argument string that selects the records you want processed.

Format for element

8 alphanumeric characters. The first character in each name must be alphabetic. Separate the names with commas.

Format for operator

- .EQ. Equal
- .NE. Not equal
- .GT. Greater than
- .LT. Less than
- .GE. Greater than or equal to
- .LE. Less than or equal to

Format for datavalue

String of 1–4096 bytes. You must put it after the period following the Boolean operator and follow it with END. If you code more than one element, you must not separate the data values with commas, blanks, or other delimiters.

Considerations

- ◆ Put a period before and after each Boolean operator.
- ◆ If you code the CRITERIA statement, you must put it before the RECORD statements.
- ◆ You can use any number of spaces before the element list, after END., and on either side of the commas.
- ◆ If you code an element name, you must also code it in the ELEMENT statement unless you coded ELEMENT (ALL).
- ◆ You cannot code a null element list in the criteria argument.
- ◆ If you code an element in the criteria argument, the element must be in all the records you want unloaded from the file.
- ◆ You must make the data values the same length as the corresponding elements you code.
- ◆ You may cross input lines with data if necessary, stopping in column 72 and continuing on the next line in column 1. (If you put data in columns 73–80, it is lost.)
- ◆ If you do not code END., the rest of the UCL program is considered data.

RECORD { ALL
record – code }

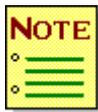
Description *Optional.* Indicates the record you want unloaded.

Default ALL

Format 2 alphanumeric characters

Considerations

- ◆ If you code this statement, you must code the ELEMENT statement. Together, they provide a map of your database record to the Unload and Load functions.
- ◆ When unloading primary files or noncoded, related files, always code RECORD (ALL); otherwise, the function unloads nothing.
- ◆ If you are unloading a coded related file, you must specify RECORD (*record-code*) if you included redefined element names in the element list.



Caution: You can lose coded records if you make errors while using RECORD (*record-code*).

- ◆ If you specify RECORD (*record-code*), be careful to include all appropriate record codes. If you leave out a record code or you forget to load an unloaded record code, you lose any records that belong to that record code.
- ◆ List only record codes that are in the file you are unloading.
- ◆ If you code more than one RECORD statement, do not code RECORD (ALL) in conjunction with RECORD (*record-code*).
- ◆ If you coded FILE (ALL), you must code RECORD (ALL).
- ◆ If you code RECORD (), the function unloads no records.

ELEMENT ({ ALL
element - list })

- Restriction** *Required* if you code the RECORD statement.
- Description** Indicates the data elements you want unloaded.
- Default** ALL
- Format** Element names must be 8 alphanumeric characters. The first character must be alphabetic. Separate names with commas.

Considerations

- ◆ Together, the RECORD and ELEMENT statements provide a map of your data record to the Unload and Load functions. For more detail on the data record, see “[Formatting the data records](#)” on page 188.



Caution: You can lose elements if you make errors when you code ELEMENT (*element-list*).

- ◆ If you forget to code an element that is in the unloaded file or load an element that you unloaded, that element will be blank.
- ◆ If RECORD (ALL) has been coded, then:

$$((\text{number of record codes}) \times (\text{number of elements specified} + 1)) + 3$$
 must be ≤ 256 .

 If you have not coded RECORD (ALL), then:

$$(\text{number of record codes}) + (\text{number of elements specified}) + 3$$
 must be ≤ 256 .
- ◆ If you code ELEMENT (ALL), you cannot change the structure of the file. When you code ELEMENT (ALL), the Unload function picks up each record from the database file exactly as it is on the schema and puts the record in the data file. When you load, the Load function picks up each record from the data file and puts the record into the database file exactly as it is on the schema. Thus if you plan to change the structure of the file, you must use ELEMENT (*element-list*).

- ◆ When you code ELEMENT (*element-list*), the unload function pulls the elements off your database record in the order in which you code them in the list and puts them in the data record. The Load function uses its element list to map the data record and to put the elements into your database record using the order in the Load schema.
- ◆ When you code ELEMENT (*element-list*), you do not need to list the element names in the same order as the schema.
- ◆ If you use an element list, do not code any linkpath elements for a related file.
- ◆ If you use an element list, you must code the key first in the list. The key element in a list for a related file is the data element associated with the specified access linkpath.
- ◆ If you want to change the file's structure in the new schema, you must make the element lists in the Unload and Load function match the data record. When you change the size of elements, use the *FILL=*nn* parameter. (Here *nn* is the number of spaces that are different. If you want to change more than 99 spaces, you can code multiple *FILL parameters in succession.)
- ◆ You can add elements, delete elements, increase their size, and decrease their size with the *FILL=*nn* parameter.

To add an element so it is automatically filled with blanks, do not code the element name in the Unload function's element list. (It is not there already.) Code the element name at the end of the Load function's element list.

You can also add an element so that it appears in the data record where you can modify it with an edit program. Your edit program can be in either the Unload or Load function. To add an element to the data record, code the *FILL=*nn* parameter in the Unload function's element list where you want the element to appear in the data record. In the Load function's element list, code the element name where it will map to the same portion of the data record.

To delete an element, do not unload or load it. That is, leave it out of the element lists in both functions.

To increase the size of an element, code `*FILL=nn` in the Unload function's element list. Replace `nn` with the number of bytes you want to add to the element. You can code `*FILL=nn` before, after, or both before and after the element name. In the Load's element list, code just the element name. The bytes are added in the Unload function and automatically set to blanks. You can modify these blanks with an exit program in either the Unload or Load function.

To reduce the size of an element, first code the element name in the Unload function's element list. Next, in the Load function's element list, code `*FILL=nn` before or after that element. The Load function then ignores the number of bytes you code in the position of the `*FILL=nn` element.



Whenever you change the elements or their size, you must make similar changes in the schemas you use for the Unload and Load functions.

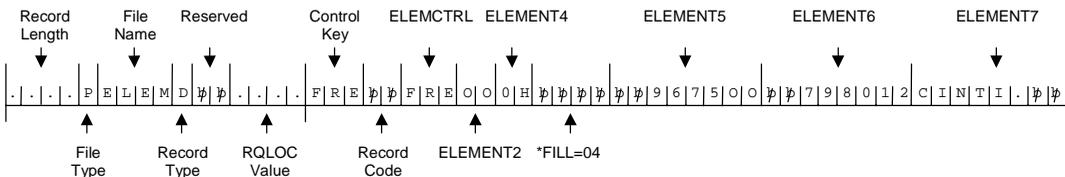
For an example of how the `*FILL` parameter works, assume you are making the following changes:

- Deleting ELEMENT3 with 8 bytes.
- Adding ELEMENT8 with 5 bytes.
- Adding ELEMENT9 with 4 bytes.
- Decreasing the size of ELEMENT6 from 8 to 4 bytes by removing the first 4 bytes.
- Adding 4 bytes to the front of ELEMENT5 to increase its size from 8 to 12 bytes.

To make these changes, code this element list in the Unload function:

```
ELEMENT (ELEMCTRL, ELEMENT2, ELEMENT4, *FILL=04, ELEMENT5, ELEMENT6, ELEMENT7).
```

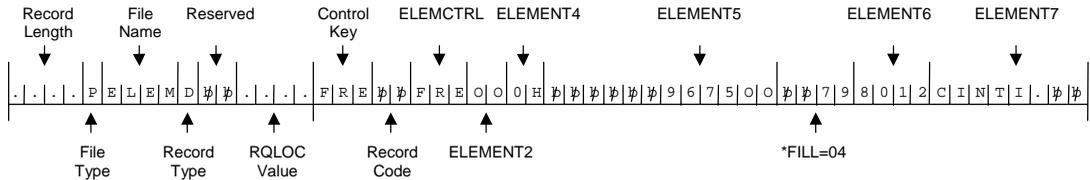
As a result, your data record would look like this:



In the Load function, code this element list:

```
ELEMENT (ELEMCTRL, ELEMENT2, ELEMENT4, ELEMENT5, *FILL=04,
          ELEMENT6, ELEMENT7, ELEMENT8, ELEMENT9)
```

During execution, the data records are passed from the Unload function to the Load function by way of the data file (CSUDATA). Thus, this element list maps the same data record as follows:



As you can see, the number of spaces in the *FILL parameter actually appears in the data record as spaces. You can modify this space with an exit program in both the Unload and Load functions.

The newly added ELEMENT8 and ELEMENT9 do not appear because they are automatically filled with spaces. These spaces do not show up in the data record because you did not code *FILL=09 in the Unload element list to create the necessary spaces.

- ◆ When you code ELEMENT (ALL), the Unload and Load functions see the database record the same way it is on the database file. That is, it is complete with the linkpath fields, record codes, and root fields.
- ◆ If you code an exit program with ELEMENT (ALL), you must be aware of what the data record looks like. See the description in Consideration 10, above.
- ◆ It is possible, but not recommended, to use ELEMENT (ALL) in the Unload function and then use an element list in the Load function, and vice-versa. If you do this, you must make the element list match the data record created when you coded ELEMENT (ALL). For example, you must code the elements in the same order as on the schema. You must also code *FILL=08 wherever a linkpath occurs because you cannot code linkpath names in the element list for a related file.

- ◆ You may use ELEMENT (ALL) in conjunction with an element list. For example, you may code the following:

```
RECORD ( 01 )
ELEMENT ( ELEMENT1 , ELEMENT2 , ELEMENT3 )
RECORD ( 02 )
ELEMENT ( ALL )
```

- ◆ If you code FILE(ALL), do not code ELEMENT (*element-list*) unless your schema has only one file.
- ◆ Additional constraints apply in the following situations:

In this context:	ELEMENT (<i>element-list</i>) must conform to these rules:
FILE(<i>primary-file</i>)	You must code the primary file's control key first in the element list. Do not include the root element in the element list.
FILE(<i>related-file</i>)	The first key defined in the schema for the related file must be the first in the element list. Do not code linkpaths in the element list.
FILE (<i>related-file</i>) LINKPATH(<i>ffffLKxx</i>)	You must put the key associated with linkpath <i>ffffLKxx</i> first in the element list. Do not code the linkpath in the element list.
FILE (<i>coded-file</i>) <i>no access-linkpath</i>	You must put the first key associated with the linkpath defined in the base portion of the record first in the element list.
FILE (<i>noncoded-file</i>)	You must put the key first in the element list.
FILE (<i>coded-file</i>) LINKPATH (<i>ffffLKxx</i>)	You must put the key associated with linkpath <i>ffffLKxx</i> first in the element list. Do not code linkpaths in the element list.
CLEAR-LINKS (<i>LKxx-list</i>)	You must put the same names in the element list that you put in the <i>LKxx-list</i> .
RECORD (<i>record-code</i>)	You must put the same names in the element list that you put in the record code unless you coded ELEMENT (ALL).
FILE (<i>coded-file</i>) RECORD (ALL)	Do not include redefined element names in the element list.

Coding the UCL for the Load function

After you code the control section as shown in “[Coding the control section](#)” on page 57, you can code the Load function as shown in the following format. For UCL examples of unloading and loading, see “[Examples of Unload, Load, and Modify functions](#)” on page 191.

FUNCTION(LOAD)

[STANDARD - EXIT (*exit - name*)]

FILE ({ ALL
file - name })...

[LINKPATH ([b
access - linkpath])]

[CLEAR - LINKS (*linkpath - list*)]

SEQUENCE(*sort - list*)

[DIRECTION([ASCEND]
[DESCEND] - *list*)]

[DATA - TYPE([HEX
CHAR
ZONED - DEC
PACKED - DEC] - *list*)]

[RECORD({ ALL
record - code })]

[ELEMENT({ ALL
element - list })] ...

FUNCTION (LOAD)

Description *Required.* Invokes the Load function.

Considerations

- ◆ If you load and unload in the same UCL program, you must code the Load function after the Unload function.
- ◆ You must match the position and length of data in element lists you use for the Load and Unload functions. If the element lengths you coded in the schema used to load the file do not match those in the Unload function, use `*FILL=nn` to make their lengths equal. For more information and an example of how to use the `*FILL` parameter, see the considerations under the `ELEMENT` statement.
- ◆ If you want to change the description of the file, you cannot execute the Unload and Load functions in the same UCL program. The utilities use only one schema in a UCL program.

STANDARD-EXIT (*exit-name*)

Restriction If you code this statement, you must put it before the `FILE` statements.

Description *Optional.* Indicates you want the exit program you name invoked while loading each record. For guidelines on writing exit programs, see [“Writing exit programs”](#) on page 181.

Format 1–8 alphanumeric characters. The first character must be alphabetic.

Considerations

- ◆ You must put your exit program in your execution library, so it is available to be loaded.
- ◆ Only one exit at a time resides in memory. If you code a new `exitname` in a subsequent function, the function deletes the current exit program before loading the new one.

FILE($\left\{ \begin{array}{l} \text{ALL} \\ \text{file - name} \end{array} \right\}$)...

Description *Required.* Names the database files you want loaded.

Format 4 alphanumeric characters. The first character must be alphabetic.

Considerations

- ◆ If you code FILE (ALL), you can load all primary and related files, but not Directory files. If you have index files, you must depopulate them before you unload and repopulate them after you load.
- ◆ If you code FILE (ALL), the files are loaded in alphabetical order with primary files first, then the related files.
- ◆ To load Directory files, you must code FILE (*file-name*). In the control section, code the bootstrap schema and environment description. In the CSIPARM file, do not code a DIRECTORY or REALM parameter.
- ◆ Do not code FILE (ALL) in conjunction with FILE (*file-name*).
- ◆ The Load function always opens files for exclusive update. When the Load function finishes, it explicitly closes them.
- ◆ You do not need to load all the unloaded files.
- ◆ You cannot code FILE (*file-name-list*).
- ◆ If you are loading a related file, you must load all associated primary files at the same time. If you want to avoid unloading and loading an associated primary file, you must clear all linkpaths in the primary files that connect to the related file before loading. The easiest way is to use the Modify function with QUALIFIER (SERIAL). With the Modify function, you can change the linkpaths so they contain eight blanks. Be careful that you modify the correct linkpath(s), or you will destroy the linkpath and the connection between the files. To recreate a destroyed linkpath, unload and load the primary and related file that shared the linkpath.
- ◆ When loading a primary file, you do not need to load all of the associated related files. Instead, you must clear the primary file linkpaths connected to the related files you are loading. however, you must not clear linkpaths connected to related files that you are not loading. You can use the CLEAR-LINKS statement in the Unload or Load function. For an example of how to load a primary file without all its related files, see “[Examples of Unload, Load, and Modify functions](#)” on page 191.

LINKPATH(^{**b**}
^{**access - linkpath**})

Restriction	Use this statement only for related files.	
Description	<i>Optional.</i> Indicates the access linkpath to use when loading a related file. The Load function ignores this statement when you load primary files.	
Default	b	
Format	8 alphanumeric character linkpath name in the format <i>ffffLKxx</i> , where <i>ffff</i> is the primary file, and <i>LKxx</i> is the linkpath.	
Options	b	Uses the first linkpath defined in the schema for that file as the access linkpath.
	<i>ffffLKxx</i>	Uses the specified linkpath.

Considerations

- ◆ The access linkpath must be in the base portion of the record.
- ◆ If you code FILE (ALL), LINKPATH (*ffffLKxx*) is normally invalid unless your schema specifies the same linkpath in all related files.
- ◆ When you load a file, you should code the same access linkpath as when you unloaded it.
- ◆ Be careful when you unload from an old schema and load to a new schema if you use the default linkpath value. Since the default linkpath is the first defined linkpath in the schema, make sure that the first defined linkpath in both schemas is the same.
- ◆ You must not code LINKPATH (*access-linkpath-list*).

CLEAR-LINKS (*linkpath-list*)

- Restriction** Use this statement only for primary files. This statement is ignored for related files.
- Description** *Optional.* Indicates which linkpaths you want blanked in a primary file.
- Format** Linkpath names must be 4 alphanumeric characters in the format LKxx, where xx is a linkpath name.

Considerations

- ◆ **IMPORTANT:** When loading a primary file and the related file linked to it, this statement must list all primary file linkpaths to that related file.
- ◆ You do not need to list the linkpaths if the Unload function cleared them.



However, we recommend that you clear linkpaths in this function so that you can decide which files to load at this time. (If you clear linkpaths when unloading, you have no choice but to reload all those files.)

- ◆ If you do not list linkpaths, they are not blanked, but retain their current pointer values. If you are unloading a related file without its primary file, you should not blank the linkpath.
- ◆ If you code the CLEAR-LINKS statement, you must code it before the RECORD statements.
- ◆ All linkpaths that you code must be in the primary file that you are loading.
- ◆ Any linkpath you code in this statement you must also code in the ELEMENT statement, either by coding ELEMENT (ALL) or by explicitly coding the linkpaths in ELEMENT (*element-list*).

SEQUENCE (*sort-list*)

- Restriction** If you code a SEQUENCE statement, you must code it before the RECORD statements.
- Description** *Optional.* Names the elements you want added to the standard sort sequence fields.
- Format** 8 alphanumeric characters.

Considerations

- ◆ If you coded PRESERVE (YES) in the Unload function and want to retain your chain sequence, do not code the SEQUENCE statement.
- ◆ If you code a sort-list of elements in this statement, you must also code them in the list of elements to be loaded. You do not need to do this if you code ELEMENT (ALL).
- ◆ Name only elements that are valid for the file you are loading.
- ◆ Sorting slows performance. The more fields you sort, the more performance is degraded. The maximum number of fields you can sort is 62.
- ◆ The elements you select to sort must be in the base portion of the record and at a displacement of 4000 bytes or less in the data file.
- ◆ You may not use more than 252 bytes for all your sort fields plus the length of the access linkpath key or control key.
- ◆ Fields in related files are sorted in the following order:
 - a. RQLOC of the record
 - b. Control key for the access linkpath
 - c. Sort elements you coded in the SEQUENCE statement
- ◆ Records in related files are sorted first by RQLOC (request location or RRN), which leaves the records that hashed to the same location (synonyms) grouped together. Next, the synonyms are sorted by control key, which leaves records with the same control keys that are in the same linkpath chain grouped together. Finally, the records with the same control keys are sorted within linkpath chains.
- ◆ Thus, any SEQUENCE statements you code will sort the records within individual linkpath chains in the access linkpath.
- ◆ With primary files, records are sorted first by RQLOC, and then by control keys. Since control keys are unique, there are no duplications and additional sorting is not necessary. Thus, this statement is not needed and slows the Load function needlessly.

DIRECTION ($\left[\begin{array}{c} \text{ASCEND} \\ \text{DESCEND} \end{array} \right]$ - *list*)

Restriction You can only use this statement following a SEQUENCE statement.

Description *Optional.* Indicates the direction in which you want a corresponding sort-field element to be sorted.

Default ASCEND

Considerations

- ◆ You may code many fields in conjunction with the SEQUENCE statement. For example, you can code the following:

```
SEQUENCE (ELEMENT1 , ELEMENT2 , ELEMENT3 )
DIRECTION (ASCEND , DESCEND , ASCEND)
```
- ◆ The DIRECTION list corresponds one-to-one with the SEQUENCE list. That is, the first element in the sequence list is sorted in the first direction listed. The second element is sorted in the second direction. If the DIRECTION list is exhausted, the default direction is used for subsequent elements in the SEQUENCE list. In the preceding example, if you added ELEMENT4 to the SEQUENCE list, but not to the DIRECTION list, ELEMENT4 would be sorted in ascending order.

DATA - TYPE ($\left[\begin{array}{l} \text{HEX} \\ \text{CHAR} \\ \text{ZONED - DEC} \\ \text{PACKED - DEC} \end{array} \right]$ - list)

Restriction You can only use this statement following a SEQUENCE statement.

Description *Optional.* Indicates the format or type of data elements you listed in the SEQUENCE statement.

Default CHAR

Options

HEX	The data element is a hexadecimal field
CHAR	The data element is a character field
ZONED-DEC	The data element is a zoned decimal field
PACKED-DEC	The data element is a packed decimal field

Considerations

- ◆ You may code many entries in conjunction with the SEQUENCE statement. For example, you may code the following:

```
SEQUENCE (ELEMENT1 , ELEMENT2 , ELEMENT3 )
DIRECTION (ASCEND , DESCEND , ASCEND )
DATA-TYPE (CHAR , HEX , ZONED-DEC )
```

- ◆ Code your entries in the same order as the elements in the SEQUENCE statement. In the preceding example, you must code ELEMENT1's data type first and ELEMENT2's data type second. ELEMENT1 is a character field, and ELEMENT2 is a hexadecimal field.
- ◆ If the DATA-TYPE list is exhausted, the default data type is used for subsequent elements in the SEQUENCE list. For example, if you coded an ELEMENT4 but not another data type, the data type would be assumed to be Character.

RECORD ({ ALL
record - code })..

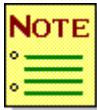
Description *Optional.* Indicates the records you want loaded.

Default ALL

Format 2 alphanumeric characters

Considerations

- ◆ If you code this statement, you must also code the ELEMENT statement. Together, they provide a map of your data record to the Unload and Load functions.
- ◆ When loading primary files or noncoded related files, you must code RECORD (ALL); otherwise, no loading occurs.
- ◆ When you are loading a coded related file, you must code RECORD (*record-code*) if you are going to include redefined element names in the element list or if you want to refer to only specific record codes.



Caution: You can lose coded records if you make errors while using RECORD (*record-code*).

- ◆ If you use RECORD (*record-code*), include all of the appropriate record codes. If you forget to include a record-code that was in the file when you unloaded it or to load an unloaded record-code, you will lose any records that begin with that record-code.
- ◆ List only record codes that are in the file you are loading.
- ◆ Do not code RECORD (ALL) in conjunction with RECORD (*record-code*).
- ◆ If you coded FILE (ALL), you must code RECORD (ALL).
- ◆ If you code RECORD (), no records are loaded.

ELEMENT ({ ALL
element - list })

Restriction *Required* if you coded the RECORD statement.

Description Indicates the elements you want loaded.

Default ALL

Format Element names must be 8 alphanumeric characters. The first character must be alphabetic. Separate names with commas.

Considerations

- ◆ If you code the RECORD statement, you must code the ELEMENT statement. Together, they provide a map of your data record to the Unload and Load functions. For more detail on the data record, see “[Formatting the data records](#)” on page 188.



Caution: You can lose elements if you make errors when you code ELEMENT (*element-list*).

- ◆ If you forget to code an element that is in the unloaded file or load an element that you unloaded, that element will be blank.
- ◆ If RECORD (ALL) has been coded, then:
 $((\text{number of record codes}) \times (\text{number of elements specified} + 1)) + 3$
must be ≤ 256 .
- ◆ If you have not coded RECORD (ALL), then:
 $(\text{number of record codes}) + (\text{number of elements specified}) + 3$
must be ≤ 256 .

- ◆ If you code ELEMENT (ALL), you cannot change the structure of the file. When you code ELEMENT (ALL), the Unload function picks up each record from the database file exactly as it is on the schema and puts the record in the data file. When you load, the Load function picks up each record from the data file and puts the record into the database file exactly as it is on the schema. Thus if you plan to change the structure of the file, you must use ELEMENT (*element-list*).
- ◆ When you code ELEMENT (*element-list*), the Unload function pulls the elements off your database record in the order in which you code them in the list and puts them in the data record. The Load function uses its element list to map the data record and to put the elements into your database record using the order in the Load schema.
- ◆ When you code ELEMENT (*element-list*), you do not need to list the element names in the same order as the schema.
- ◆ If you use an element list, you must code the key first in the list. The key element in a list for a related file is the data element associated with the specified access linkpath.
- ◆ If you use an element list, do not code any linkpath elements for a related file.
- ◆ If you want to change the file's structure in the new schema, you must make the element lists in the Unload and Load function match the data record. When you change the size of elements, use the *FILL=*nn* parameter. (Here *nn* is the number of spaces that are different.) If you want to change more than 99 spaces, you can code multiple *FILL parameters in succession.
- ◆ You can add elements, delete elements, increase their size, and decrease their size with the *FILL=*nn* parameter. Detailed directions and examples are in Consideration 10 of the Unload function's ELEMENT statement in "Coding the UCL for the Unload function" on page 155.
- ◆ If you code an exit program with ELEMENT (ALL), you must be aware of what the data record looks like. See the description in the preceding consideration .
- ◆ When you code ELEMENT (ALL), the Unload and Load functions see the database record the same way it is on the database file. That is, it is complete with the linkpath fields, record codes, and root fields.

- ◆ It is possible, but not recommended, to use ELEMENT (ALL) in the Unload function and then use an element list in the Load function, and vice-versa. If you do this, you must make the element list match the data record created when you coded ELEMENT (ALL). For example, you must code the elements in the same order as on the schema. You must also code *FILL-08 wherever a linkpath occurs because you cannot code linkpath names in the element list for a related file.

- ◆ You may use ELEMENT (ALL) in conjunction with an element list. For example, you may code the following:

```
RECORD ( 01 )
ELEMENT ( ELEMENT1 , ELEMENT2 , ELEMENT3 )
RECORD ( 02 )
ELEMENT ( ALL )
```

- ◆ If you code FILE (ALL), do not code ELEMENT (*element-list*) unless your schema has only one file.
- ◆ Additional constraints apply in the following situations:

In this context:	ELEMENT (<i>element-list</i>) must conform to these rules:
FILE(<i>primary-file</i>)	First entry in the element-list must be the primary file control-key. Do not include the root element in the element list.
FILE(<i>related-file</i>) no <i>access-linkpath</i>	First entry in the element-list must be the first key defined in schema for related file. Do not include linkpaths in the element list.
FILE (<i>related-file</i>) LINKPATH(<i>ffffLKxx</i>)	First entry in element-list must be the key associated with <i>ffffLKxx</i> . Do not include linkpaths in the element list.
FILE (<i>coded-file</i>) no <i>access-linkpath</i>	First entry in element-list must be the key associated with the first linkpath defined in the base portion of the record. Do not include linkpaths in element-list.
FILE (<i>coded-file</i>) LINKPATH (<i>ffffLKxx</i>)	First entry in element-list must be the key associated with linkpath <i>ffffLKxx</i> . Do not include linkpaths in the element list.
CLEAR-LINKS (<i>LKxx-list</i>)	Names appearing in <i>LKxx-list</i> must appear in element-list, unless ELEMENT (ALL) is specified..
FILE (<i>coded-file</i>) RECORD (ALL)	Do not include redefined element names in the element list.

Writing exit programs

There are exit points in both the Unload and Load functions. The Unload function's exit point is located after the record is extracted from the database but before it is written to the output file. Thus, when the record is passed to your exit program, the program may modify the record, delete it, or add a new one. The Unload function passes to your program the address of the current record and the address of the function name.

The Load function's exit point is after the records are sorted and the primary file linkpaths have been created. The Load function passes the address of the current record and the address of the function name to your exit program. The program can modify or delete, but not add, records.

For information on how the exit programs are loaded, how they operate, the languages you can use to write them, and the register conventions you must follow, see “[Inserting exit programs into functions](#)” on page 49. In register 1, for example, you must code the parameter list address. For a description of the parameter list address, see the following table:

Parameter	Data type	Contents before exit (passed to exit program)	Contents after exit (passed from exit program)
Record	<i>n</i> bytes of data	Data record	Must be unchanged
Function name	8 bytes character	UNLOAD bbb or LOAD bbb	Must be unchanged



If your exit program changes anything it is not authorized to change, the results are unpredictable.

Modifying the data record

You can modify a record with exit programs in both the Unload and Load functions. You may modify any data field in the record except the control key and the data record prefix. These fields are saved before calling your exit program. If you alter any part of them, they are restored. Your exit program may alter any other data field.

Your exit program should use register 15 to pass a return code of 0 back to the Exit Interface to indicate either that no action was taken or that the record was modified. The record is then written to the output data file.

Deleting the current data record

You can delete a record with exit programs in both the Unload and Load functions. To delete a record that is passed to your program, your program should pass a return code of 8 in register 15 back to the Exit Interface. Then, the record is not written to the output data file.

Adding a new data record

You can add a record only with an exit program in the Unload function. To add a data record ahead of the one that was passed to the exit program:

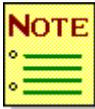
- ◆ Pass a return code of 4 in register 15 to the Exit Interface.
- ◆ Pass the address of the record to be added back in register 1.

Do not move the new record back to the utility data area with your program. The Exit Interface moves the record into the Unload module's data area.

In your program, you must create the new data record in the format described in “[Formatting the data records](#)” on page 188.

After the Unload function writes the new record to the output data file, it passes the previous record back to your program via the Exit Interface.

You must use caution when adding new records with exit programs.



We recommend that you add all records near the beginning because the Unload function does not make further calls to your program when it reaches the end of the database file.

If you try to add records interspersed with ones you are unloading, you may not get the opportunity to add all your records before the function reads the last record from your file. Adding records at the beginning does not add to processing time because the Unload function does not sort them. Therefore, your program does not need to check for collating sequence before adding records.

However, if you are unloading a related file with the PRESERVE (YES) option, you must insert records into the place you want them in the chain.

Retaining the format of the data file

The data file is a sequential work file that passes records from the Unload to the Load function. You define the data file to the Unload and Load functions when you code the DATA-FILE statements in the control section of the UCL or when you use the defaults.

However, you need to be aware of the format in these situations:

- ◆ When you create a data file by writing your own program instead of using the Unload function. You must build the records in the correct format so you can use the data file as input to the Load function.
- ◆ When you write exit programs for the Unload and Load functions. As you delete, modify, or add records in your exit program, you need to keep the formats of the records intact.

The records on the data file are in the following order:

- ◆ Run control record (one record per data file)
- ◆ File pre-header record (one record per database file unloaded)
- ◆ File header record (one record per database file unloaded)
- ◆ Data records (one or more records per database file unloaded)
- ◆ File trailer records (one record per database file unloaded)

You need only one run control record on each data file and it must be the first record on the first volume. You follow it with a set of records for every file that you unloaded or you want to load. The set consists of a file pre-header record, a file header record, a number of data records, and a file trailer record.

You put the primary files first in the data file, and then the related files. You list both types of files in ascending sequence by database file name.

Indicate the record length in the first field in each record. You indicate the record length whether the format is fixed or variable.

Formatting the run control record

You need to put the run control record first on the data file. You need only one run control record per file. If you leave out the run control record, the Unload and Load functions do not execute.

The following is the format for the run control record:

	Record Length	File Type	File Name	Record Type	Run Date	Run Time	Record Format	Record Size	Block Size	Schema Name	Env-Desc Name
bytes	4	1	4	1	8	8	2	4	4	8	8

where:

- ◆ **Record length** is a 4-byte field containing the binary integer 48.
- ◆ **File type** is a 1-byte field that is blank (X'40').
- ◆ **File name** is a 4-byte field that is blank (4 X'40').
- ◆ **Record type** is a 1-byte field containing a "C".
- ◆ **Run date** is an 8-byte field containing the date that the Unload function executed. The format is either MM/DD/YY or DD/MM/YY, depending on your Common Logic Module (CLM) option.
- ◆ **Run time** is an 8-byte field containing the time the Unload function began unloading this file. The format is HH:MM:SS.
- ◆ **Record format** is a 2-byte field containing the record format of the data file. You can have F^b, FB, V^b, or VB.
- ◆ **Record size** is a 4-byte hexadecimal field containing the record size of the data file.
- ◆ **Block size** is a 4-byte hexadecimal field containing the block size of the data file.
- ◆ **Schema name** is an 8-byte field containing the name of the schema used to unload the files on the data file.
- ◆ **Env-Desc name** is an 8-byte field containing the name of the environment description used to unload the files on the data file.

Consideration The minimum length of the run control record is 52 bytes. If the data file is longer than that, you must add blanks to the rest of the record.

Formatting the pre-header record

Pre-header records are optional in files you create yourself and are used when a file is not in its expected position. The Load function reads a pre-header record for the next file, then it can process the next file normally. If you include pre-header records, you need one for each file.

The following is the format you need to use for the pre-header record:

Record Length	File Type	File Name	Unused
4	1	4	26

bytes

where:

- ◆ **Record length** is a 4-byte field containing the binary integer 31.
- ◆ **File type** is a 1-byte field containing an R for related files or a P for primary files.
- ◆ **File name** is a 4-byte field containing the file name for the pre-header record.
- ◆ **Unused** is a 26-byte field that is not used.

Formatting the file header record

You need one file header record for every file on the data file. If you leave out the file header record, the Load function attempts to bypass that file and process the next file. If you do not have file header records for any of the files, the Load function processes no files.

The following is the format you need for the file header record:

	Record Length	File Type	File Name	Record Type	Run Date	Run Time	User Code	Access Linkpath
bytes	4	1	4	1	8	8	1	8

where:

- ◆ **Record length** is a 4-byte field containing the binary integer 31.
- ◆ **File type** is a 1-byte field containing an R for related files or a P for primary files.
- ◆ **File name** is a 4-byte field containing the name of the unloaded or loaded file.
- ◆ **Record type** is a 1-byte field containing an H for file header records.
- ◆ **Run date** is an 8-byte field containing the date the Unload function began unloading this file. The format is MM/DD/YY or DD/MM/YY, depending on the CLM option.
- ◆ **Run time** is an 8-byte field containing the time the Unload function began unloading this file. The format is HH:MM:SS.
- ◆ **User code** is a 1-byte field containing a C if the Unload function created this data file. If you created the data file with your program, you should put a U in this field.
- ◆ **Access linkpath** is an 8-byte field. For a related file, this field contains the name of the linkpath used to control the unload or load. For a primary file, this field contains blanks (8 X '40).

Consideration The minimum length of the file header record is 31 bytes. If the record is longer than 31 bytes, the remainder is filled with blanks.

Formatting the data records

You need one data record for every database record that you selected.

The following is the format for a data record:

	Record Length	File Type	File Name	Record Type	Reserved	RQLOC Value	Control-Key	Record Code	Unloaded Data
bytes	4	1	4	1	2	4	1-256	2	1- <i>n</i>

where:

- ◆ **Record length** is a 4-byte binary integer containing the length of the record. For the value, see Consideration 1 below and subtract 4 from the sum. The record length itself is not included.
- ◆ **File type** is a 1-byte field containing an R for related files or a P for primary files.
- ◆ **File name** is a 4-byte field containing the name of the file that is unloaded or loaded.
- ◆ **Record type** is a 1-byte field containing a D for data records.
- ◆ **Reserved** is a 2-byte field reserved for internal use.
- ◆ **RQLOC value** is a 4-byte field. The contents vary depending on the file type and the options you selected with the Unload or Load function. For the possible values, see Consideration 4 below.
- ◆ **Control-Key** is a 1- to 256-byte field containing the key associated with the data that was unloaded or loaded. For primary files, it is the *ffffCTRL* element value. For related files, it is the key associated with the access linkpath.
- ◆ **Record code** is a 2-byte field containing a record code for a coded record or blanks for a noncoded file.
- ◆ **Unloaded data** is a 1- to *n*-byte field containing all data that was unloaded from the record, including the control key and record code.

Considerations

- ◆ You need to calculate the length of the data record as follows:
$$\text{LRECL}+4 \text{ (for record length)} + 12 \text{ (for prefix length)} + \text{key length} + 2 \text{ (for record code)} + \text{length of all elements to be unloaded}$$
- ◆ When you create data records with an exit program instead of unloading them from a file, you must insert all the elements as defined on the Directory, including the 2-byte reserved field.
- ◆ To understand what is in the Unloaded Data field, see the RECORD and ELEMENT statements in the UCL. Include the record code and the control key even though they are already defined in the data record prefix.
- ◆ The RQLOC value field contains different values at different times. If you are creating your own data file or adding a data record, you must fill the RQLOC field with the value in Considerations A or B below, depending on the type of file and how you coded the PRESERVE statement. The last 3 considerations below describe how this field is used by the loader during processing.
 - After you unload primary or related files with PRESERVE (NO), the RQLOC value field contains (4 X'FF').
 - After you unload a related file with PRESERVE (YES), the RQLOC value field contains a sequential count.
 - While you are loading a primary file, the RQLOC value field contains the RQLOC value calculated from the control key.
 - While you are loading a related file with PRESERVE (NO), the RQLOC value field contains a RQLOC value calculated from the control key defined by access-linkpath.
 - While you are loading a related file with PRESERVE (YES), the RQLOC value field contains a sequential count.

Formatting the file trailer record

You need one file trailer record for every file in the data file. You must put this record after the last data record on each file. If the Load function finds no trailer record, it assumes that the Unload function encountered a nonrecoverable error while creating the data file and stopped before reaching the end-of-file. If the Load function finds no trailer record, it prints out a message and bypasses the file.

The following is the format you must use for each file trailer record:

	Record Length	File Type	File Name	Record Type	Run Date	Run Time	Reserved	Record Count
bytes	4	1	4	1	8	8	2	4

where:

- ◆ **Record length** is a 4-byte field containing the binary integer 28.
- ◆ **File type** is a 1-byte field containing an R for related files or a P for primary files.
- ◆ **File name** is a 4-byte field containing the name of the file just processed.
- ◆ **Record type** is a 1-byte field containing a T for trailer records.
- ◆ **Run date** is an 8-byte field containing the date that the Unload function finished. The format is either MM/DD/YY or DD/MM/YY, depending on the CLM option.
- ◆ **Run time** is an 8-byte field containing the time the Unload function finished.
- ◆ **Reserved** is a 2-byte field reserved for internal use.
- ◆ **Record count** is a 4-byte field containing a binary count of the number of data records unloaded or loaded for this file. Control, header, and trailer records are not included. If the load was unsuccessful, this field contains X'FFFF'.

Considerations

- ◆ The Load function compares the Record Count field to the number of records it loaded. Any discrepancy in the count produces an error message.
- ◆ When you create files, you must include a file trailer record. The Record Count field must be accurate.

Examples of Unload, Load, and Modify functions

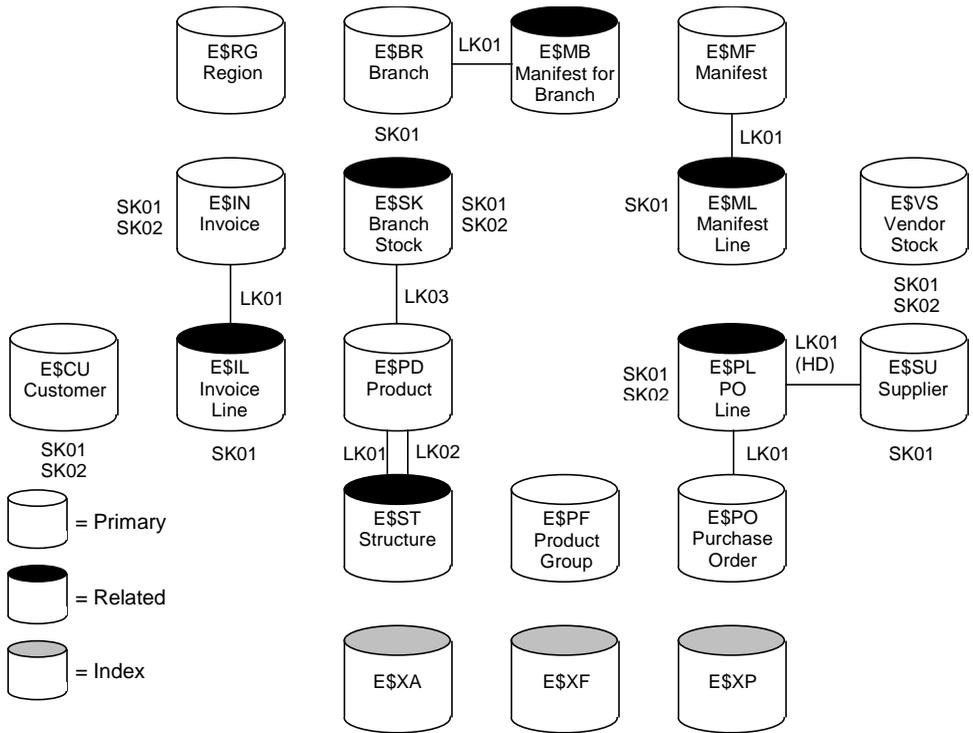
Two examples illustrate how to use the Unload, Load, and Modify functions. The first example shows how to unload and reload all the files in the Burry's database. The second example shows how to unload, change the structure, and reload four of the files: two primary and two related. The change in structure shows how to use the *FILL parameter to add and delete elements in the files. Both examples illustrate when to clear linkpaths. The second example also shows how to use the Modify function to clear a linkpath to a file that was not unloaded.

Since it is necessary to depopulate and repopulate files when you unload and reload them, those steps are shown in both examples. Since the intention here is to reload immediately, the examples show the index files depopulated before unloading. If you are unloading to get a backup copy that you may never reload, you do not need to depopulate. If you ever want to reload the backup copy, you must depopulate first.

To help you understand the examples, the following figure shows the files in the Burry's database. The tables that follow this figure show the files' internal schema. The first table shows the four files that will change as they are unloaded. The modified internal schema in the second table shows the four changed files as they are loaded.



The description of the Burry's database files may not match those on your release of SUPRA. Therefore, do not use them as a basis for decisions you make about Burry's. In addition, these descriptions are not complete; they contain only the information you need to unload and load.



Descriptions of the files in the preceding figure are listed alphabetically in the following table so you can refer to them easily. The files whose structures change are included in the second table where additional information is given.

The following table lists the internal schema of the Burry's database:

Name of file	Type of file	Physical fields	Length of physical fields	Name of secondary keys	
E\$BR	Primary	E\$BRROOT	8	E\$BRSK01	
		E\$BRCTRL	4		
		E\$BRLK01	8		
		E\$BRNAME	20		
		E\$BRADDR	20		
		E\$BRCITY	13		
		E\$BRSTAT	2		
		E\$BRZIPC	5		
		E\$BRREGN	3		
		E\$BRDRTE	2		
		E\$BRSALQ	9		
		E\$BRSTFQ	5		
E\$CU	Primary	E\$CUROOT	8	E\$CUSK01	
		E\$CUCTRL	6		E\$CUSK02
		E\$CUNAME	20		
		E\$CUADDR	20		
		E\$CUCITY	13		
		E\$CUSTAT	2		
		E\$CUZIPC	5		
		E\$CUCLAS	2		
		E\$CUCRAT	2		
		E\$CUCLIM	9		
		E\$CUBRAN	4		
		E\$IL	Related	E\$ILE\$IN	
E\$ILLK01	8				
E\$ILE\$PD	9				
E\$ILQNTY	5				
E\$ILPRCE	9				

Name of file	Type of file	Physical fields	Length of physical fields	Name of secondary keys
E\$IN	Related	E\$INROOT	8	E\$INSK01
		E\$INCTRL	4	E\$INSK02
		E\$INLK01	8	
		E\$INLK04	8	
		E\$INSLMN	4	
		E\$INTOTL	9	
		E\$INBRAN	4	
		E\$INDATE	5	
E\$MB	Related	E\$MBE\$BR	4	none
		E\$MBLK01	8	
		E\$MBE\$MF	5	
		E\$MBFILL	4	
E\$MF	Primary	E\$MFROOT	8	none
		E\$MFCTRL	5	
		E\$MFLK01	8	
		E\$MFTOTL	9	
		E\$MFBRAN	4	
		E\$MFDATE	5	
E\$ML	Related	E\$MLE\$MF	5	E\$MLSK01
		E\$MLLK01	8	
		E\$MLE\$PD	9	
		E\$MLQNTY	5	
		E\$MLVLUE	9	
E\$PG	Primary	E\$PGROOT	8	none
		E\$PGCTRL	2	
		E\$PGDESC	30	

Name of file	Type of file	Physical fields	Length of physical fields	Name of secondary keys
E\$RG	Primary	E\$RGROOT	8	
		E\$RGCTRL	3	
		E\$RGNAME	20	
E\$SK	Related	E\$SKE\$BR	4	E\$SKSK01
		E\$SKE\$PD	9	
		E\$SKLK03	8	
		E\$SKQNTY	5	
		E\$SKBINL	5	
		E\$SKSYTD	9	
E\$SU	Primary	E\$SUROOT	8	E\$SUSK01
		E\$SUCTRL	6	
		E\$SULK01	8	
		E\$SUNAME	20	
		E\$SUADDR	20	
		E\$SUCITY	13	
		E\$SUSTAT	2	
		E\$SUZIPC	5	
E\$VS	Primary	E\$VSROOT	8	E\$VSSK01
		E\$VSCTRL	15	E\$VSSK02
		E\$VSE\$SU		
		E\$VSE\$PD		
		E\$VSNUMB		
		E\$VSVCS		

To see the change in structure, you need additional information about the files: the logical record length, total logical records, type of physical field, and number of decimal places. The type of field can be binary, character, or zoned decimal, which is shown as B, C, and Z in the following table.

The following table shows the internal schema of files before unloading:

Name of file	Type of file	Physical fields	Length of physical fields	Type of physical field	Decimal	Name of secondary keys
E\$PD	Primary	E\$PDROOT	8	B	0	E\$PDLK01
		E\$PDCTRL	9	C	0	
		E\$PDLK01	8	B	0	
		E\$PDLK02	8	B	0	
		E\$PDLK03	8	B	0	
		E\$PDDESC	30	C	0	
		E\$PDWQTY	5	Z	0	
		E\$PDPRCE	9	Z	2	
		E\$PDPGRP	2	C	0	
LOGICAL RECORD LENGTH = 87						
TOTAL LOGICAL RECORDS = 484						
E\$PO	Primary	E\$POROOT	8	B	0	none
		E\$POCTRL	6	C	0	
		E\$POLK01	8	B	0	
		E\$POTOTL	9	Z	2	
		E\$PODATE	5	Z	0	
LOGICAL RECORD LENGTH = 36						
TOTAL LOGICAL RECORDS = 1177						
E\$ST	Related	E\$STASSM	9	C	0	none
		E\$PDLK01	8	B	0	
		E\$STCOMP	9	C	0	
		E\$PDLK02	8	B	0	
		E\$STQNTY	5	Z	0	
LOGICAL RECORD LENGTH = 39						
TOTAL LOGICAL RECORDS = 1078						

Name of file	Type of file	Physical fields	Length of physical fields	Type of physical field	Decimal	Name of secondary keys
E\$PL	Coded Related					
Base Portion		E\$PLCODE	2	C	0	E\$PLSK01
		E\$PLE\$PO	6	C	0	E\$PLSK01
		E\$POLK01	8	B	0	
		E\$PLDATA	31	C	0	
LOGICAL RECORD LENGTH = 47						
HD portion redefines E\$PLDATA		E\$PLE\$SU	6	C	0	
		E\$SULK01	8	B	0	
		E\$PLDATE	5	Z	0	
		E\$PLFILL	12	C	0	
LOGICAL RECORD LENGTH of redefined portion = 31						
LN portion redefines E\$PLDATA		E\$PLE\$SU	9	C	0	
		E\$PLE\$PD	5	Z	0	
		E\$PLCOST	9	Z	2	
		E\$PLFILL	8	C	0	
LOGICAL RECORD LENGTH of redefined portion = 31						
PD portion redefines E\$PLDATA		E\$PLDELN	2	B	0	
		E\$PLDELD	5	Z	0	
		E\$PLDELQ	5	Z	0	
		E\$PLDELP	9	C	0	
		E\$PLFILR	10	C	0	
LOGICAL RECORD LENGTH of redefined portion = 31						
TOTAL LOGICAL RECORDS = 902						

The following table shows the internal schema of the files after modification. Asterisks mark the changes.

Name of file	Type of file	Physical fields	Length of physical fields	Type of physical field	Decimal	Name of secondary keys	
E\$PD	Primary	E\$PDROOT	8	B	0	E\$PDLK01	
		E\$PDCTRL	9	C	0		
		E\$PDLK01	8	B	0		
		E\$PDLK02	8	B	0		
		E\$PDLK03	8	B	0		
		E\$PDDESC	30	C	0		
		E\$PDWQTY	5	Z	0		
		*	E\$PDPRCE	7	Z		0
*	E\$PDPGRP	12	C	0			
*	E#PDDES2	20	C	0			
* LOGICAL RECORD LENGTH = 115 TOTAL LOGICAL RECORDS = 484							
E\$PO	Primary	E\$POROOT	8	B	0		
		E\$POCTRL	6	C	0		
		E\$POLK01	8	B	0		
		E\$POTOTL	9	Z	2		
		E\$PODATE	5	Z	0		
* LOGICAL RECORD LENGTH = 36 TOTAL LOGICAL RECORDS = 1200							
E\$ST	Related	E\$STASSM	9	C	0		
		E\$PDLK01	8	B	0		
		E\$STCOMP	9	C	0		
		E\$PDLK02	8	B	0		
		*	E\$STQNTY	5	Z	0	
		*	E\$STCOMM	20	C	0	
* LOGICAL RECORD LENGTH = 59 TOTAL LOGICAL RECORDS = 1078							

Name of file	Type of file	Physical fields	Length of physical fields	Type of physical field	Decimal	Name of secondary keys
E\$PL	Coded Related					
Base Portion		E\$PLCODE	2	C	0	E\$PLSK01
		E\$PLE\$PO	6	C	0	E\$PLSK01
		E\$POLK01	8	B	0	
		E\$PLDATA	31	C	0	
LOGICAL RECORD LENGTH = 47						
HD portion redefines E\$PLDATA		E\$PLE\$SU	6	C	0	
		E\$SULK01	8	B	0	
		E\$PLDATE	5	Z	0	
		E\$PLFILL	12	C	0	
LOGICAL RECORD LENGTH of redefined portion = 31						
LN portion redefines E\$PLDATA		E\$PLE\$SU	9	C	0	
		E\$PLE\$PD	5	Z	0	
		E\$PLCOST	9	Z	2	
		E\$PLFILL	8	C	0	
LOGICAL RECORD LENGTH of redefined portion = 31						
PD portion redefines E\$PLDATA		E\$PLDELN	2	B	0	
		E\$PLDELD	5	Z	0	
*		E\$PLDELP	9	C	0	
*		E\$PLFILR	15	C	0	
LOGICAL RECORD LENGTH of redefined portion = 31						
* TOTAL LOGICAL RECORDS = 950						

Example 1—Unloading and loading all of the Burry's database files

You unload and reload all of your files to improve performance after many updates have changed the structure of the files. You can determine when you need to unload in two ways:

- ◆ When your applications begin finding broken linkpath chains.
- ◆ When your files are no longer structured for best performance. For example, your primary files have many out-of-block synonyms. You can determine if you have structural degradation by executing the File Statistics function regularly.

Unloading and reloading files have the following benefits:

- ◆ Repairing broken linkpath chains.
- ◆ Reorganizing the synonym chains in primary files to minimize the number of out-of-block synonyms.
- ◆ Reorganizing the linkpath chains in related files to optimize access along the primary (or access) linkpath.
- ◆ Reorganizing the secondary key tree structure in the index files.

The last benefit is actually a result of depopulating and repopulating—steps you must take before and after you unload and load. You execute four functions when you unload and load: the Depopulate, Unload, Load, and Sorted-Populate functions.

UCL samples

Before you unload, you must depopulate all your secondary keys. The following UCL is for the **Depopulate** function:

```
CONTROL(BEGIN)
*
  ENV-DESC (BURRYENN)
  SCHEMA   (BURRYSCH)
LIST (ALL)
  DATA-FORMAT (HEX CHAR)
  DIAGNOSTICS (EXTENDED)
*
FUNCTION(DEPOPULATE)
  STATISTICS (ALL)
  FILE (E$BR)
  FILE (E$CU)
  FILE (E$IL)
  FILE (E$IN)
  FILE (E$ML)
  FILE (E$PD)
  FILE (E$PL)
  FILE (E$SK)
  FILE (E$SU)
  FILE (E$VS)
*
CONTROL(END)
```

After you depopulate secondary keys, you can unload your files. The following UCL unloads all the Burry's database files:

```
CONTROL (BEGIN)
*
  ENV-DESC (BURRYENN)
  SCHEMA   (BURRYSCH)
LIST (ALL)
  DATA-FORMAT (HEX CHAR)
  DATA-FILE (CSUDATA)
  DEVICE (DISK)
*
FUNCTION (UNLOAD)
*
**** PRIMARY FILES ****
FILE (E$BR)
  RECORD (ALL)
  ELEMENT (ALL)
FILE (E$CU)
  RECORD (ALL)
  ELEMENT (ALL)
FILE (E$IN)
  RECORD (ALL)
  ELEMENT (ALL)
FILE (E$RG)
  RECORD (ALL)
  ELEMENT (ALL)
FILE (E$MF)
  RECORD (ALL)
  ELEMENT (ALL)
FILE (E$PD)
  RECORD (ALL)
  ELEMENT (ALL)
FILE (E$VS)
  RECORD (ALL)
  ELEMENT (ALL)
FILE (E$SU)
  RECORD (ALL)
  ELEMENT (ALL)
```

```
FILE (E$PO)
    RECORD(ALL)
        ELEMENT(ALL)
FILE (E$PG)
    RECORD(ALL)
        ELEMENT(ALL)
*
**** RELATED FILES ****
FILE (E$IL)
    LINKPATH()
    RECORD(ALL)
        ELEMENT(ALL)
FILE (E$SK)
    LINKPATH()
    RECORD(ALL)
        ELEMENT(ALL)
FILE (E$MB)
    LINKPATH()
    RECORD(ALL)
        ELEMENT(ALL)
FILE (E$ML)
    LINKPATH()
    RECORD(ALL)
        ELEMENT(ALL)
FILE (E$ST)
    LINKPATH()
    RECORD(ALL)
        ELEMENT(ALL)
FILE (E$PL)
    LINKPATH()
    RECORD(ALL)
        ELEMENT(ALL)
*
CONTROL(END)
```

After you unload your files, you can reload them. The following UCL reloads all Burry's database files and clears all linkpaths:

```
CONTROL(BEGIN)
*
  ENV-DESC (BURRYENN)
  SCHEMA   (BURRYSCH)
LIST (ALL)
  DATA-FORMAT (HEX CHAR)
  DATA-FILE (CSUDATA)
  DEVICE (DISK)
*
FUNCTION(LOAD)
*
**** PRIMARY FILES ****
FILE (E$BR)
  CLEAR-LINKS(LK01)
  RECORD(ALL)
  ELEMENT(ALL)
FILE (E$CU)
  CLEAR-LINKS( )
  RECORD(ALL)
  ELEMENT(ALL)
FILE (E$IN)
  CLEAR-LINKS(LK01)
  RECORD(ALL)
  ELEMENT(ALL)
FILE (E$RG)
  CLEAR-LINKS( )
  RECORD(ALL)
  ELEMENT(ALL)
FILE (E$MF)
  CLEAR-LINKS(LK01)
  RECORD(ALL)
  ELEMENT(ALL)
FILE (E$PD)
  CLEAR-LINKS(LK01,LK02,LK03)
  RECORD(ALL)
  ELEMENT(ALL)
FILE (E$VS)
  CLEAR-LINKS( )
  RECORD(ALL)
  ELEMENT(ALL)
```

```
FILE (E$SU)
    CLEAR-LINKS(LK01)
    RECORD(ALL)
    ELEMENT(ALL)
FILE (E$FO)
    CLEAR-LINKS(LK01)
    RECORD(ALL)
    ELEMENT(ALL)
FILE (E$PG)
    CLEAR-LINKS( )
    RECORD(ALL)
    ELEMENT(ALL)
*
**** RELATED FILES ****
FILE (E$IL)
    LINKPATH( )
    RECORD(ALL)
    ELEMENT(ALL)
FILE (E$SK)
    LINKPATH( )
    RECORD(ALL)
    ELEMENT(ALL)
FILE (E$MB)
    LINKPATH( )
    RECORD(ALL)
    ELEMENT(ALL)
FILE (E$ML)
    LINKPATH( )
    RECORD(ALL)
    ELEMENT(ALL)
FILE (E$ST)
    LINKPATH( )
    RECORD(ALL)
    ELEMENT(ALL)
FILE (E$PL)
    LINKPATH( )
    RECORD(ALL)
    ELEMENT(ALL)
*
CONTROL( END)
```

After you reload your files, you can repopulate your index files with secondary keys. The following UCL is for the **Sorted-Populate** function:

```
CONTROL(BEGIN)
*
  ENV-DESC (BURRYENN)
  SCHEMA   (BURRYSCH)
LIST (ALL)
  DATA-FORMAT (HEX CHAR)
  DIAGNOSTICS (EXTENDED)
*
FUNCTION(SORTED-POPULATE)
  STATISTICS (ALL)
  FILE (E$BR)
  FILE (E$CU)
  FILE (E$IL)
  FILE (E$IN)
  FILE (E$ML)
  FILE (E$PD)
  FILE (E$PL)
  FILE (E$SK)
  FILE (E$SU)
  FILE (E$VS)
*
CONTROL(END)
```

Sample listing

The following listing shows the output you receive as a result of the sample UCL:

```

CSUL0101I  :  COMMENCING COMMAND VALIDATION.
1          CONTROL(BEGIN)
2          *
3          ENV-DESC (BURRYENN)
4          SCHEMA   (BURRYSCH)
5          LIST (ALL)
6          DATA-FORMAT (HEX CHAR)
7          DATA-FILE (CSUDATA)
8          DEVICE (DISK)
9          *
10         FUNCTION(UNLOAD)
11         *
12         **** PRIMARY FILES ****
13         FILE (E$BR)
14         RECORD(ALL)
15         ELEMENT(ALL)
16         FILE (E$CU)
17         RECORD(ALL)
18         ELEMENT(ALL)
19         FILE (E$IN)
20         RECORD(ALL)
21         ELEMENT(ALL)
22         FILE (E$RG)
23         RECORD(ALL)
24         ELEMENT(ALL)
25         FILE (E$MF)
26         RECORD(ALL)
27         ELEMENT(ALL)
28         FILE (E$PD)
29         RECORD(ALL)
30         ELEMENT(ALL)
31         FILE (E$VS)
32         RECORD(ALL)
33         ELEMENT(ALL)
34         FILE (E$SU)
35         RECORD(ALL)
36         ELEMENT(ALL)

```

```

37             FILE (E$PO)
38             RECORD(ALL)
39             ELEMENT(ALL)
40             FILE (E$PG)
41             RECORD(ALL)
42             ELEMENT(ALL)
43             *
44             **** RELATED FILES ****
45             FILE (E$IL)
46             LINKPATH( )
47             RECORD(ALL)
48             ELEMENT(ALL)
49             FILE (E$SK)
50             LINKPATH( )
51             RECORD(ALL)
52             ELEMENT(ALL)
53             FILE (E$MB)
54             LINKPATH( )
55             RECORD(ALL)
56             ELEMENT(ALL)
57             FILE (E$ML)
58             LINKPATH( )
59             RECORD(ALL)
60             ELEMENT(ALL)
61             FILE (E$ST)
62             LINKPATH( )
63             RECORD(ALL)
64             ELEMENT(ALL)
65             FILE (E$PL)
66             LINKPATH( )
67             RECORD(ALL)
68             ELEMENT(ALL)
69             *
70             CONTROL(END)
CONTENTS OF SOURCE LINES OUTSIDE 1..72 MARGINS IGNORED.
 0             SYNTAX ERRORS DETECTED.
70            COMMAND LINES READ.
 1             CONTROL SECTIONS ANALYZED.
 1             FUNCTION COMMANDS ANALYZED.

```

```
CSUL0102I : COMMENCING COMMAND EXECUTION.
CSUL0301I : COMMENCING CONTROL SECTION USING ENVIRONMENT DESCRIPTION BURRYENN AND SCHEMA BURRYSCH.
CSUL0302I : COMMENCING UNLOAD PROCESS.
CSUL0311I : COMMENCING UNLOAD AGAINST FILE E$BR.
CSUL1703I : UNLOADING PRIMARY FILE E$BR TO DATA FILE CREATED ON 10/31/88 AT 14:39:47.
CSUL1704I :          39 DATA RECORDS WRITTEN TO DATA FILE DURING UNLOAD PROCESSING.
CSUL0349I : END-OF-FILE HAS BEEN ENCOUNTERED ON THE CURRENT FILE.
CSUL0321I : UNLOAD          PROCESSING AGAINST FILE E$BR TERMINATING NORMALLY
CSUL0311I : COMMENCING UNLOAD AGAINST FILE E$CU.
CSUL1703I : UNLOADING PRIMARY FILE E$CU TO DATA FILE CREATED ON 10/31/88 AT 14:39:47.
CSUL1704I :          43 DATA RECORDS WRITTEN TO DATA FILE DURING UNLOAD PROCESSING.
CSUL0349I : END-OF-FILE HAS BEEN ENCOUNTERED ON THE CURRENT FILE.
CSUL0321I : UNLOAD          PROCESSING AGAINST FILE E$CU TERMINATING NORMALLY
CSUL0311I : COMMENCING UNLOAD AGAINST FILE E$IN.
CSUL1703I : UNLOADING PRIMARY FILE E$IN TO DATA FILE CREATED ON 10/31/88 AT 14:39:47.
CSUL1704I :          96 DATA RECORDS WRITTEN TO DATA FILE DURING UNLOAD PROCESSING.
CSUL0349I : END-OF-FILE HAS BEEN ENCOUNTERED ON THE CURRENT FILE.
CSUL0321I : UNLOAD          PROCESSING AGAINST FILE E$IN TERMINATING NORMALLY
.
.
.
          FUNCTION = UNLOAD          FILE = E$ST
CSUL0311I : COMMENCING UNLOAD AGAINST FILE E$ST.
CSUL1703I : UNLOADING RELATED FILE E$ST TO DATA FILE CREATED ON 10/31/88 AT 14:40:24 USING
LINKPATH E$PDLK01.
CSUL1704I :          67 DATA RECORDS WRITTEN TO DATA FILE DURING UNLOAD PROCESSING.
CSUL0349I : END-OF-FILE HAS BEEN ENCOUNTERED ON THE CURRENT FILE.
CSUL0321I : UNLOAD          PROCESSING AGAINST FILE E$ST TERMINATING NORMALLY
CSUL0303I : UNLOAD          PROCESS TERMINATING
CSUL0305I : CONTROL SECTION TERMINATING
CSUL0306I : SUMMARY DATA FOR TERMINATING CONTROL SECTION:
CSUL0361I : NUMBER OF READS ISSUED TO THE PDM = 3699
CSUL0362I : NUMBER OF RECORDS RECEIVED FROM THE PDM = 3683
CSUL0363I : NUMBER OF RECORDS PROCESSED = 3683
CSUL0364I : NUMBER OF RECORDS PRINTED = 0
CSUL0365I : NUMBER OF RECORDS UPDATED = 0
CSUL0366I : NUMBER OF WRITES ISSUED TO PDM = 0
CSUL0307I : ALL CONTROL SECTIONS PROCESSED.
```

```
CSUL0308I : CUMULATIVE SUMMARY DATA FOR ALL CONTROL SECTIONS :
CSUL0361I : NUMBER OF READS ISSUED TO THE PDM = 3699
CSUL0362I : NUMBER OF RECORDS RECEIVED FROM THE PDM = 3683
CSUL0363I : NUMBER OF RECORDS PROCESSED = 3683
CSUL0364I : NUMBER OF RECORDS PRINTED = 0
CSUL0365I : NUMBER OF RECORDS UPDATED = 0
CSUL0366I : NUMBER OF WRITES ISSUED TO PDM = 0
CSUL0103I : DATA BASE UTILITIES SUCCESSFUL TERMINATION.
CSUL0101I : COMMENCING COMMAND VALIDATION.
1          CONTROL(BEGIN)
2          *
3          ENV-DESC (BURRYENN)
4          SCHEMA (BURRYSCH)
5          LIST (ALL)
6          DATA-FORMAT (HEX CHAR)
7          DATA-FILE (CSUDATA)
8          DEVICE (DISK)
9          *
10         FUNCTION(LOAD)
11         *
12         **** PRIMARY FILES ****
13         FILE (E$BR)
14             CLEAR-LINKS(LK01)
15             RECORD(ALL)
16             ELEMENT(ALL)
17         FILE (E$CU)
18             CLEAR-LINKS( )
19             RECORD(ALL)
20             ELEMENT(ALL)
21         FILE (E$IN)
22             CLEAR-LINKS(LK01)
23             RECORD(ALL)
24             ELEMENT(ALL)
25         FILE (E$RG)
26             CLEAR-LINKS( )
27             RECORD(ALL)
28             ELEMENT(ALL)
29         FILE (E$MF)
30             CLEAR-LINKS(LK01)
31             RECORD(ALL)
32             ELEMENT(ALL)
```

```

33          FILE (E$PD)
34          CLEAR-LINKS(LK01,LK02,LK03)
35          RECORD(ALL)
36          ELEMENT(ALL)
37          FILE (E$VS)
38          CLEAR-LINKS( )
39          RECORD(ALL)
40          ELEMENT(ALL)
41          FILE (E$SU)
42          CLEAR-LINKS(LK01)
43          RECORD(ALL)
44          ELEMENT(ALL)
45          FILE (E$PO)
46          CLEAR-LINKS(LK01)
47          RECORD(ALL)
48          ELEMENT(ALL)
49          FILE (E$PG)
50          CLEAR-LINKS( )
51          RECORD(ALL)
52          ELEMENT(ALL)
53          *
54          **** RELATED FILES ****
55          FILE (E$IL)
56          LINKPATH( )
57          RECORD(ALL)
58          ELEMENT(ALL)
59          FILE (E$SK)
60          LINKPATH( )
61          RECORD(ALL)
62          ELEMENT(ALL)
63          FILE (E$MB)
64          LINKPATH( )
65          RECORD(ALL)
66          ELEMENT(ALL)
67          FILE (E$ML)
68          LINKPATH( )
69          RECORD(ALL)
70          ELEMENT(ALL)
71          FILE (E$ST)
72          LINKPATH( )
73          RECORD(ALL)
74          ELEMENT(ALL)
75          FILE (E$PL)
76          LINKPATH( )
77          RECORD(ALL)
78          ELEMENT(ALL)
78          *
80          CONTROL(END)
CONTENTS OF SOURCE LINES OUTSIDE 1...72 MARGINS IGNORED.
0          SYNTAX ERRORS DETECTED.
80         COMMAND LINES READ.
1          CONTROL SECTIONS ANALYZED.
1          FUNCTION COMMANDS ANALYZED.

```

CSUL0102I : COMMENCING COMMAND EXECUTION.
CSUL0301I : COMMENCING CONTROL SECTION USING ENVIRONMENT DESCRIPTION BURRYENN AND SCHEMA BURRYSCH.
CSUL0302I : COMMENCING LOAD PROCESS.
CSUL0311I : COMMENCING LOAD AGAINST FILE E\$BR.
CSUL1300I : LOADING FILES WHICH WERE UNLOADED ON 10/31/88 AT 14:39:46.
USING UNLOAD SCHEMA BURRYSCH AND UNLOAD ENVIRONMENT DESCRIPTION BURRYENN .
FILES ARE BEING LOADED USING SCHEMA BURRYSCH AND ENVIRONMENT DESCRIPTION BURRYENN .
CSUL1302I : 39 DATA RECORDS READ FROM DATA FILE DURING LOAD PROCESSING.
CSUL2800I : FILE E\$BR IS NOW FORMATTED.
CSUL0321I : LOAD PROCESSING AGAINST FILE E\$BR TERMINATING NORMALLY
FUNCTION = LOAD FILE = E\$CU
CSUL0311I : COMMENCING LOAD AGAINST FILE E\$CU.
CSUL1302I : 43 DATA RECORDS READ FROM DATA FILE DURING LOAD PROCESSING.
CSUL2800I : FILE E\$CU IS NOW FORMATTED.
CSUL0321I : LOAD PROCESSING AGAINST FILE E\$CU TERMINATING NORMALLY
.
.
.
FUNCTION = LOAD FILE = E\$SK
CSUL0311I : COMMENCING LOAD AGAINST FILE E\$SK.
CSUL1302I : 2628 DATA RECORDS READ FROM DATA FILE DURING LOAD PROCESSING.
CSUL2800I : FILE E\$SK IS NOW FORMATTED.
CSUL0321I : LOAD PROCESSING AGAINST FILE E\$SK TERMINATING NORMALLY
FUNCTION = LOAD FILE = E\$ST
CSUL0311I : COMMENCING LOAD AGAINST FILE E\$ST.
CSUL1302I : 67 DATA RECORDS READ FROM DATA FILE DURING LOAD PROCESSING.
CSUL2800I : FILE E\$ST IS NOW FORMATTED.
CSUL0321I : LOAD PROCESSING AGAINST FILE E\$ST TERMINATING NORMALLY
CSUL0303I : LOAD PROCESS TERMINATING
CSUL0305I : CONTROL SECTION TERMINATING
CSUL0306I : SUMMARY DATA FOR TERMINATING CONTROL SECTION:
CSUL0361I : NUMBER OF READS ISSUED TO THE PDM = 0
CSUL0362I : NUMBER OF RECORDS RECEIVED FROM THE PDM = 0
CSUL0363I : NUMBER OF RECORDS PROCESSED = 0
CSUL0364I : NUMBER OF RECORDS PRINTED = 0
CSUL0365I : NUMBER OF RECORDS UPDATED = 0
CSUL0366I : NUMBER OF WRITES ISSUED TO PDM = 3683
CSUL0307I : ALL CONTROL SECTIONS PROCESSED.
CSUL0308I : CUMULATIVE SUMMARY DATA FOR ALL CONTROL SECTIONS :
CSUL0361I : NUMBER OF READS ISSUED TO THE PDM = 0
CSUL0362I : NUMBER OF RECORDS RECEIVED FROM THE PDM = 0
CSUL0363I : NUMBER OF RECORDS PROCESSED = 0
CSUL0364I : NUMBER OF RECORDS PRINTED = 0
CSUL0365I : NUMBER OF RECORDS UPDATED = 0
CSUL0366I : NUMBER OF WRITES ISSUED TO PDM = 3683

Example 2—Unloading, changing and loading files

You can unload and reload to make changes to the structure of some files. In addition to changing the structure of the file, you also gain the same benefits as when you unload and reload them; that is, you repair broken linkpath chains, minimize the number of out-of-block synonyms, optimize access along primary linkpaths, and reorganize tree structures.

In this example, you are making three changes to the primary file, E\$PD:

- ◆ Decreasing the size of element E\$PDPRCE from 9 to 7 bytes by removing the two zoned decimals from the front of the element.
- ◆ Adding the 20-character element E\$PDDES2 to the end of the record.
- ◆ Increasing the size of element E\$PDPGRP from 2 to 12 bytes by adding 10 characters to the front.

You are making one change to the related file E\$ST: adding a 20-character element E\$STCOMM to the end.

You are making one change to the primary file E\$PO: increasing its size from 1177 to 1200 total logical records.

You are making two changes to the primary file E\$PL:

- ◆ Increasing its size from 902 to 950 total logical records.
- ◆ In the PD portion, deleting the element E\$PLDELQ, which has five zoned decimals, and increasing the corresponding fill element, E\$PLFILL, from 10 to 15 characters. You are leaving the HD and LN portions the same.

To make these changes, you perform the same steps as in the first example where you unloaded all files: depopulate, unload, load, and populate. However, in this example, you add another step before the Load function: you clear the linkpath to the file E\$SU with the Modify function. Thus, you execute five functions when you unload and load only some of the files. The UCL for each function follows.

Depopulating files

The first step is to depopulate the secondary keys for the files E\$PD and E\$PL. There are no secondary keys for the files E\$ST and E\$PO. The following UCL is for the **Depopulate** function:

```
CONTROL(BEGIN)
*
  ENV-DESC (BURRYENN)
  SCHEMA   (BURRYOLD)
  LIST (ALL)
    DATA-FORMAT (HEX CHAR)
  DIAGNOSTICS (EXTENDED)
*
FUNCTION(DEPOPULATE)
  STATISTICS (ALL)
  FILE (E$PD)
  FILE (E$PL)
*
CONTROL(END)
```

Unloading files

The second step is to unload the four files.

To unload E\$PD, perform these steps:

- ◆ Clear linkpaths LK01 and LK02. Do not clear LK03 because it connects to a file that is not unloaded, E\$SK. To clear linkpaths LK01 and LK02 implicitly, do not include them in the element list. Although not shown in this example, you could include the linkpaths in the element list and code them in the CLEAR-LINKS statement to clear them explicitly.
- ◆ Increase the size of E\$PDPGRP by adding *FILL=10 to the element list.
- ◆ You do not need to code the *FILL parameter to add element E\$PDDDES2, because you will add it to the end of the element list in the Load function's UCL.

To unload E\$PO, code ELEMENT (ALL) because you are not changing any elements. The linkpath is cleared in the Load step.

To unload E\$ST, list it in the FILE statement. You do not need to code a *FILL parameter to add the E\$STCOMM element because you will add it to the end of the element list in the Load function's UCL.

To unload E\$PL, perform these steps:

- ◆ Either list the elements in the HD and LN records, or code ALL in the element list. This example shows both ways.
- ◆ Increase the size of E\$PLFILR by adding *FILL=05.
- ◆ Delete the element E\$PLDELQ by not including it in the element list.

The following UCL shows these steps:

```

CONTROL(BEGIN)
* ENV-DESC (BURRYENN)
  SCHEMA (BURRYOLD)
  LIST (ALL)
    DATA-FORMAT (HEX-CHAR)
  DATA-FILE (CSUDATA)
    DEVICE (DISK)
* FUNCTION(UNLOAD)
*
**** PRIMARY FILES ****
FILE(E$PD)
  RECORD(ALL)
  ELEMENT(E$PDCtrl,E$PDLK03,E$PDDESC,E$PDWQTY,E$PDPDPRCE,*FILL=10,E$PDPGRP)

FILE(E$PO)
  RECORD(ALL)
  ELEMENT(ALL)
*
**** RELATED FILES ****
FILE(E$ST)
  LINKPATH(E$PDLK01)
  RECORD(ALL)
  ELEMENT(E$STASSM,E$STQNTY,E$STCOMP)
FILE(E$PL)
  LINKPATH(E$POLK01)
  RECORD(HD)
  ELEMENT(E$PLE$PO,E$PLCODE,E$PLE$SU,E$PLDATE,E$PLFILL)
  RECORD(LN)
  ELEMENT(ALL)
  RECORD(PD)
  ELEMENT(E$PLE$PO,E$PLCODE,E$PLDELN,E$PLDELD,E$PLDELP,*FILL=05,E$PLFILR)
*
CONTROL(END)

```

Clearing the linkpath to a file that was not unloaded

The third step is to clear the linkpath from the file E\$PL to the file E\$SU. When you unload and load the file E\$PL, the linkpath from E\$PL to E\$PO, E\$POLK01, is cleared so that it can accept the newly created linkpath information that the Load function inserts. However, since you do not unload or load E\$SU, use the **Modify** function to clear the linkpath E\$SULK01. The following UCL shows how to code the Modify function:

```
CONTROL (BEGIN)
*
  ENV-DESC (BURRYENN)
  SCHEMA   (BURRYSCH)
  LIST (NONE)
*
FUNCTION (MODIFY)
*
  FILE (E$SU)
    QUALIFIER (SERIAL)
    RECORD (ALL)
      ELEMENT (E$SULK01)
      DATA (.      END.)
*
CONTROL (END)
```

Loading files

The fourth step is to load the four files.

To load E\$PD, use the same element list you used in the unload step.

- ◆ Add the element E\$PDDES2 to the element list.
- ◆ Add *FILL=02 to decrease the size of the E\$PDPRCE element.

To load the E\$PO file, complete these steps:

- ◆ Code ELEMENT (ALL) because you are not changing any elements.
- ◆ Because you coded ALL for the element list, code LK01 in the BLANK-LINKS statement to clear the linkpath.

When you load E\$ST, code the element E\$STCOMM at the end of the element list.

When you load E\$PL, remove *FILL from the element list.

The following UCL illustrates these steps:

```

CONTROL (BEGIN)
*
ENV-DESC (BURRYENN)
  SCHEMA      (BURRYSCH)
  LIST(ALL)
    DATA-FORMAT (HEX-CHAR)
  DATA-FILE (CSUDATA)
    DEVICE (DISK)
*
FUNCTION (LOAD)
*
**** PRIMARY FILES ****
FILE (E$PD)
  RECORD (ALL)

  ELEMENT (E$PDCTRL, E$PDLK03, E$PDDESC, E$PDWQTY, *FILL=02, E$PDPRCE, E$PDPGRP, E$PDDES2)
FILE (E$PO)
  CLEAR-LINKS (LK01)
  RECORD (ALL)
    ELEMENT (ALL)
*
**** RELATED FILES ****
FILE (E$ST)
  LINKPATH (E$PDLK01)
  RECORD (ALL)
    ELEMENT (E$STASSM, E$STQNTY, E$STCOMP, E$STCOMM)
FILE (E$PL)
  LINKPATH (E$POLK01)
  RECORD (HD)
    ELEMENT (E$SPE$PO, E$SPCODE, E$SPE$SU, E$SPDATE, E$SPFILL)
  RECORD (LN)
    ELEMENT (ALL)
  RECORD (PD)
    ELEMENT (E$PLE$PO, E$PLCODE, E$PLDELN, E$PLDELD, E$PLDELP, E$PLFILR)
*
CONTROL (END)

```

Populating files

The last step is to populate the secondary keys for the files E\$PD and E\$PL. The other two files had no secondary keys. The following UCL illustrates how to code the **Sorted-Populate** function:

```
CONTROL (BEGIN)
*
  ENV-DESC (BURRYENN)
  SCHEMA   (BURRYSCH)
  LIST (ALL)
    DATA-FORMAT (HEX CHAR)
  DIAGNOSTICS (EXTENDED)
*
FUNCTION (SORTED-POPULATE)
  STATISTICS (ALL)
  FILE (E$PD)
  FILE (E$PL)
*
CONTROL (END)
```

Sample listing

The following listing shows the output you receive as a result of the sample statements:

```

CSUL0101I : COMMENCING COMMAND VALIDATION.
 1 CONTROL(BEGIN)
 2 *
 3     ENV-DESC (BURRYENN)
 4     SCHEMA   (BURRYOLD)
 5     LIST(ALL)
 6         DATA-FORMAT (HEX CHAR)
 7     DATA-FILE (CSUDATA)
 8     DEVICE   (DISK)
 9 *
10 FUNCTION(UNLOAD)
11 *
12 **** PRIMARY FILES ****
13     FILE(E$PD)
14         RECORD(ALL)
15             ELEMENT(E$PDCTRL,E$PDLK03,E$PDDESC,E$PDWQTY,
16 E$PDPDPRCE,*FILL=10,E$PDPGRP)
17     FILE(E$PO)
18         RECORD(ALL)
19             ELEMENT(ALL)
20 *
21 **** RELATED FILES ****
22     FILE(E$ST)
23         LINKPATH(E$PDLK01)
24         RECORD(ALL)
25             ELEMENT(E$STASSM,E$STQNTY,E$STCOMP)
26     FILE(E$PL)
27         LINKPATH(E$POLK01)
28         RECORD(HD)
29             ELEMENT(E$PLE$PO,E$PLCODE,E$PLE$SU,E$PLDATE,E$PLFILL)
30         RECORD(LN)
31             ELEMENT(ALL)
32         RECORD(PD)
33             ELEMENT(E$PLE$PO,E$PLCODE,E$PLDELN,E$PLDELD,E$PLDELP,
34 *FILL=05,E$PLFILR)
35 *
36 CONTROL(END)
CONTENTS OF SOURCE LINES OUTSIDE 1..72 MARGINS IGNORED.
 0 SYNTAX ERRORS DETECTED.
36 COMMAND LINES READ.
 1 CONTROL SECTIONS ANALYZED.
 1 FUNCTION COMMANDS ANALYZED.

```

```

CSUL0102I : COMMENCING COMMAND EXECUTION.
CSUL0301I : COMMENCING CONTROL SECTION USING ENVIRONMENT DESCRIPTION BURRYENN AND SCHEMA BURRYOLD.
CSUL0302I : COMMENCING UNLOAD PROCESS.
CSUL0311I : COMMENCING UNLOAD AGAINST FILE E$PD.
CSUL1703I : UNLOADING PRIMARY FILE E$PD TO DATA FILE CREATED ON 10/31/88 AT 15:32:57.
CSUL1704I :          88 DATA RECORDS WRITTEN TO DATA FILE DURING UNLOAD PROCESSING.
CSUL0349I : END-OF-FILE HAS BEEN ENCOUNTERED ON THE CURRENT FILE
CSUL0321I : UNLOAD PROCESSING AGAINST FILE E$PD TERMINATING NORMALLY
            FUNCTION=UNLOAD          FILE=E$PO
CSUL0311I : COMMENCING UNLOAD AGAINST FILE E$PO.
CSUL1703I : UNLOADING PRIMARY FILE E$PO TO DATA FILE CREATED ON 10/31/88 AT 15:32:59.
CSUL1704I :          26 DATA RECORDS WRITTEN TO DATA FILE DURING UNLOAD PROCESSING.
CSUL0349I : END-OF-FILE HAS BEEN ENCOUNTERED ON THE CURRENT FILE
CSUL0321I : UNLOAD PROCESSING AGAINST FILE E$PO TERMINATING NORMALLY
            FUNCTION=UNLOAD          FILE=E$PL
CSUL0311I : COMMENCING UNLOAD AGAINST FILE E$PL.
CSUL1701I : UNLOADING RELATED FILE E$PL TO DATA FILE CREATED ON 10/31/88 AT 15:33:03 USING
            LINKPATH E$POLK01.
CSUL1704I :          122 DATA RECORDS WRITTEN TO DATA FILE DURING UNLOAD PROCESSING.
CSUL0349I : END-OF-FILE HAS BEEN ENCOUNTERED ON THE CURRENT FILE
CSUL0321I : UNLOAD PROCESSING AGAINST FILE E$PL TERMINATING NORMALLY
            FUNCTION=UNLOAD          FILE=E$ST
CSUL0311I : COMMENCING UNLOAD AGAINST FILE E$ST.
CSUL1701I : UNLOADING RELATED FILE E$ST TO DATA FILE CREATED ON 10/31/88 AT 15:33:05 USING
            LINKPATH E$PDLK01.
CSUL1704I :          67 DATA RECORDS WRITTEN TO DATA FILE DURING UNLOAD PROCESSING.
CSUL0349I : END-OF-FILE HAS BEEN ENCOUNTERED ON THE CURRENT FILE
CSUL0321I : UNLOAD PROCESSING AGAINST FILE E$ST TERMINATING NORMALLY
CSUL0303I : UNLOAD PROCESS TERMINATING
CSUL0305I : CONTROL SECTION TERMINATING
CSUL0306I : SUMMARY DATA FOR TERMINATING CONTROL SECTION:
CSUL0361I : NUMBER OF READS ISSUED TO THE PDM =307
CSUL0362I : NUMBER OF RECORDS RECEIVED FROM THE PDM = 303
CSUL0363I : NUMBER OF RECORDS PROCESSED = 303
CSUL0364I : NUMBER OF RECORDS PRINTED = 0
CSUL0365I : NUMBER OF RECORDS UPDATED = 0
CSUL0366I : NUMBER OF WRITES ISSUED TO PDM = 0
CSUL0307I : ALL CONTROL SECTIONS PROCESSED.
CSUL0308I : CUMULATIVE SUMMARY DATA FOR ALL CONTROL SECTIONS :
CSUL0361I : NUMBER OF READS ISSUED TO THE PDM =307
CSUL0362I : NUMBER OF RECORDS RECEIVED FROM THE PDM = 303
CSUL0363I : NUMBER OF RECORDS PROCESSED = 303
CSUL0364I : NUMBER OF RECORDS PRINTED = 0
CSUL0365I : NUMBER OF RECORDS UPDATED = 0
CSUL0366I : NUMBER OF WRITES ISSUED TO PDM = 0
CSUL0103I : DATA BASE UTILITIES SUCCESSFUL TERMINATION.
CSUL0101I : COMMENCING COMMAND VALIDATION.

```

```

1  CONTROL(BEGIN)
2  *
3      ENV-DESC (BURRYENN)
4      SCHEMA   (BURRYSCH)
5      LIST(NONE)
6  *
7  FUNCTION(MODIFY)
8  *
9      FILE(E$SU)
10     QUALIFIER(SERIAL)
11     RECORD(ALL)
12     ELEMENT(E$SULK01)
13     DATA(.      END.)
14  *
15  CONTROL(END)

```

CONTENTS OF SOURCE LINES OUTSIDE 1..72 MARGINS IGNORED.

```

0  SYNTAX ERRORS DETECTED
15 COMMAND LINES READ.
1  CONTROL SECTIONS ANALYZED.
1  FUNCTION COMMANDS ANALYZED.

```

CSUL0102I : COMMENCING COMMAND EXECUTION.

CSUL0301I : COMMENCING CONTROL SECTION USING ENVIRONMENT DESCRIPTION BURRYENN AND SCHEMA BURRYSCH.

CSUL0302I : COMMENCING MODIFY PROCESS.

CSUL0311I : COMMENCING MODIFY AGAINST FILE E\$SU.

CSUL0349I : END-OF-FILE HAS BEEN ENCOUNTERED ON THE CURRENT FILE

CSUL0321I : MODIFY PROCESSING AGAINST FILE E\$SU TERMINATING NORMALLY

CSUL0303I : MODIFY PROCESS TERMINATING

CSUL0305I : CONTROL SECTION TERMINATING

CSUL0306I : SUMMARY DATA FOR TERMINATING CONTROL SECTION:

```

CSUL0361I : NUMBER OF READS ISSUED TO THE PDM = 15
CSUL0362I : NUMBER OF RECORDS RECEIVED FROM THE PDM = 14
CSUL0363I : NUMBER OF RECORDS PROCESSED = 14
CSUL0364I : NUMBER OF RECORDS PRINTED = 0
CSUL0365I : NUMBER OF RECORDS UPDATED = 14
CSUL0366I : NUMBER OF WRITES ISSUED TO PDM = 14

```

CSUL0307I : ALL CONTROL SECTIONS PROCESSED.

CSUL0308I : CUMULATIVE SUMMARY DATA FOR ALL CONTROL SECTIONS :

```

CSUL0361I : NUMBER OF READS ISSUED TO THE PDM = 15
CSUL0362I : NUMBER OF RECORDS RECEIVED FROM THE PDM = 14
CSUL0363I : NUMBER OF RECORDS PROCESSED = 14
CSUL0364I : NUMBER OF RECORDS PRINTED = 0
CSUL0365I : NUMBER OF RECORDS UPDATED = 14
CSUL0366I : NUMBER OF WRITES ISSUED TO PDM = 14

```

CSUL0103I : DATA BASE UTILITIES SUCCESSFUL TERMINATION.

CSUL0101I : COMMENCING COMMAND VALIDATION.

```

1  CONTROL (BEGIN)
2  *
3  ENV-DESC (BURRYENN)
4  SCHEMA   (BURRYSCH)
5  LIST (ALL)
6  DATA-FORMAT (HEX-CHAR)
7  DATA-FILE (CSUDATA)
8  DEVICE (DISK)
9  *
10 FUNCTION (LOAD)
11 *
12 **** PRIMARY FILES ****
13 FILE (E$PD)
14 RECORD (ALL)
15 ELEMENT (E$PDCTRL, E$PDLK03, E$PDDESC, E$PDWQTY,
16 *FILL=02, E$PDPRCE, E$PDPGRP, E$PDDES2)
17 FILE (E$PO)
18 CLEAR-LINKS (LK01)
19 RECORD (ALL)
20 ELEMENT (ALL)
21 *
22 **** RELATED FILES ****
23 FILE (E$ST)
24 LINKPATH (E$PDLK01)
25 RECORD (ALL)
26 ELEMENT (E$STASSM, E$STQNTY, E$STCOMP, E$STCOMM)
27 FILE (E$PL)
28 LINKPATH (E$POLK01)
29 RECORD (HD)
30 ELEMENT (E$SPE$PO, E$SPCODE, E$SPE$SU, E$SPDATE, E$SPFILL)
31 RECORD (LN)
32 ELEMENT (ALL)
33 RECORD (PD)
34 ELEMENT (E$PLE$PO, E$PLCODE, E$PLDELN, E$PLDELD, E$PLDELP, E$PLFILR)
35 *
36 CONTROL (END)

```

CONTENTS OF SOURCE LINES OUTSIDE 1...72 MARGINS IGNORED.

```

0  SYNTAX ERRORS DETECTED.
36 COMMAND LINES READ.
1  CONTROL SECTIONS ANALYZED.
1  FUNCTION COMMANDS ANALYZED.

```

CSUL0102I : COMMENCING COMMAND EXECUTION.
CSUL0301I : COMMENCING CONTROL SECTION USING ENVIRONMENT DESCRIPTION BURRYENN AND SCHEMA BURRYSCH.
CSUL0302I : COMMENCING LOAD PROCESS.
CSUL0311I : COMMENCING LOAD AGAINST FILE E\$PD.
CSUL1300I : LOADING FILES WHICH WERE UNLOADED ON 10/31/88 AT 15:32:55.
USING UNLOAD SCHEMA BURRYOLD AND UNLOAD ENVIRONMENT DESCRIPTION BURRYENN.
FILES ARE BEING LOADED USING SCHEMA BURRYSCH AND ENVIRONMENT DESCRIPTION BURRYENN.
CSUL1302I : 88 DATA RECORDS READ FROM DATA FILE DURING LOAD PROCESSING.
CSUL2800I : FILE E\$PD IS NOW FORMATTED.
CSUL0321I : LOAD PROCESSING AGAINST FILE E\$PD TERMINATING NORMALLY
FUNCTION=LOAD FILE=E\$PO
CSUL0311I : COMMENCING LOAD AGAINST FILE E\$PO.
CSUL1302I : 26 DATA RECORDS READ FROM DATA FILE DURING LOAD PROCESSING.
CSUL2800I : FILE E\$PO IS NOW FORMATTED.
CSUL0321I : LOAD PROCESSING AGAINST FILE E\$PO TERMINATING NORMALLY
FUNCTION=LOAD FILE=E\$PL
CSUL0311I : COMMENCING LOAD AGAINST FILE E\$PL.
CSUL1302I : 122 DATA RECORDS READ FROM DATA FILE DURING LOAD PROCESSING.
CSUL2800I : FILE E\$PL IS NOW FORMATTED.
CSUL0321I : LOAD PROCESSING AGAINST FILE E\$PL TERMINATING NORMALLY
FUNCTION=UNLOAD FILE=E\$ST
CSUL0311I : COMMENCING LOAD AGAINST FILE E\$ST.
CSUL1302I : 67 DATA RECORDS READ FROM DATA FILE DURING LOAD PROCESSING.
CSUL2800I : FILE E\$ST IS NOW FORMATTED.
CSUL0321I : LOAD PROCESSING AGAINST FILE E\$ST TERMINATING NORMALLY
CSUL0303I : LOAD PROCESS TERMINATING
CSUL0305I : CONTROL SECTION TERMINATING
CSUL0306I : SUMMARY DATA FOR TERMINATING CONTROL SECTION:
CSUL0361I : NUMBER OF READS ISSUED TO THE PDM = 0
CSUL0362I : NUMBER OF RECORDS RECEIVED FROM THE PDM = 0
CSUL0363I : NUMBER OF RECORDS PROCESSED = 0
CSUL0364I : NUMBER OF RECORDS PRINTED = 0
CSUL0365I : NUMBER OF RECORDS UPDATED = 0
CSUL0366I : NUMBER OF WRITES ISSUED TO PDM = 303
CSUL0307I : ALL CONTROL SECTIONS PROCESSED.
CSUL0308I : CUMULATIVE SUMMARY DATA FOR ALL CONTROL SECTIONS :
CSUL0361I : NUMBER OF READS ISSUED TO THE PDM = 0
CSUL0362I : NUMBER OF RECORDS RECEIVED FROM THE PDM = 0
CSUL0363I : NUMBER OF RECORDS PROCESSED = 0
CSUL0364I : NUMBER OF RECORDS PRINTED = 0
CSUL0365I : NUMBER OF RECORDS UPDATED = 0
CSUL0366I : NUMBER OF WRITES ISSUED TO PDM = 303
CSUL0103I : DATA BASE UTILITIES SUCCESSFUL TERMINATION.

11

Coding the Version 2 Unload, Load, and Insert Linkpath functions

Version 2 overview

Use the Version 2 Unload, Load, and Insert Linkpath functions if performance is critical or you are reloading the files in a SUPRA converted or Series 80 format. These are the only functions you may use to convert files from the SUPRA native format to the SUPRA converted or Series 80 format. With these functions, you may unload a file in any format (SUPRA native, SUPRA converted, or Series 80) and then reload the file in the same or any other format.



When you use this version of the Unload function, you must use the Version 2 Load Function to reload. When you use the Version 1 Unload function, you must use the Version 1 Load function to reload.

Use the **Version 1 Unload and Load** functions in the following situations:

- ◆ Performance is not critical.
- ◆ You are loading files in the SUPRA native format.
- ◆ You want to code elements in the redefined portion of a coded related file.
- ◆ You want to use the UCL or other features of those functions.



The Version 2 Unload and Load functions do not process index files. If your PDM files have secondary keys, you must depopulate them either before or after unloading them. Then you must repopulate them after you load. If you execute the Insert Linkpath function, you can use it before or after you repopulate.

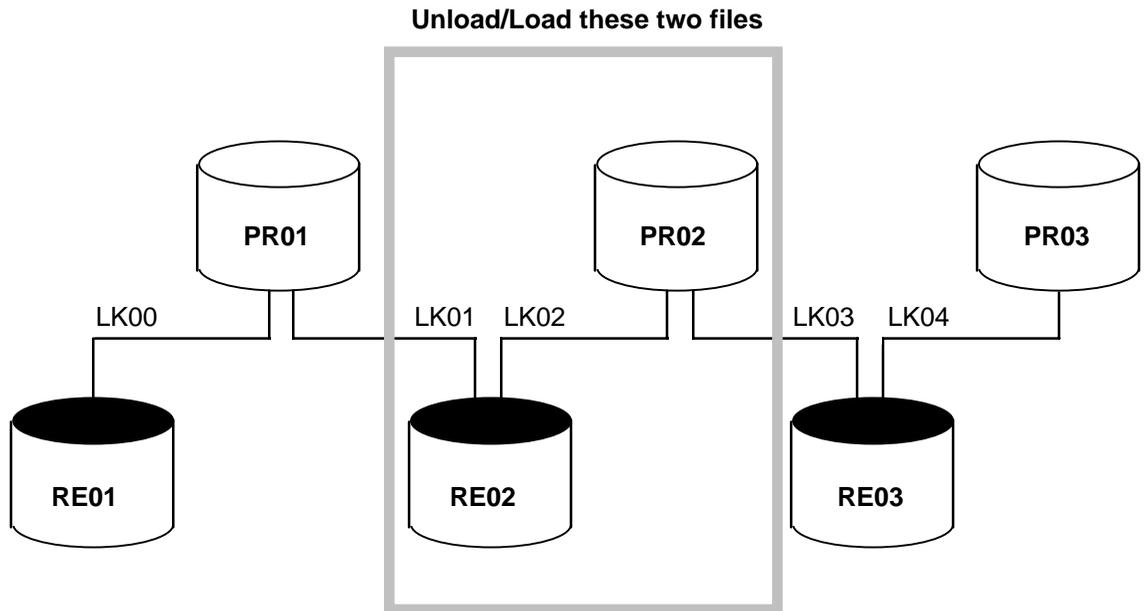
To depopulate secondary keys, use the Directory Maintenance DEPOPULATE command with the Remove parameter or the Depopulate function described in “[Coding the Depopulate function](#)” on page 105. After you have reloaded the files, you can repopulate secondary keys with the Sorted-Populate function described in “[Coding the Sorted-Populate function](#)” on page 91 or the Directory Maintenance POPULATE command. For details on the DEPOPULATE and POPULATE commands, refer to the *SUPRA Server PDM Directory Online User's Guide (OS/390 & VSE)*, P26-1260, or the *SUPRA Server PDM Directory Batch User's Guide (OS/390 & VSE)*, P26-1261.

The Version 2 Unload function (CSUNLOAD) unloads SUPRA Directory or PDM files at serial speed to a tape or disk device. It restructures the files into the format and sequence required by the Version 2 Load function.

The Version 2 Load function (CSULOADR) loads SUPRA Directory or PDM files from the output of the Version 2 Unload function. In addition, this function automatically establishes linkage information.

The Version 2 Insert Linkpath function (CSUINSRT) inserts linkpath data that the Load function saved in the work files. The Insert Linkpath function inserts the data into Directory or PDM primary files. However, because the Load function establishes linkpath information, you do not need to use Insert Linkpath in most cases.

Use the Insert Linkpath function when you are unloading and loading only some of your files. For example, you may unload a related file and only one of the primary files to which it is connected. After you reload the files, insert the connection to the primary file you did not unload. The following figure illustrates this process.



What to do with linkpaths when you unload and load

In the preceding figure, you need to use the Insert Linkpath function for linkpath LK01. You do not need to use it for any other linkpath. These four linkpaths illustrate the four possible states for your linkpaths. The following table explains what action you need to take in each case.

Description	Example linkpath	Action	Effect
You do not unload either the primary or the related file to which the linkpath is connected.	LK04 and LK00	None	The linkpath information remains as it was before you unloaded.
You unload and load both the primary and related file to which the linkpath is connected.	LK02	Code LK02 in the BLANK-LINKS file of the Unload function's control parameter for file PR02.	The linkpath information in the OUTFILE is blanked in the unload step and then recreated and inserted in the load step.
You unload and load the primary file to which the linkpath is connected, but not the related file.	LK03	None. Do not code LK03 in the BLANK-LINKS parameter for file PR02.	The linkpath information is left intact and remains the same as it was before you unloaded.
You unload and load the related file to which the linkpath is connected, but not the primary file.	LK01	Execute the Insert Linkpath function and code PR01 in the FILES parameter and PR01LK01 in the CLEARLKS parameter of the Insert Linkpath run control statement. (Do not code PR01LK00 in the CLEARLKS parameter.)	The linkpath information is created in the load step. The linkpath information is then blanked and inserted in the insert step.

For each of the Version 2 functions, you must code the CSIPARM file, JCL to define files, and control statements. To find the information on each function, see the following sections:

Information	Section
Coding the CSIPARM file for Unload, Load, and Insert Linkpath functions	“Coding the CSIPARM file for Unload, Load, and Insert Linkpath” on page 230
Coding the JCL for Unload, Load, and Insert Linkpath functions	“Coding JCL for Unload, Load, and Insert Linkpath functions” on page 232
Unloading PDM files	“Unloading PDM files” on page 235
Unloading Directory files	“Unloading Directory files” on page 264
Using exit points in the Unload function	“Using exit points” on page 265
Loading PDM files	“Loading PDM files” on page 292
Loading Directory files	“Loading Directory files” on page 322
Inserting linkpath data	“Coding the Insert Linkpath function” on page 322
Examples of Unload, Load, and Insert functions	“Examples of Unload, Load, and Insert Linkpath functions” on page 333

Coding the CSIPARM file for Unload, Load, and Insert Linkpath

When you use Version 2 functions, you need to code the CSIPARM file. In the environment description, you must code an open mode of NONE for all files read by these functions. Also, if you change the schema and environment description in the run control statements, you must change the DIRECTORY and REALM parameters in the CSIPARM file.

Coding CSIPARM file and run control statements for PDM files

The following table shows how to code the CSIPARM file and run control statements for PDM files. If you are processing Directory files, see [“Coding CSIPARM file and run control statements for directory files”](#) on page 231.

In Version 2 utility	Code	For PDM files with no changes in schema	For PDM files with changes in schema
UNLOAD	CSIPARM file	DIRECTORY=(bootschema, bootenvdesc)	DIRECTORY=(bootschema, bootenvdesc)
	CSIPARM file	REALM=(yourschema, yourenvdesc)	REALM=(youoldschema, ouoldenvdesc)
	Run control statements	no NEW-SCHEMA no NEW-ENVDESC	NEW-SCHEMA=yournewschema NEW-ENVDESC=yournewenvdesc
LOAD	CSIPARM file	DIRECTORY=(bootschema, bootenvdesc)	DIRECTORY=(bootschema, bootenvdesc)
	CSIPARM file	REALM=(yourschema, yourenvdesc)	REALM=(yournewschema, yournewenvdesc)
	Run control statements	SCHEMA=yourschema	SCHEMA=yournewschema
INSERT LINKPATH	CSIPARM file	DIRECTORY=(bootschema, bootenvdesc)	DIRECTORY=(bootschema, bootenvdesc)
	CSIPARM file	REALM=(yourschema, yourenvdesc)	REALM=(yournewschema, yournewenvdesc)
	Run control statements	none	none

Coding CSIPARM file and run control statements for directory files

If you are processing Directory files, refer to the following table:

In Version 2 utility	Code	For directory files with no changes in schema	For directory files with changes in schema
UNLOAD	CSIPARM file	no DIRECTORY parameter	no DIRECTORY parameter
	CSIPARM file	REALM= (bootschema, bootenvdesc)	REALM= (oldbootschema, oldbootenvdesc)
	Run control statements	no NEW-SCHEMA no NEW-ENVDESC	NEW-SCHEMA= newbootschema NEW-ENVDESC= newbootenvdesc
LOAD	CSIPARM file	DIRECTORY= (bootschema, bootenvdesc)	DIRECTORY= (newbootschema, newbootenvdesc)
	CSIPARM file	no REALM parameter	no REALM parameter
	Run control statements	SCHEMA= bootschema	SCHEMA= newbootschema
INSERT LINKPATH	CSIPARM file	DIRECTORY= (bootschema, bootenvdesc)	DIRECTORY= (newbootschema, newbootenvdesc)
	CSIPARM file	no REALM parameter	no REALM parameter
	Run control statements	none	none

Coding JCL for Unload, Load, and Insert Linkpath functions

The following figures show the files that you must define in OS/390 and VSE. You use some files to hold your input to the functions and others to hold output from the functions. The CSIPARM and CSUAUX files, which hold input to the Unload, Load, and Insert Linkpath functions, are examples of input files.

Other input files hold run control and file control statements.

OS/390

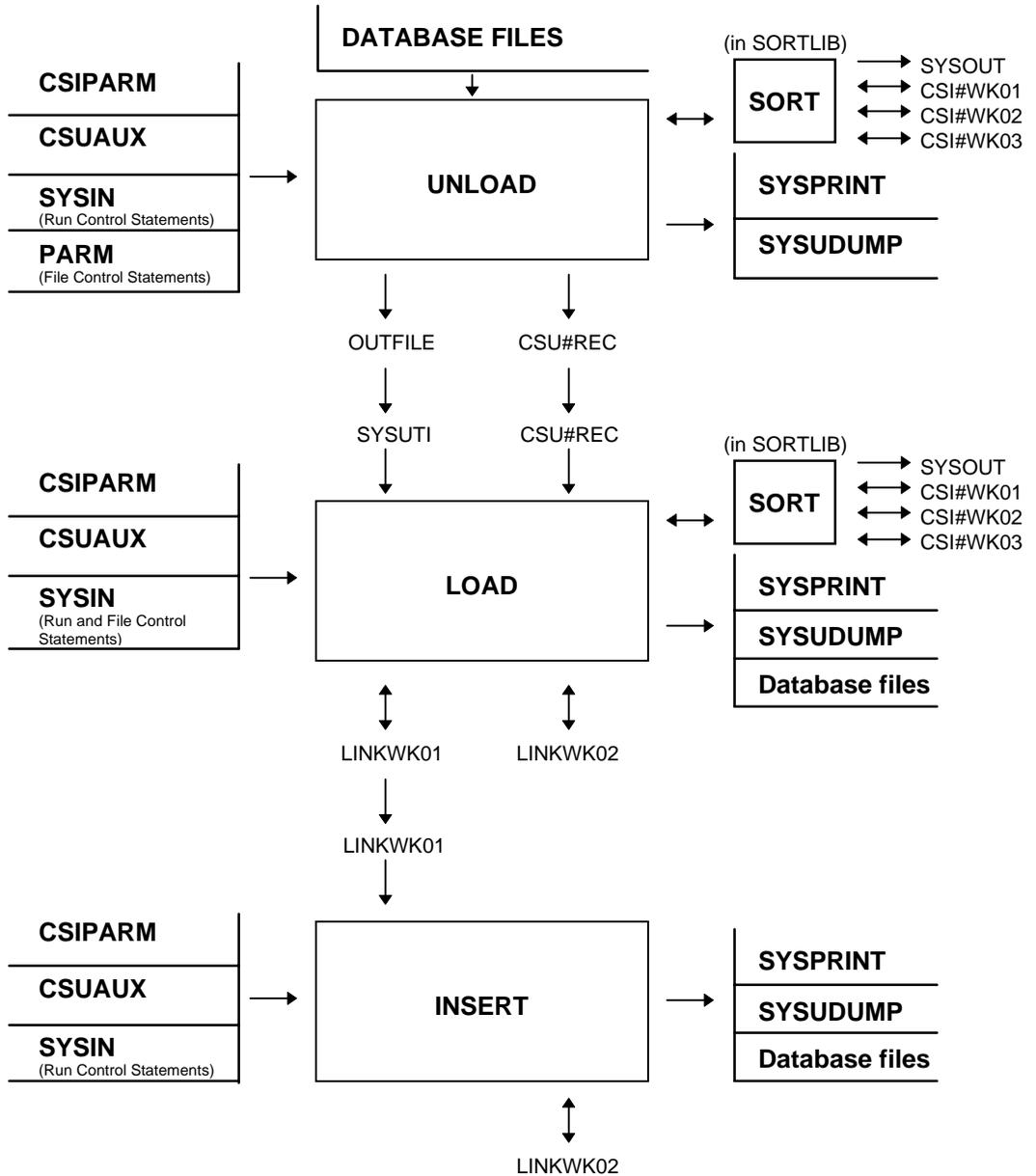
In OS/390, SYSIN and PARM hold these statements for the Unload function; SYSIN holds them for the Load and Insert Linkpath functions.

VSE

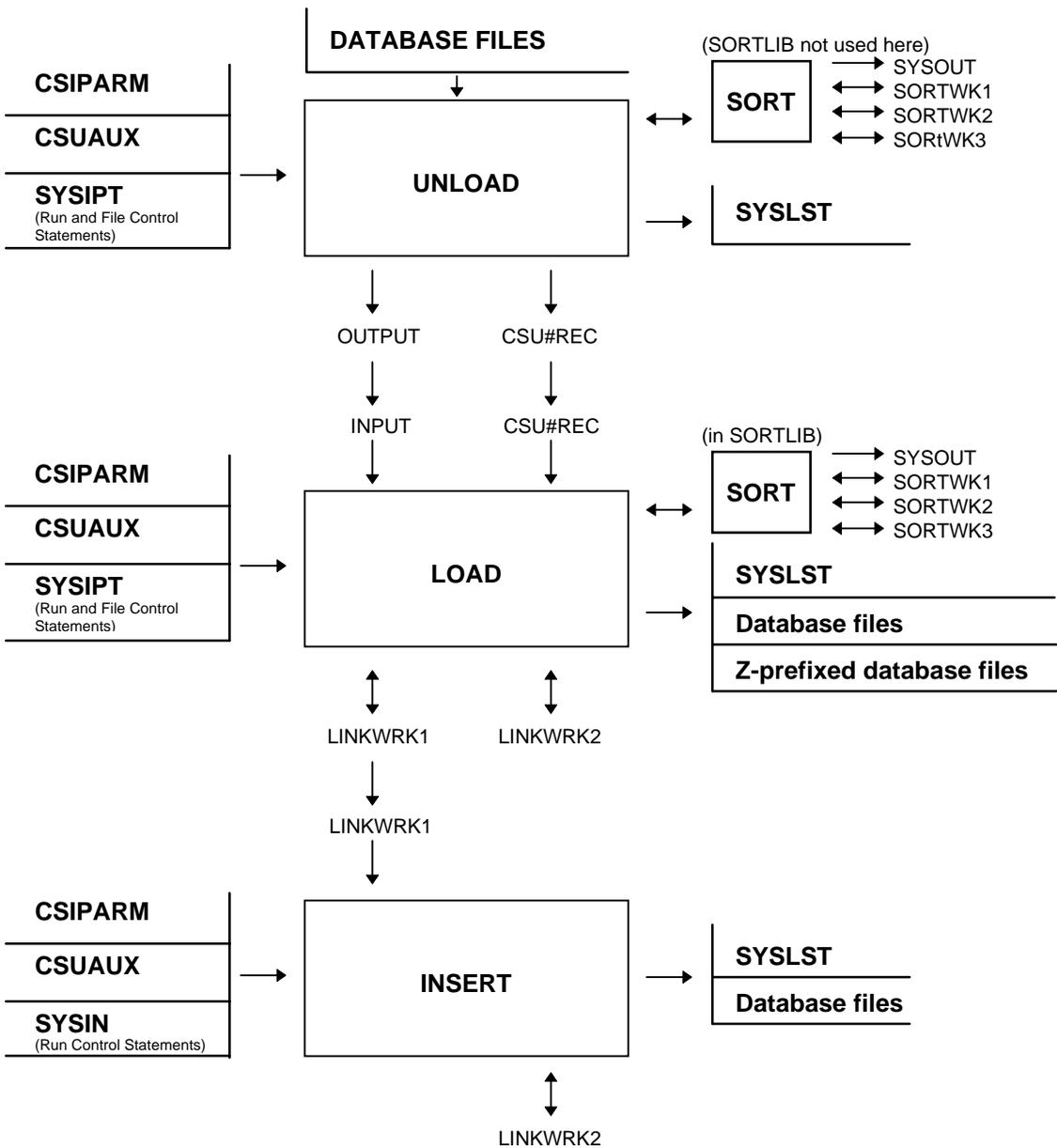
In VSE, SYSIPT holds them for all three functions.

Other files hold output from one function that becomes input to another. The OUTFILE (OUTPUT in VSE) and CSU#REC files hold output from the Unload function and input to the Load function. Similarly, the linkwork files are output from the Load function and input to the Insert Linkpath function. Therefore, you must code these files the same for each function.

Files you define in OS/390 JCL



Files you define in VSE JCL



Unloading PDM files

To use the Version 2 Unload function, you do not code UCL. Instead, you must code the following input:

- ◆ File definitions
- ◆ Run control statements
- ◆ File control statements

You may also insert your own code at exit points.

Defining files

To execute the Unload function in OS/390 or VSE, you define the files listed in the following table in your JCL, and execute the Unload program named CSUNLOAD. In OS/390, rather than coding all the file definitions, you can use the cataloged procedure TISUTUNL. If you want to change the symbolic parameters in TISUTUNL, refer to the *SUPRA PDM and Directory Administration Guide*, P26-2250.

File definitions for the Unload function

DD or file name	Description	Considerations
OS/390 CSI#WKnn	Identifies the sort work files.	VSE For VSE, see SORTWKn. If not enough virtual storage is allocated to sort in place, identify the needed sort work files (CSI#WK01, CSI#WK02, and CSI#WK03). Format and space allocation are identical to standard SORTWKnn statements as defined in the appropriate sort manual.
CSIPARM	Identifies the CSIPARM file, which contains control information that the PDM needs.	See “Coding the CSIPARM file for Unload, Load, and Insert Linkpath” on page 230.
CSU#REC	Holds the number of records that you unloaded for each file.	See “Defining the CSU#REC file” on page 238.
CSUAUX	Holds the auxiliary information to define files that are not in native format.	See “Defining the CSUAUX file” on page 238.
OS/390 fffffff VSE fffffff and Zffffff	Use to define the file you want unloaded.	You may code up to 57 primary and 57 related files. File names must be defined in the SUPRA Directory for the schema you are unloading. VSE For VSE, you must code primary and related files on two separate DLBL statements. Code each file twice: once for direct access with the file name (ffffff) on the DLBL statement and the second time for sequential access with a Z before the file name. Truncate to seven characters (Zffffff).
OUTFILE	Indicates the file where you want data from all unloaded files written.	See “Defining the OUTFILE” on page 241.

DD or file name	Description	Considerations
OS/390 PARM	Holds the file control statements.	See “ Coding file control statements ” on page 254. VSE For VSE, see discussion on SYSIPT.
OS/390 SORTLIB	Indicates the library holding the standard sort program.	This file has no counterpart in VSE.
VSE SORTWK n	Indicates the work files you want used in sorting.	OS/390 For OS/390, see discussion on CSI#WK nn . If insufficient virtual storage is allocated to sort in place, identify the required standard sort work files (SORTWK1, SORTWK2, SORTWK3), as defined in the appropriate sort manual.
OS/390 SYSIN	Holds the run control statements.	See “ Coding run control statements ” on page 242. VSE For VSE, see discussion on SYSIPT.
VSE SYSIPT	Holds the run control and file control statements.	SYSIPT contains two files, separated by a /* control statement. The first file, which contains run control statements, is discussed in “ Coding run control statements ” on page 242. The second file, which contains file control statements, is discussed in “ Coding file control statements ” on page 254.
VSE SYSLST OS/390 SYSPRINT	Indicates the output file for the printed listing of all control statements, diagnostic messages, etc.	
OS/390 SYSOUT	Indicates the file you want the standard sort program to use.	
OS/390 SYSUDUMP	Indicates a dump file.	Optional.

Defining the CSU#REC file

The Unload function creates the CSU#REC file, which passes the number of records to the Load function so that you can unload and load in one job. After the Unload step, the CSU#REC file contains one record for each file you unloaded.

After the Unload function passes the CSU#REC file, the Load utility reads it sequentially. Each record holds a four-byte file name and a fullword binary integer indicating the number of records unloaded for that file. The Load function does not use the rest of the 80 bytes. The file is fixed block and has a block size of 800 bytes.



For VSE, its SYS number is SYS021.

Defining the CSUAUX file

In the CSUAUX file, you describe database files as you want them reloaded, not as they are when unloaded. You must code the files that you want loaded in Series 80 or converted format. You use the CSUAUX file to pass the Unload function the additional parameters from the Series 80 or converted files. When you code the CSUAUX file, you must indicate the format in which you want the files reloaded. For related files, you must indicate the number of records per cylinder and how full you want the files when they are reloaded.



While you do not need to code files that you want loaded in native format, we recommend that you include all your files so that you can set up the CSUAUX file once to use with all jobs.

When you code the files, you must code a separate record for each file; you cannot put more than one file on a record.

You create no problems by including all the files because the PDM ignores statements for files that are not loaded. For example, if you want to reload files in native format, all the statements and parameters would be unnecessary. However, you must include an CSUAUX file in the job stream even if the file is blank.

The PDM also ignores unnecessary parameters and records. If you repeat a parameter in a record, the PDM uses the last one. If you repeat a record for a file, the PDM uses the first one and ignores the others. However, you cannot code null parameter values. When you leave out the value, the function does not use the default.

You must make the format of the CSUAUX file fixed or fixed blocked and the logical record length 80 bytes. The function uses only the first 73 bytes and ignores the rest.

You can separate the parameters with any number of blanks or commas. The parameters are keyword rather than positional; therefore, you do not need to put them in any particular order.

$$\text{FILE} = \text{ffff} \text{RCYL} = \text{nnnn} \text{CYLL} = \left\{ \begin{array}{c} 80 \\ \text{nnnn} \end{array} \right\} \text{LOAD} = \left[\begin{array}{l} \text{COMPATIBILITY} \\ \text{CONVERTED} \\ \text{NATIVE} \end{array} \right]$$

FILE=ffff

Description *Required.* Names the file for which you are passing parameters.

Format 4 alphanumeric characters. The first character must be alphabetic.

Considerations

- ◆ You must code a separate record in the CSUAUX file for each file you want to unload. In each record, you must name the file you want unloaded.
- ◆ If you do not name the file on a record, the Unload function ignores all parameters on that record.

RCYL=nnnn

Restriction *Required* for related files that you want loaded in Series 80 or converted format.

Description Indicates the number of records per logical cylinder.

Format 1–16 numeric characters

Considerations

- ◆ You must code this parameter for all related files that you do not want loaded in native format.
- ◆ As the active records in the Directory or PDM files you are coding increase or decrease, you may need to change this value. The active records may change if you are changing them on the new schema.

$$CYLL = \left\{ \begin{array}{l} 80 \\ nnn \end{array} \right\}$$

Restriction *Required* for related files that you want loaded in Series 80 or converted format.

Description Indicates the maximum percent of each logical cylinder that you want filled with data records during the load function.

Default 80

Options 0–100

Considerations

- ◆ You must code this parameter for all related files that you do not want loaded in native format.
- ◆ As the active records in the Directory or PDM files increase or decrease, you may need to change this value. The active records may change if you are changing them on the new schema.

$$LOAD = \left[\begin{array}{l} COMPATIBILITY \\ CONVERTED \\ NATIVE \end{array} \right]$$

Description *Required.* Specifies the format in which you want the file loaded.

Default NATIVE

Options COMPATIBILITY Database files in Series 80 format.

 CONVERTED Series 80 database files that have been changed to the converted format with the File Convert utility or have previously been loaded in the converted file format. For further information, refer to the *SUPRA Server PDM and Directory Administration Guide (OS/390 & VSE)*, P26-2250.

 NATIVE Newly created SUPRA database files, files that used to be converted, or Series 80 files that have been changed to a native file format.

The following example shows how to code the CSUAUX file's statement:

E\$BR is a primary file, and E\$BI is a related file. You want both loaded in compatibility format. You want the related file loaded with 10,298 records per logical cylinder and a cylinder load limit of 85%.

```
FILE=E$BI,RCYL=10298,CYLL=85,LOAD=COMPATIBILITY
FILE=E$BR,LOAD=COMPATIBILITY
```

In this example, both the primary file, E\$CN, and the related file, E\$CM, are native files. Therefore, you do not need to code the RCYL or CYLL parameter for the related file. Although it is shown in this example, you do not need to code LOAD=NATIVE because it is the default.

```
FILE=E$CN LOAD=NATIVE
FILE=E$CM LOAD=NATIVE
```

Defining the OUTFILE

As indicated in [“File definitions for the Unload function”](#) on page 236, the output file for the Unload function is the OUTFILE in OS/390. In VSE, it is the OUTPUT file. In VSE, you must code the characteristics of the OUTPUT file in the run control records. For more information, see [“Coding the RECFORM statement \(VSE only\)”](#) on page 249.

You use the OUTFILE to hold the records from the files you unload. You must write the unloaded files to the OUTFILE sequentially whether you use tape or disk. The OUTFILE becomes the SYSUT1 input file for the Load function. (In VSE, the OUTPUT file becomes the INPUT file.) Therefore, you must define the file exactly the same way in the JCL for the Unload and Load functions.

In your JCL, you must indicate whether you are using a disk or tape unit and whether the format of the records is fixed or fixed blocked. The latter is recommended. When you code the logical record length (LRECL), add at least eight bytes to the largest logical record length of any file you unloaded. When you code the block size, code it a multiple of the LRECL parameter.

Coding run control statements

After you define the SYSIN file in OS/390 or the SYSIPT file in VSE, you code the run control statements in it. You use the run control statements to indicate to the Unload function the new schema, environment description, primary or related files, and sort program to use.

In some statements, you code only one parameter; in others, you code more than one. In either case, begin each statement in position 1. Some statements may require more than one record. If they do, start each statement on a new record (except for SORTNAME and WORK for VSE). You must code the RELATED: file list before the PRIMARY: file list, and code each list in ascending sequence by file name.

Run control statements for the Unload function

The following table gives you a brief description of the statements.

Statement	Description	Section
DUMP	Indicates whether you want a storage dump printed when errors occur.	"Coding the DUMP statement" on page 244
NEW-SCHEMA / NEW-ENVDESC	Indicates the names of the new schema and environment description you want used when the files are reloaded.	"Coding the NEW-SCHEMA/NEW-ENVDESC C statement" on page 245
RELATED:	Lists the names of the related files you want unloaded.	"Coding the RELATED: statement" on page 247
V-E:	Lists the names of the related files you want unloaded.	"Coding the V-E: statement" on page 248
PRIMARY:	Lists the names of the primary files you want unloaded.	"Coding the PRIMARY: statement" on page 248
S-E:	Lists the names of the primary files you want unloaded.	"Coding the S-E: statement" on page 249
VSE RECFORM	Defines the characteristics of the OUTPUT file.	"Coding the RECFORM statement (VSE only)" on page 249
SORTNAME	Names the sort program if not the standard program.	"Coding the SORTNAME statement" on page 252
VSE WORK	Indicates the number of tape files or disk extents available for intermediate sort storage.	"Coding the WORK statement (VSE only)" on page 253
TEST	Checks and validates all statements without unloading the files.	"Coding the TEST statement" on page 253

Coding the DUMP statement

Use the DUMP statement to indicate whether you want a storage dump printed if Unload encounters errors.

$$\text{DUMP} = \left\{ \begin{array}{l} \text{no} \\ \text{yes} \end{array} \right\}$$

$$\text{DUMP} = \left\{ \begin{array}{l} \text{no} \\ \text{yes} \end{array} \right\}$$

Description *Optional.* Controls printing of a storage dump if Unload encounters errors.

Default NO

Considerations

- ◆ Place this statement first.
- ◆ If you code DUMP=NO, you receive a return code of 12 if Unload encounters errors. If you code DUMP=YES, an error results in abnormal termination with a dump.

Coding the NEW-SCHEMA/NEW-ENVDESC statement

If you are unloading and reloading a file without changing the physical configuration, you should not code NEW-SCHEMA and NEW-ENVDESC statements. If you omit them, the function uses the descriptions in the CSIPARM file for both the old and the new environment.

However, if you change the physical configuration, such as by increasing the number of tracks, you must create the new schema and environment description. If you change some files and not others, you must still code NEW-SCHEMA and NEW-ENVDESC statements.

You must create the new schema and environment description in your Directory files before you unload. To access information it needs from the new schema and environment description, the Unload function signs on separately to the PDM. When it actually unloads the files, the function signs off and then signs back on with the old schema and environment description.

When you code the new schema and environment description, you must include the same elements that you will define in the CSIPARM file for the Load function. In addition, you must code both the NEW-SCHEMA and NEW-ENVDESC statements.

To unload your PDM files, your CSIPARM file must specify your bootstrap schema and environment description in the DIRECTORY parameter, and your schema and environment description in the REALM parameter. For more information on coordinating the CSIPARM file with the NEW-SCHEMA and NEW-ENVDESC statements, see [“Coding CSIPARM file and run control statements for PDM files”](#) on page 230.

NEW-SCHEMA=*schemaname*, NEW-ENVDESC=*envdescname*

NEW-SCHEMA=schemaname

- Restriction** Required if you change the physical configuration of a file.
- Description** *Conditional.* Identifies the schema containing the file definitions you want used when the files are reloaded.
- Format** 1–8 alphanumeric characters

NEW-ENVDESC=envdescname

- Restriction** Required if you change the physical configuration of a file.
- Description** *Conditional.* Identifies the environment description containing the file definitions you want used when the files are reloaded.
- Format** 1–8 alphanumeric characters
- Consideration** Calculations of RQLOC values are based on values in the new schema and new environment description.

Coding the RELATED: statement

Use the RELATED: statement to list the names of the related files you want unloaded. If you do not want to unload any related files, omit this statement.

List the names of the related files in ascending sequence. Code the RELATED: statement before the PRIMARY: statement.

The V-E: statement, which serves the same purpose as the RELATED: statement, is supported for compatibility with existing Series 80 and TIS 1.x job streams. You may use the Series 80 V-E: statement in place of the RELATED: statement. However, use one or the other, not both.

RELATED:rrrr1 [rrrr2 ...rrrrn] END.

or

V-E:vvvv1 [vvvv2 ...vvvvn] END.

RELATED:rrrr1 [rrrr2 ...rrrrn] END.

V-E:vvvv1 [vvvv2 ...vvvvn] END.

Description *Optional.* Identifies the related file(s) you want unloaded.

Format

- ◆ Code RELATED: in positions 1–8 of the first record only, or code V-E: in positions 1–4.
- ◆ You may use up to three records for this statement. If you use more than one record, begin file names in position 1 of the second and third records.
- ◆ Use all 80 positions unless it is the last or only record.

Considerations

- ◆ By using up to three records, you may identify up to 57 files.
- ◆ Unload your PDM files in a separate job from your Directory files.
- ◆ You must code END. immediately after the last file name to indicate the end of the control statement.

Coding the V-E: statement

The V-E: statement serves the same purpose as the RELATED: statement. It is supported for compatibility with existing Series 80 and TIS 1.x job streams. For the format of the V-E: statement, see the RELATED: statement, “Coding the RELATED: statement” on page 247.

Coding the PRIMARY: statement

Code the PRIMARY: statement to list the names of the primary files you want unloaded. If you do not want to unload any primary files, omit this statement.

List names of primary files in ascending sequence. Code all RELATED: statements before PRIMARY: statements.

You may use the Series 80 S-E: statement in place of the PRIMARY: statement. However, use one or the other, not both. The S-E: statement is supported for compatibility with existing Series 80 and TIS 1.x job streams.

PRIMARY:pppp1 [pppp2 ...ppppn] END.

or

S-E:mmmm1 [mmmm2 ...mmmmn] END.

PRIMARY:pppp1 [pppp2 ...ppppn] END.

S-E:mmmm1 [mmmm2 ...mmmmn] END.

Description *Optional.* Identifies the primary file(s) you want unloaded.

Format

- ◆ Code PRIMARY: in positions 1–8 of the first record only, or code S-E: in positions 1–4.
- ◆ You can use up to three records. If you use more than one record, begin file names in position 1 of the second and third records.
- ◆ Fill all 80 positions unless it is the last or only record.

Considerations

- ◆ By using up to three input records, you may identify up to 57 files.
- ◆ Unload your PDM files in a separate job from your Directory files.
- ◆ You must code END. immediately after the last file name to indicate the end of the control statement.

Coding the S-E: statement

The S-E: statement is supported for compatibility with existing Series 80 and TIS 1.x job streams. It serves the same purpose as the PRIMARY: statement. For the format of the S-E: statement, see “[Coding the PRIMARY: statement](#)” on page 248.

Coding the RECFORM statement (VSE only)

Use the RECFORM statement to define the characteristics of the OUTPUT file. The statement is optional because defaults are supplied for all parameters. If you code any of the parameters, you must separate them from any other statement. You can use continuation characters, but you do not need to use continuation characters. Also, you must code them exactly like the parameters you code in the RECFORM statement for the Load utility. For information on the RECFORM statement in the Load function, see “[Coding the RECFORM statement \(VSE only\)](#)” on page 311.

$$\left[\text{RECFORM} = \left\{ \begin{array}{l} \text{FIXBLK} \\ \text{FIXUNB} \end{array} \right\} \right] \left[, \text{DEVICE} = \left\{ \begin{array}{l} \text{DISK} \\ \text{TAPE} \end{array} \right\} \left[, \text{FILABL} = \left\{ \begin{array}{l} \text{NO} \\ \text{STD} \end{array} \right\} \right] \right] \left[\right]$$

$$\left[, \text{BLKSIZE} = \left\{ \begin{array}{l} 1000 \\ \text{N} \end{array} \right\} \right]$$

$$\left[, \text{RECSIZE} = \left\{ \begin{array}{l} 100 \\ \text{N} \end{array} \right\} \right] \left[, \text{DEVADDR} = \left\{ \begin{array}{l} \text{SYS030} \\ \text{SYS}nnn \end{array} \right\} \right]$$

$$\left[\text{RECFORM} = \begin{cases} \text{FIXBLK} \\ \text{FIXUNB} \end{cases} \right]$$

Description *Optional.* Indicates the format of the records in the file.

Default FIXBLK

Options FIXBLK Fixed-length, blocked records
 FIXUNB Fixed-length, unblocked records

$$, \text{DEVICE} = \begin{cases} \text{DISK} \\ \text{TAPE} \end{cases}$$

Description *Optional.* Indicates the device type of the file.

Options DISK Disk device
 TAPE Magnetic tape unit

$$, \text{FILABL} = \begin{cases} \text{NO} \\ \text{STD} \end{cases}$$

Restriction Valid only when DEVICE=TAPE.

Description *Optional.* Indicates whether the tape contains file labels.

Default NO

Options NO Does not contain labels
 STD Contains standard labels

$$, \text{BLKSIZE} = \begin{cases} \text{1000} \\ n \end{cases}$$

Description *Optional.* Indicates the file's block size.

Default 1000

Format Use numeric characters.

Consideration You must code a value that is a multiple of the value in the RECSIZE parameter.

$$,RECSIZE = \left\{ \frac{100}{n} \right\}$$

Description *Optional.* Indicates the file's record size in bytes.

Default 100

Format Use numeric characters.

Considerations Add the following items to determine the value of this parameter:

- S The sum of the lengths of the data elements you want unloaded plus the length of the control key. Calculate this for each file you unload and use the largest value.
- +4 The length of the file name. Always add this value.
- +X where X is:
 - +2 The length of the record code. Add this if you are unloading at least one related file with coded records and are not unloading any primary files.
 - +4 The RQLOC. Add this if unloading primary files, regardless of the above.
 - +0 If neither of the above.

$$,DEVADDR = \left\{ \begin{array}{l} \underline{\text{SYS030}} \\ \text{SYS}nnn \end{array} \right\}$$

Description *Optional.* Indicates the device address (SYS number symbolic unit) associated with the file.

Default SYS030

Format *nnn* Must be 3 digits

Coding the SORTNAME statement

Use the SORTNAME statement to name the sort program you want used.

SORTNAME =	}	IERRCO00	OS/390
		SORT	VSE
		progrname	

Description *Optional.* Identifies the sort program.

Default OS/390 IERRCO00

VSE SORT

Format 1–8 alphanumeric characters

Consideration VSE For VSE, code the SORTNAME and WORK statements on the same record. Separate the statements with a comma.

Coding the WORK statement (VSE only)

Use the WORK statement to indicate the number of tape drives or disk extents available for intermediate storage during sorts.

$$\text{WORK} = \left\{ \begin{array}{l} 1 \\ n \end{array} \right\}$$

Description	<i>Optional.</i> Indicates the number of tape drives or disk extents available for intermediate storage during sorts.
Default	1
Options	1–9
Consideration	Code the WORK and SORTNAME statements on the same record. Separate the statements with a comma.

Coding the TEST statement

Use the TEST statement to indicate whether the Unload function should validate the run control statements. If you code TEST=YES, the Unload function opens and closes the SUPRA files, but does not unload them. It points out any errors in the run control statements, so you can correct them. To actually unload files, code TEST=NO.

$$\text{TEST} = \left\{ \begin{array}{l} \text{NO} \\ \text{YES} \end{array} \right\}$$

Description	<i>Optional.</i> Indicates whether the Unload utility is to actually unload the files or just analyze the statements.	
Default	NO	
Options	NO	The Unload function analyzes the statements and unloads the files.
	YES	The Unload function validates all run control statements without actually unloading the files.

Coding file control statements

After defining the files and coding the run control statements, you code file control statements. For OS/390, you must code the file control statements in the PARM file. For VSE, you must code them in a second SYSIPT file. You must put the second SYSIPT file directly after the run control statements and end-of-file record in the first SYSIPT file.

Use the file control statements to define the layout of the output data record and select and order the data you want unloaded. You code some of these statements with several parameters while you code others with only one. You must code some statements with more than one record. In that case, start each statement on a new record. Code the file name in positions 1–4, the parameters in positions 5–76, and leave positions 77–80 blank.

Code the statements for the related files first, and then those for primary files. For each related file, you may code a LINKPATH statement if you want, and then you must code an Element List statement. For each primary file, you must code an Element List statement, and then you may code a BLANK-LINKS statement if you like.

You must code file control statements for all files that you coded in the RELATED: and PRIMARY: run control statements. In addition, you must code them in the same order.

File control statements for the Unload function

The following table gives a brief description of each statement you need to code.

Statement	Description	Reference
LINKPATH	Indicates the linkpath you want used as an access linkpath to unload a related file.	“Coding the LINKPATH statement” on page 255
Element list	Indicates the data elements you want unloaded from a particular file.	“Coding the Element List statement” on page 257
BLANK-LINKS	Indicates the primary file linkpaths you want blanked while unloading the primary file.	“Coding the BLANK-LINKS statement” on page 263

Coding the LINKPATH statement

Use the LINKPATH statement to identify the access linkpath for unloading a related file. You can unload a file faster if you use the primary access linkpath because it is clustered for faster performance. The other linkpaths are not clustered. When you identify the access linkpath, you must name the same one in both the Unload and the Load functions.

You must select an access linkpath and its associated key from the base portion of a record. If you select coded records, you cannot use a linkpath in the redefined portion. You must code a LINKPATH statement immediately before the Element List statement with which it is associated.

If you want to unload several, but not all, coded records, code several LINKPATH statements and several element lists. The following sequence illustrates how to do this:

```
TV01LINKPATH=ppppLKxx
TV01ALL.END
TV02LINKPATH=ppppLKxx,RC=01
TV02TV02CODETV02KEYITV02DATAEND.
TV02LINKPATH=ppppLKxx,RC=02
TV02TV02CODETV02KEYITV02DATAEND.
```

[rrrrLINKPATH=ppppLKxx]

[,PRESERVE = { $\left. \begin{array}{l} \text{NO} \\ \text{YES} \end{array} \right\}$]

[,RC=yy]

rrrrLINKPATH=ppppLKxx

Description	<i>Optional.</i> Identifies the access linkpath you want used to unload a related file.	
Format	<i>rrr</i>	4-character related file name.
	<i>ppppLKxx</i>	The linkpath name as coded in the schema.

$$\left[,\text{PRESERVE} = \left\{ \begin{array}{l} \text{NO} \\ \text{YES} \end{array} \right\} \right]$$

Description *Optional.* Indicates whether to preserve the order of the records on the access linkpath chains.

Default NO

Considerations

- ◆ If you code PRESERVE=NO, the Unload function uses the standard sort program. It sorts related file data in ascending sequence by access control key. (The Unload function sorts primary files in ascending sequence by RQLOC.)
- ◆ If you code PRESERVE=YES, the Unload function writes the records in sequence by the linkpath you code without sorting the records.
- ◆ Do not code PRESERVE=YES if the related file has integrity problems. If you do, the Load function determines that duplicate linkpaths exist and does not load the file.
- ◆ If chains are broken and you want to preserve the order of the record, code PRESERVE=NO and use an exit program to control the sorting sequence.
- ◆ **VSE** In VSE when you code PRESERVE=YES, code the files as direct access in your JCL. When you code PRESERVE=NO, code the file as sequential access.
- ◆ If you code the PRESERVE clause, it must appear on the same line as the LINKPATH statement.

,RC=yy

Restriction Use for coded records only.

Description *Optional.* Use this parameter to unload only the records with the record code you supply.

Format 2-character record code as defined in the Directory.

Considerations

- ◆ If you want to code elements in your element list that are in the redefined portion of a coded file, use the Version 1 Unload function for that file. You can use the Version 1 functions for one file and use the Version 2 functions for the rest of the files.
- ◆ You must code the statements for the record codes in the same order as they are defined in the schema.

Coding the Element List statement

Use the Element List statement to indicate the data elements you want unloaded from the file. The unloaded data is sequentially written to the OUTFILE discussed in “[Defining the OUTFILE](#)” on page 241. The formats of primary and related records in the OUTFILE are different when you supply an element list and when you request all the elements. For an illustration of the different formats, see the figure at the end of this section.

```
fff { element, [* FILL = nn elementz * FILL = nn] } END.  
    { ALL. }
```

fff

Description	<i>Required.</i> Identifies the file.
Format	4-character file name as coded in the schema. Code the file name in positions 1–4.

{ *element*, [** FILL = nn elementz * FILL = nn*] }
 { ALL. }

Description *Required.* Identifies the individual element(s) or all elements (ALL.).

Format To unload all elements, code ALL. in positions 5–8. For an element list, use positions 5–76. Positions 77–80 are ignored.

Considerations

- ◆ You may specify between one and 100 data elements on a maximum of 12 records.
- ◆ For a related file containing coded records, you must supply the record code as the first element and the control key associated with the specified linkpath as the second. For noncoded, related records, supply the control key as the first element. For a primary file, code the name of the control key as the first element.
- ◆ To specify particular record codes, you must provide a separate LINKPATH statement and element list for every record code you are unloading. When you code each LINKPATH statement, you must append the RC (record code) parameter. In addition, you must put the record codes in the same order as in the schema.
- ◆ Generally, if you need to refer to the redefined portion of coded records at a level more detailed than *rrrrDATA*, you can do so only with the Unload function.

Since you cannot code elements in the redefined portion with the Version 2 Load function, you must use the [Version 1 Unload and Load](#) functions for that file.

If you still want to refer to the redefined portion in the Version 2 Unload function, you should not leave any of that portion undefined or define any part with the FILLER element name. In addition, if you are changing this area for the database you are loading, you must unload only the data you are reloading or you must use the **FILL=nn* parameter in the Unload record to add space for new elements.

- ◆ For related files, you must code the Element List statement immediately after the LINKPATH statement to which it applies.
- ◆ If you are unloading linkpaths for primary files, the linkpaths must follow the control key in the element list and you must define them in the same order as in the schema.

- ◆ You can add elements, delete them, and increase their size with the *FILL=*nn* parameter. However, you cannot decrease their size with the *FILL=*nn* parameter. To do that, you must use an exit program at exit point 20 or 30.

To add an element so it is automatically filled with blanks, do not code the element name in the Unload function's element list. Code the element name at the end of the Load function's element list.

You can also add an element so it appears in the data record where you can modify it with an exit program. To add an element, code the *FILL=*nn* parameter in the Unload function's element list where you want the element to appear in the data record. In the Load's element list, code the element name where it will map to the same portion of the data record.

To delete an element, do not unload or load it. That is, leave it out of the element lists in both functions.

To increase the size of an element, code *FILL=*nn* in the Unload function's element list. Replace *nn* with the number of bytes you want to add to the element. You can code *FILL=*nn* before or after the element name. You can also code it twice, both before and after the name. In the Load's element list, code just the element name.

Bytes are added in the Unload function and automatically set to blanks. You can modify these blanks with an exit program in either the Unload or Load function.

To decrease the size of an element, first code the element name in the Unload function's element list. Include an exit program at exit point 20 or 30 to change the size. Code the exit program to shift the data in the record before it goes to the OUTFILE in the Unload function. Shift the data you want to keep so it covers the data you want to delete.

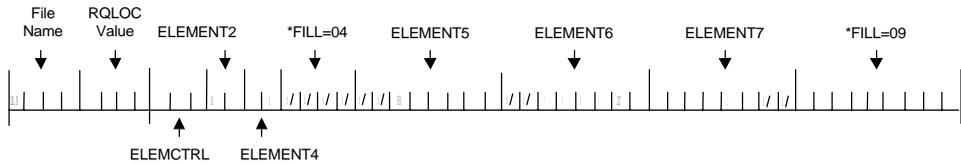
Whenever you change the elements or their size, you must make similar changes in the schemas you use for the Unload and Load functions. For an example of how to account for internal schema changes in your element list, assume you are making the following changes:

- Deleting ELEMENT3 with eight bytes
- Adding ELEMENT8 with five bytes
- Adding ELEMENT9 with four bytes
- Decreasing the size of ELEMENT6 from eight to four bytes by removing the first four bytes.
- Adding four bytes to the front of ELEMENT5 to increase its size from eight to 12 bytes

To make these changes, code this element list in the Unload function:

```
ELEMELEMCTRLELEMENT2ELEMENT4 *FILL=04ELEMENT5ELEMENT6
ELEMENT7 *FILL=09
```

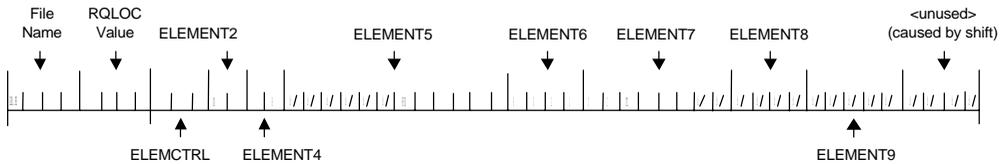
As a result, your data record would look like this:



In the Load function, code this element list:

```
ELEMELEMCTRLELEMENT2ELEMENT4ELEMENT5ELEMENT6
ELEMENT7ELEMENT8ELEMENT9
```

During execution, the data records are passed from the Unload to the Load function by way of the OUTFILE (or INPUT in VSE). To decrease the size of ELEMENT6 by four bytes, you shifted ELEMENT7, ELEMENT8, and ELEMENT9 to the left four bytes with an exit program. Thus, this element list maps to the same data record as follows:



As you can see, the number of spaces in the *FILL parameter actually appear in the data record as spaces. You can modify this space with exit programs in both the Unload and Load functions.

The newly added ELEMENT8 and ELEMENT9 do not appear because they are automatically filled with spaces. These spaces do not show up in the data record because you did not code the *FILL=09 in the Unload element list to create the necessary spaces.

- ◆ Since you cannot exceed the maximum of 12 records per file, the space required to code each *FILL=*nn* parameter may reduce the total number of data elements permitted for the Unload function.
- ◆ You may expand two adjacent elements with only one *FILL=*nn* parameter if the first element is at the end of the field and the second is at the beginning of the field. To do this, code a value equal to the total number of spaces required for both elements.
- ◆ The bytes inserted by the *FILL parameter are set to spaces (X'40') in the output record. You may need to increase the LRECL parameter for OUTFILE to account for the extra fields.
- ◆ If you code ALL. in the Unload function, code it also in the Load function.
- ◆ All element names you code must have been previously defined for the file in the appropriate schema. That is, in the Unload element list, the elements must be defined in the schema used to unload the old schema. In the Load element list, the elements must be defined in the schema used to load the new schema, or if there is no change in schemas, the old schema.
- ◆ It is not necessary to specify the elements in the same order as they appear in the schema.
- ◆ When you are unloading and loading to recreate related files and not coding ALL. in the element list, you must code all control keys defined for each unloaded file. This includes the control key for the access linkpath as well as the control keys for all the secondary linkpaths. Coding of the control keys helps ensure database integrity. A control key not defined in the new schema is an exception.
- ◆ You must code END. immediately after the last element to indicate the end of the control statement.

ELEM-LIST = Element list you supply

Primary Record Format ▶	ffff PRIMARY FILE NAME	RQLOC VALUE	CONTROL- KEY	All remaining data elements and LINKPATHS as coded on the ELEM-LIST control card(s) (CONTROL-KEY is excluded).
Standard Related Record Format ▶	ffff RELATED FILE NAME	ACCESS-KEY		All remaining data elements as coded on the ELEM-LIST control card(s) (ACCESS-KEY is excluded).
Coded Related Record Format ▶	ffff RELATED FILE NAME	RECORD CODE	ACCESS- KEY	All remaining data elements as coded on the ELEM-LIST control card(s) (ACCESS-KEY is excluded).

ELEM-LIST = ALL. END.

Primary Record Format with Linkpaths in File ▶	ffff PRIMARY FILE NAME	RQLOC VALUE	CONTROL- KEY	All LINKPATHS in the order in which they appear in the internal record	All data elements excluding the root field in the order in which they appear in the internal record control card(s) (CONTROL-KEY excluded).
Primary Record Format without Linkpaths ▶	ffff PRIMARY FILE NAME	RQLOC VALUE	CONTROL- KEY	All data elements excluding the root field in the order in which they appear in the internal record (CONTROL-KEY excluded).	
Standard Related Record Format ▶	ffff RELATED FILE NAME	ACCESS-KEY		All data elements in the order in which they appear in the internal record (ACCESS-KEY excluded).	
Coded Related Record Format ▶	ffff RELATED FILE NAME	RECORD CODE	ACCESS- KEY	All data elements in the order in which they appear in the internal record (RECORD CODE and ACCESS-KEY excluded).	

Coding the BLANK-LINKS statement

Use the BLANK-LINKS statement to identify the linkpaths you want cleared while unloading the primary file. The linkpaths are set to blanks on the Unload output file (OUTFILE), but not on the primary file from which they were unloaded.

```
ppppBLANK-LINKS=LKxx1 [LKxx2 ...LKxxn]JEND.
```

```
ppppBLANK-LINKS=LKxx1 [LKxx2 ...LKxxn] END.
```

Description *Optional.* Identifies the primary file (*pppp*) containing the linkpaths (LKxx) you want blanked.

Format

<i>pppp</i>	4 alphanumeric character primary file name
<i>xx</i>	2 alphanumeric linkpath identifier

Considerations

- ◆ You must code the same primary file (*pppp*) that you coded in the Element List statement immediately before this statement.
- ◆ You can code a maximum of 14 linkpaths (LKxx) on this statement. Code only one statement per file.
- ◆ When the Load function reloads the file, it inserts valid linkpath data into all linkpaths for which there is link data. If there is no link data, that linkpath chain is empty.
- ◆ You code the BLANK-LINKS statement for a linkpath depending on whether the related file to which it is connected is unloaded and loaded at the same time as its primary file. If the related file is loaded, code its linkpath to be blanked. If the related file is not loaded, do not code its linkpath to be blanked.
- ◆ If you incorrectly code a linkpath to be blanked, the valid chain information is deleted. It will appear that there are no related file records connected to that primary file. Since your database is corrupted, you will have to execute the Unload and Load functions correctly to recreate the chain information.
- ◆ If you neglect to code a linkpath to be blanked, it retains its current chain information. Because the information is no longer valid, your database will be corrupt. To clear and recreate the chain information, you must execute the Unload and Load functions correctly.
- ◆ You must code END. immediately after the last linkpath to indicate the end of the control statement.

Unloading Directory files

Unload the Directory files in a separate job from your PDM files. When you unload Directory files, you must define the same files as you did to unload PDM files. For directions, see [“Defining files”](#) on page 235. When you unload the related files, always code PRESERVE=YES to preserve the order of the records.

While you need run control and file control statements, you do not need to code them. The run control statements are provided in the data member, CSUSUNLD. The file control statements are in data member, CSUPUNLD. After installation is complete, you may alter only the NEW-SCHEMA and NEW-ENVDESC statements in CSUSUNLD. You cannot alter any statements in CSUPUNLD.

In your CSIPARM file, code the bootstrap schema and environment description. If you code NEW-SCHEMA and NEW-ENVDESC statements, you must code the bootstrap schema and environment description in the REALM parameter of your CSIPARM file. In that case, do not code the DIRECTORY parameter. [“Coding CSIPARM file and run control statements for directory files”](#) on page 231 shows how to coordinate coding your CSIPARM file with your run control statements.

Using exit points

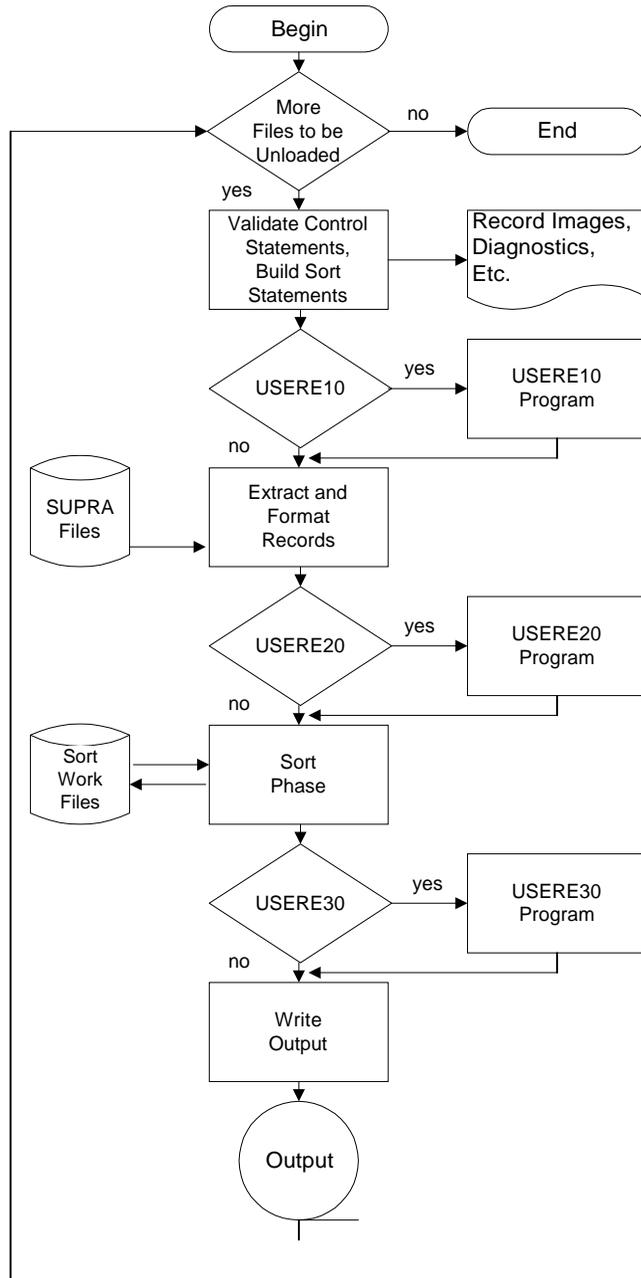
There are three exit points in the Unload Function:

- ◆ At exit point 10 (entry point USERE10), your program can add sort parameters to those the Unload function automatically uses: the file name and a control key. During normal operation, the Unload function automatically sorts the records in a file according to the file name and a control key. The function can sort according to any additional record fields you code.
- ◆ At exit point 20 (entry point USERE20), your program can modify or delete records before the Unload function sorts them. Your program cannot add records at this point.
- ◆ At exit point 30 (entry point USERE30), you can modify or delete records after the Unload function sorts them. You can also add records in sequence with those you are unloading.

The following figure illustrates the exit points in the Unload function where the function can access each of the exit programs. When you take the exits, the Unload function automatically passes control to your exit program at the proper point in the processing cycle. After your exit has executed, control automatically returns to the Unload function.

To use your exit programs, you must relinkedit the Unload function with the linkdeck, CSUULKUN. Include in the Linkage Editor input the exit module(s) containing the entry points you want to use: USERE10, USERE20, and/or USERE30. Do this by adding INCLUDE control statements after the INCLUDE statement for module CSUUMRND in the CSUULKUN linkdeck Cincom has supplied.

The name of your exit program must be a CSECT name or entry point in your program. While you must write your program for exit point 10 in Assembler, you may write programs for the other two in either COBOL or Assembler. Assembler language examples for all three exit points and COBOL examples for exit points 20 and 30 are provided.



Exit Points

Using exit point 10

At exit point 10 (USERE10), you can code additional fields to use as sort parameters. The Unload function sorts records according to file name and then control key. For a primary file, there is no need to sort by any additional fields because the control keys are unique. Sorting by other fields does not change the order of the records.

However, for a related file, you may continue sorting by another field. When the Unload function sorts related records by control key, it groups together all records in a chain on the access linkpath. If you sort by other fields, you can order the groups of records within the same control key to control the order of the records on the chain.

Whether you code additional sort parameters or not, the Unload function builds a SORT statement after validating the run control and file control statements. Before reading the records from each file, it checks to see whether you have included a program for exit point 10.

If so, the Unload function sets up register 1 with the address of a 3-word parameter list. In the first word, it puts the address of the first character of the SORT statement. In the second word, it puts the address of the last character of the SORT statement. In the third word, it puts the address of a 4-character field with the name of the file you are unloading. More information on the registers is in [“Using registers”](#) on page 269.

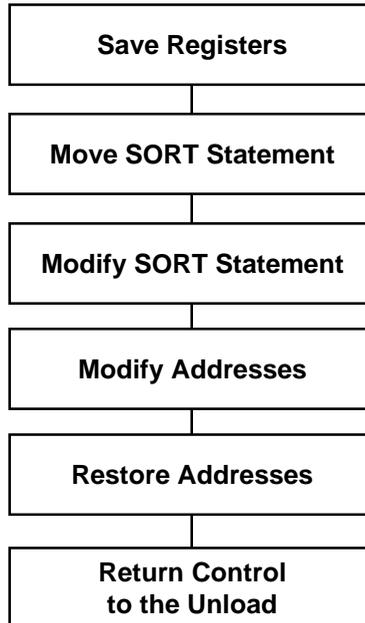
The Unload function passes its SORT statement to your program when it passes control to it, that is, once for each file you unload. If your exit program will modify the SORT statement, it must first move the statement to a work area large enough to hold it plus your additional sort parameters.

When your program builds a new SORT statement, it must retain the file name and control key used by the Unload function. When you add sort parameters, you must code them according to the restrictions in the appropriate sort manual.

After your exit program has built the new SORT statement, it must change the first two full words pointed to by register 1 so that they point to the first and last characters of the new SORT statement. When your program is finished, it returns control to the Unload function through register 14.

The following figure shows the steps that your exit program must take.

If you coded PRESERVE=YES in the LINKPATH statement for a file, the Unload function does not call your exit program for that file; it passes the SORT statement it generated to the SORT program unchanged. To preserve the order of the records on the linkpath, the Unload function writes the records in sequence without sorting them.



Using registers

When the Unload function calls your program at exit point 10, these four registers contain the following information. When your exit program returns control to the Unload function, it must restore all 16 registers to their contents at entry.

Register	Function
R1	Address of a 3-word parameter list. The first word contains the address of the first character of the SORT statement. The second word contains the address of the last character of the SORT statement. The third word contains the address of the four-character field with the file name.
R13	Address of a standard 72-byte save area. In this area, your exit program saves the contents of the registers when it enters. When it exits, your program restores them from this area.
R14	Return address. Your exit program returns control to the Unload function at this address.
R15	Address of entry point USERE10.

Retaining values in the SORT statement

You need to retain the values in the SORT statement that the Unload function generates, so you can include them in the new SORT statement that your program generates. The Unload function generates them in the following format:

```
OS/390  bSORT FIELDS=(1,xxxxx,CH,A,yyyyy,zzzzz,CH,A),SIZE=Evvvvvvvb
```

```
VSE  SORT FIELDS=(1,xxxxx,CH,A,yyyyy,zzzzz,CH,A),WORK=nb
```

xxxxx

Description Represents a 5-byte numeric field indicating the length of the first sort parameter. This sort parameter consists of the file name for related files and the file name plus the RQLOC value for primary files.

yyyyy

Description Represents a 5-byte field indicating the position of the second sort parameter within the unloaded record. This sort parameter is the linkpath control key.

zzzzz

Description Represents a 5-byte numeric field indicating the length of the second sort parameter.

vvvvvvv

Description Represents a 7-byte numeric field indicating the estimated number of records to be sorted.

n

Description Represents a 1-byte numeric field defining the number of work files available to the sort program.

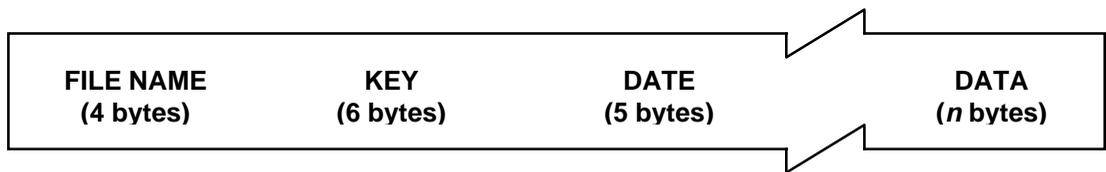
For a complete description of the sort program and the sort statement, refer to your sort manual. Also, see the examples in “[Sample programs for exit point 10](#)” on page 271.

Sample programs for exit point 10

Two examples show how to use a program at exit point 10 to sort records. The Unload function sorts only on the key field. If you want to sort by another field as well, you can insert a program here. The first example shows a specific solution in Assembler for the sample problem given below. After the specific solution is a generalized program that you can use for the sample problem or any similar one. All you need to do in the generalized program is substitute your own values (file names, etc.) in the appropriate places.

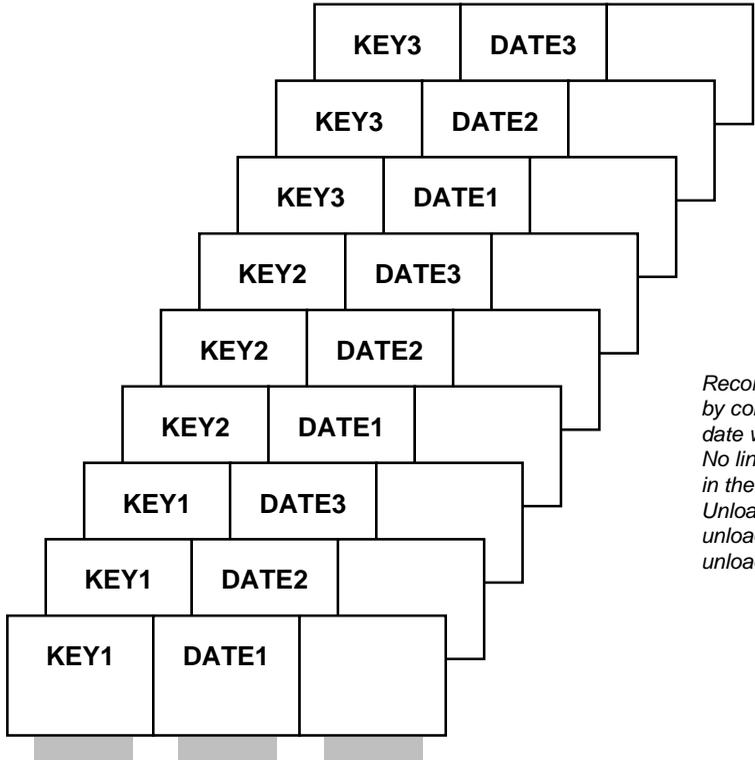
The following problem is the basis for the two examples:

Assume you wish to unload a related file which, when unloaded, has records in the OUTPUT file in the following format:



Also assume you want to keep the records in sequence by key and date.

Record sequence by Unload SORT parameters and extended SORT parameters provided by program at exit point 10. Since the Unload function sorts only on the key, you may use a program at exit point 10 to code the date field as an additional sort field as shown in the following figure.



Records are in sequence by control-key and by date within control-key. No linkpaths are shown in the records because the Unload function does not unload linkpaths when unloading records.

Control- key field	Date field	Data field
--------------------------	---------------	---------------

OS/390 The following is an Assembler program for exit point 10 to run in OS/390:

USERE10	CSECT		Entry point.
	USING	*,R15	R15 will be the base register.
	STM	R14,R12,12(R13)	Save Unload's registers.
	L	R2,0(,R1)	Pick up starting address of SORT statement.
*			
	MVC	SRTSTMT(43),0(R2)	Move the first 43 characters of the old SORT statement to work area.
*			
	MVC	SRTSIZE(16),43(R2)	Move estimated total number of records to be sorted to SRTSIZE.
*			
	LA	R2,SRTSTMT	Start of SORT statement.
	ST	R2,0(R1)	Store new start of SORT statement in first word of *
*			parameter list.
	LA	R2,ESRTSTMT	End of SORT statement.
	ST	R2,4(,R1)	Store new end of SORT statement in second word of parameter *
*			list.
	LM	R14,R12,12(R13)	Restore Unload's registers.
	BR	R14	Return to Unload.
SRTSTMT	DC	C' SORT FIELDS='	Area for old SORT parameters.
	DC	C'(1,xxxxx,CH,A, '	
	DC	C'YYYYY,zzzzz,CH,A'	
MYSRT	DC	C',11,5,CH,A'	Additional SORT parameters.
SRTSIZE	DC	C'),SIZE=Evvvvvvv'	
ESRTSTMT	DC	C' '	End of new SORT statement.
	END		

VSE The following is a program for exit point 10 written for VSE:

```

USERE10      CSECT      Entry point.
              USING     *,R15      R15 will be the base register.
              STM       R14,R12,12(R13)  Save Unload's register.
              L         R2,0(,R1)      Pick up starting address of SORT
*                                     statement.
              MVC       SRTSTMT(42),0(R2)  Move the first 42 characters of
*                                     the old SORT statement to
*                                     work area.
              MVC       SRTWORK(9),42(R2)  Move number of available work
*                                     files to work area.
              LA        R2,SRTSTMT      Start of SORT statement.
              ST        R2,0(,R1)      Store new start of SORT
*                                     statement in first word of
*                                     parameter list.
              LA        R2,ESRTSTMT     End of SORT statement.
              ST        R2,4(,R1)      Store new end of SORT statement
*                                     in second word of parameter
*                                     list.
              LM        R14,R12,12(R13)  Restore Unload's registers.
              BR        R14            Return to Unload.
SRTSTMT      DC         C'SORT FIELDS='  Area for old SORT parameters.
              DC         C'(1,xxxxx,CH,A,'
              DC         C'yyyyy,zzzzz,CH,A'
MYSRT       DC         C',11,5,CH,A'
SRTWORK      DC         C'),WORK=n'
              DC         C' '
              END
    
```

The following is a generalized program written for OS/390 and VSE:

You need to code and link this program only once for the environment where you want to execute it. In this example, a load module containing additional sort fields was created for each file where you want to use additional sort fields. The program for exit point 10 searches a table for the file currently being processed. If found, it loads the module corresponding to that file. It then builds the SORT statement in the loaded module by adding the segments of the original SORT statement to the additional sort fields. The program returns pointers to the beginning and end of this new SORT statement.

```

USERE10      CSECT                Entry point.
             STM          R14,R12,12(R13)    Save Unload's registers.
             LR           R12,R15           Load Base register.
             USING        USERE10,R12      R12 will be the base register.
             ST           R1,SAVE1         Save register R1.
             LM           R2,R4,0(R1)      Load the registers with
*                                                    parameters - R2 contains the
*                                                    starting address of the SORT
*                                                    statement. R3 contains ending
*                                                    address of SORT statement, and
*                                                    R4 contains address of file name.
*
* The following code is a simple table search:
*
             LA           R11,TABLE-4      Point R11 one entry short of
*                                                    first entry in table.
             LA           R6,(TABEND-TABLE)/4  Place number of entries in table*
*                                                    in R6.
LOOP         LA           R11,4(,R11)      Increment R11 to point to next *
*                                                    entry in table.
             CLC          0(4,R4),0(R11)   Compare table entry with file
*                                                    name supplied as SORT parameter.
             BE           EXITN           Branch if file name is in table.
             BCT          R6,LOOP         Continue searching table unless *
*                                                    searched entire table.
RETURN      LM           R14,R12,12(R13)   Restore Unload's registers.
             BR           R14            Return control to Unload.

```

The following code constructs a module name:

```

EXITN      DS           0H                This routine executes only when
*                                                    the table of file names
*                                                    contains a match against the
*                                                    file name passed as the third
*                                                    input parameter.

```

At this point, the code for OS/390 and VSE differs.

OS/390 Environments

```

*      MVC      MODPRFX,0(R4)      R4 points to the third input
*                                     parameter (SUPRA file name).
*                                     This instruction builds a string
*                                     of the form ffffUS10.
*      LOAD      EPLOC=MODNAME      The load module, whose
*                                     8-character name is now
*                                     found at MODNAME, is loaded into
*                                     virtual storage and its address
*                                     is returned in R0.
*      LR        R1, R0             Register 1 now contains the *
*                                     address of the first byte of
*                                     module ffffnUS10.
*

```

The following instructions move additional sort field definitions into the original SORT statement for the file.

```

*      LM        R4,R5,0(R1)        Pick up start and end addresses
*                                     of new SORT statement.
*      MVC        0(43,R4),0(R2)     Move original sort fields.
*      MVC        0(16,R5),43(R2)    Move original size parameter.
*      L          R1,SAVE1           Restore original register 1.
*      LA         R5,16(,R5)         Bump to end of SORT.
*      STM        R4,R5,0(R1)        Store addresses of modified SORT
*                                     statement in SORT parameter list.
*      B          RETURN             Branch to RETURN.

```

The following table requires one entry for each file requiring additional parameters in the SORT statement:

```

TABLE   DS      0F                Header label.
        DC      C'ffff1'          Appropriate number of four-
        DC      C'ffff2'          character file names.
        DC      C'ffffn'
        .
        .
        DC      C'END.'           Code this only if you code no
*                                     file name.
TABEND  EQU     *                 End of the above list.

```

The following area contains the constructed module name passed as the parameter to the load module:

```

MODNAME DS      0D
MODPRFX DC      CL4'ffff'         File name moved from table.
MODEND  DC      CL4'US10'        Any four characters you choose
*                                     as a common ending for the
*                                     loaded module names.
SAVE1   DS      F                Area to save original parameter
*                                     pointer.
        END

```

The following code illustrates a sample of a module loaded by the preceding program. You must replace *ffffn* with the name of a file for which you want to add a sort field. You must also code the same file name in the table at label TABLE in the preceding program. You can code any valid sort parameters at label YOURSORT. You can code as many of these modules as you need.

```
ffffnUS10      CSECT                Use 4-character file name.
                DC                A(SRTBEG)          Required as shown.
                DC                A(SRTEND)          Required as shown.
SRTBEG         DC                CL43' '            Required as shown.
YOURSORT       DC                C',11,5,CH,A'       Additional SORT parameters.
*                                                       The date starts 11 bytes into *
                                                       the record and is 5 bytes long.
SRTEND         DC                CL16' '            Required as shown.
                END
```

The following routine constructs a phase name:

VSE Environments

```
*           MVC                MODPRFX,0(R4)          R4 points to the third input
*                                                       parameter (SUPRA file name).
*                                                       Builds a string of the form
*                                                       ffffUS10.
                LA                R1,MODNAME          The load phase, whose
                LA                R0,MODADDR          8-character name is now
                LOAD              (1),(0)            found at MODNAME, is loaded into
*                                                       virtual storage and its address *
*                                                       is returned in R1; that is,
*                                                       register 1 now contains the *
*                                                       address of the first byte of
*                                                       phase ffffUS10.
```

The following instructions override the default Unload parameters with those specified by the CSECT associated with the file.

```
*           LM                R4,R5,0(R1)            Pick up unrelocated start and
*                                                       end addresses of new SORT
*                                                       statement.
                AR                R4,R1              Relocate ADCONS.
                AR                R5,R1
                MVC              0(42,R4),0(R2)      Move original sort fields.
                MVC              0(9,R5),42(R2)      Move original work parameter.
                L                 R1,SAVE1           Restore original register 1.
                LA                R5,9(,R5)          Bump to end of SORT.
                STM              R4,R5,0(R1)         Store addresses of modified SORT *
*                                                       statement in SORT parameter list.
                B                 RETURN             Branch to RETURN.
```

The following table requires one entry for each file requiring additional parameters in the SORT statement:

TABLE	DS	0F	Header label.
TABENT	DS	0F	To supply appropriate implied * length of entry for searching.
	DC	C'ffff1'	Appropriate number of four-character file names.
	DC	C'ffff2'	
	DC	C'ffffn'	
	.	.	
	.	.	
	DC	C'END.'	Code this only if you code no * file name.
TABEND	EQU	*	End of the above list.

The following area contains the constructed phase name passed as the parameter to the load module:

MODNAME	DS	0D	
MODPRFX	DC	CL4'ffff'	File name moved from table.
MODEND	DC	CL4'US10'	Any four characters you choose as a common ending for the loaded phase names.
*			
*			
SAVE1	DS	F	Area to save original parameter pointer.
	DS	0D	
MODADDR	DS	CLnnn	Where nnn is the size of the largest program loaded.
*			
	END		

The following code illustrates a sample of a phase loaded by the preceding program. You must replace *ffffn* with the name of a file for which you want to add a sort field. You must also code the same file name in the table at label TABLE in the preceding program. You can code any valid sort parameters at label YOURSORT to satisfy your requirements. You can code as many of these phases as you need. This phase must be link edited at +0.

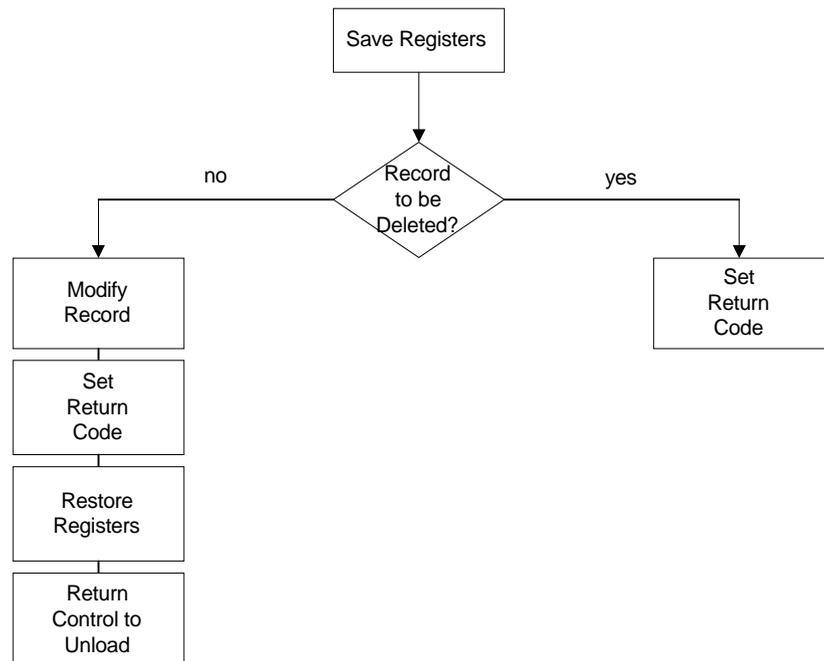
ffffnUS10		CSECT	Use 4-character file name.
	DC	A(SRTBEG)	Required as shown.
	DC	A(SRTEND)	Required as shown.
SRTBEG	DC	CL42' '	Required as shown.
YOURSORT	DC	C',11,5,CH,A'	Additional SORT parameters.
*			The date starts 11 bytes into the record and is 5 bytes long.
*			
SRTEND	DC	CL9' '	Required as shown.
	END		

Using exit point 20

At exit point 20 (USERE20), you can modify or delete a record before sorting it. The figure in “Using exit points” on page 265 shows the point where the Unload function checks to see whether you have included a program for exit point 20. This point is after the function extracts and formats a record from the file and before it passes the record to the SORT program.

If you have included a program, the Unload function loads register 15 with the starting address of your program. At the same time, it loads register 1 with the address of the record you want written to the output file. The Unload function then passes control to your program once for each record it unloads. If it is unloading related files and you coded PRESERVE=YES, the function does not access your exit program and preserves the order of the records on the linkpath.

The following figure shows the steps that your exit program takes. It evaluates each record according to your criteria and either deletes the record or passes it on for further processing.



Processing at Exit Point 20

If you want to delete the record, your program should place a return code of 4 in register 15 (or in 'RETURN-CODE' in COBOL). This returns control to the Unload function and deletes the record. After your program processes the last record, it returns control to the function which then passes control to the SORT program.

If you do not want the record deleted, your program can modify it. After modification, your program should place a return code of 0 in register 15 (or 'RETURN-CODE' in COBOL) and return control to the Unload function. The Unload function then passes the record on to the SORT program.

If you have defined or expanded fields with the *FILL=nn parameter in the element list, you may initialize them in your program at either exit point 20 or 30. For examples of programs you can insert at exit point 20, see ["Sample programs for exit point 20"](#) on page 281.

Using registers

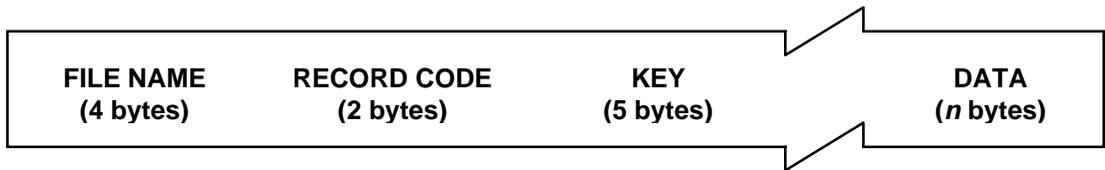
When the Unload function calls your program at exit point 20, these four registers contain the following information. When your exit program returns to the Unload function, it must restore all registers, except 15, to their contents at entry. Register 15 must contain a return code, as shown below.

Register	Function
R1	Address of a fullword containing the address of the record you want sorted.
R13	Address of a standard 72-byte save area. In this area, your program saves the contents of the registers as they were when it gained control. When it returns control, it uses the contents in this area to restore them.
R14	Return address. Your exit returns control to the Unload function at this address.
R15	At entry, address of entry point USERE20. At exit, this register contains one of the following return codes: <ul style="list-style-type: none"> 0 Directs the Unload function to pass the record to SORT. 4 Directs the Unload function not to pass the record to the SORT program.

Sample programs for exit point 20

Since exit points 10 and 20 are frequently used together, this example presents a problem illustrating the use of both exits. It shows example programs in both Assembler and COBOL languages for exit point 20. Also shown are modifications to the programs for exit point 10. These modifications make the exit point 10 programs applicable to the example shown below.

Assume you wish to unload a related file which, when unloaded, has records in the OUTPUT file in the following format:



You want to keep the records in a specific sequence within the key. You want the record codes in the following order: HR, DT, CR, and DB. These are the only record codes in the file. As before, you use the program at exit point 10 to extend the sort parameter so it includes the record code. Since the record codes are not in alphabetical order, this program at exit point 20 modifies the record codes into a collating sequence. Exit point 20 changes HR to 01, DT to 02, CR to 03, and DB to 04. After sorting, exit point 30 restores the record codes to their original values. [“Record code modification by exit programs illustrating the use of exit points 20 and 30”](#) on page 284 illustrates the changes to the record codes. [“Record sequence before and after being sorted with record codes modified at exit points 20 and 30”](#) on page 285 shows the sequence of records in the file before and after the sort.

The following are the modifications to the programs at exit point 10:

The coding is the same as shown in the Assembler program, except that the MYSRT statement is changed to read as follows:

```
MYSRT      DC      C',5,2,CH,A'
```

The following are the modifications to the generalized program:

The coding is the same as shown in the generalized routine, except that the YOURSORT statement is changed to read as follows:

```
YOURSORT      DC      C',5,2,CH,A'
```

You can use the programs at exit point 10 to add the record code to the sort fields already generated by the Unload function. Since the record code is located in the fifth and sixth positions of the unloaded records, you can change the exit point 10 programs as described below. The changes are the same for both OS/390 and VSE.

If you use the Assembler exit program, make the following change:

```
MYSRT        DC      C',5,2,CH,A'
```

If you use the generalized exit program, make the following change:

```
YOURSORT      DC      C',5,2,CH,A'
```

The following is an Assembler program for exit point 20:

```
USERE20      CSECT      Entry point.
              USING     * ,R15      R15 will be the base register.
              STM       R14,R12,12(R13) Save Unload's registers.
              L         R1,0(,R1)   Pick up address of record from
*                                     Unload.
              LA        R2,TABLE    Address of start of conversion
*                                     table.
USERE201     DS         0H          End of table?
              CLC       0(4,R2),HEXFF If invalid record code, delete
              BE        USERE202    record.
*
              CLC       0(,R2),4(R1) Does table match record code in *
*                                     record?
              BE        USERE203    If match, change to internal
*                                     code.
              LA        R2,4(,R2)   Check next code.
              BUSERE201 Try next entry in table.
USERE202     DS         0H          No match, delete record.
              LM        R14,R12,12(R13) Restore Unload's registers.
              LA        R15,4      Set return code to delete record.
              BR        R14        Return.
USERE203     DS         0H          Match, sort record.
*                                     Change record code to internal
              MVC       4(2,R1),2(R2) code.
              LM        R14,R12,12(R13) Restore Unload's registers.
              SR        R15,R15    Set return to sort record.
              BR        R14        Return control to Unload.
TABLE        DC         C'HR01'    Table of record code values.
              DC         C'DT02'
              DC         C'CR03'
              DC         C'DB04'
HEXFF        DC         XL4'FFFFFFFF'
              END
```

The following is a COBOL program for exit point 20:

```

IDENTIFICATION DIVISION.
PROGRAM-ID.  USERE20.
DATA DIVISION.

WORKING-STORAGE SECTION.
    01 TABLE.                                Record code conversion table.
        02 FILLER PIC X(4) VALUE 'HR01'.
        02 FILLER PIC X(4) VALUE 'DT02'.
        02 FILLER PIC X(4) VALUE 'CR03'.
        02 FILLER PIC X(4) VALUE 'DB04'.
    01 TABS REDEFINES TABLE.
        02 SEARCH-TAB OCCURS 4 TIMES         Four entries are to be redefined.
            INDEXED BY TAB-1.
                03 TAB-ARG PIC XX.           Value to represent record code
                03 TAB-PLUG PIC XX.          in sort.

LINKAGE SECTION.
    01 RECORD-LAYOUT.                         Record passed to exit from
        02 FILE-NAME PIC XXXX.               Unload.
        02 RECORD-CODE PIC XX.
        02 KEY PIC X(5).
        02 BALANCE-OF-DATA PIC X(90).       Corresponds to Unload element list.

PROCEDURE DIVISION.
    USING RECORD-LAYOUT.

BEGIN.
    SET TAB-1 TO 1.                           Start at beginning of table.
    SEARCH SEARCH-TAB                          Search table for matching record code.
        AT END GO TO WRONGO                   Go to WRONGO if no match.
        WHEN RECORD-CODE EQUALS              Go to FOUNDEM if a match is found.
            TAB-ARG(TAB-1)
        GO TO FOUNDEM.

WRONGO.
    MOVE 4 TO RETURN-CODE.                    Set return code to delete record.
    GOBACK.                                    Return.

FOUNDEM.
    MOVE TAB-PLUG(TAB-1) TO                  Change record code to value used for sort.
    RECORD-CODE.
    MOVE 0 TO RETURN-CODE.                   Set return code to sort this record.
    GOBACK.                                    Return.

```

Record code modification by exit programs illustrating the use of exit points 20 and 30

Content of a record prior to modification using exit 20

Record Code field	Control-key field	Data fields
HR	A	
DT	A	
CR	A	
DB	A	

Record codes are HR, DT, CR, and DB. (Actual records will not be in sequence, as shown here.) The control-key may be any valid control-key value, e.g., A, B, C, etc. Any of the record codes may appear with any control-key. The control-key value A is used here for illustrative purposes only.

Content of a record after modification using exit 20

Record Code field	Control-key field	Data fields
01	A	
02	A	
03	A	
04	A	

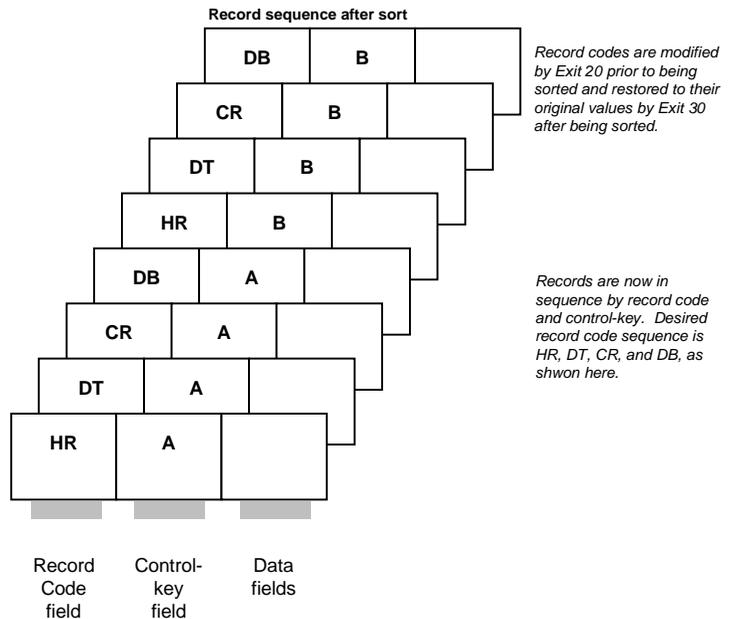
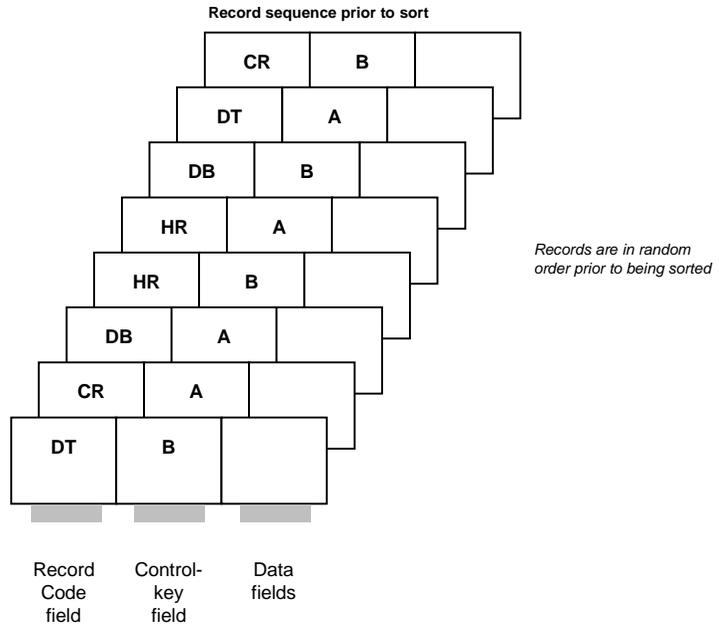
Record codes are changed from HR to 01; DT to 02; CR to 03; and DB to 04 for sorting. These are the values to be used by the SORT program to sort the records into ascending collating sequence by record code and control-key. Original record code values will be restored after all records in the file have been sorted.

Content of a record after modification using exit 30

Record Code field	Control-key field	Data fields
HR	A	
DT	A	
CR	A	
DB	A	

Record codes have been restored to their original values. Record codes have been restored from 01 to HR; 02 to DT; 03 to CR; and 04 to DB.

Record sequence before and after being sorted with record codes modified at exit points 20 and 30



Using exit point 30

At exit point 30 (USERE30), you can delete, modify, or add records in the output file. The figure in “Using exit points” on page 265 shows the point where the Unload function checks to see if you have included a program for exit point 30. It is after the sort phase, but before the function writes the unloaded records to the output file.

If you have included a program, the Unload function loads register 15 with the starting address of your program. At the same time, it loads register 1 with the address of the record you want written to the output file. The Unload function passes control to your program once for every record. After your program has processed the last record, it returns control to the Unload function.

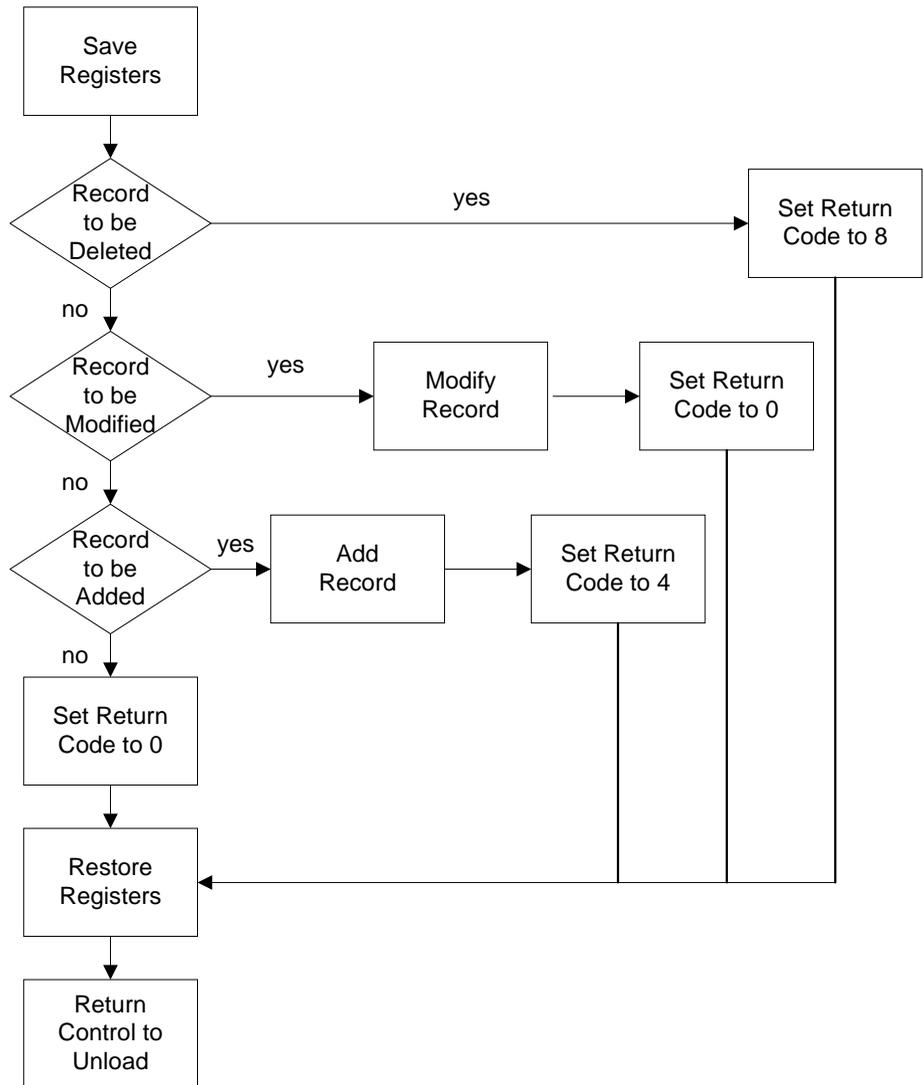
The figure at the end of this section shows the steps that your program must take. At the beginning of your program, it should save the contents of the Unload registers by moving them to a 72-byte save area. Its beginning address is in register 13.

Next, your program evaluates each record according to criteria you define. Then it adds, deletes, or modifies the record and has the Unload function write it to the output file. Your program tells the Unload function what to do with each record by placing a return code in register 15 (or in 'RETURN-CODE' in COBOL). After it sets the return code, it returns control to the function, which either deletes the record or writes it to the output file.

If you are adding records, you must have previously sorted them according to the same criteria, put them in the same element list format, and put them in the same order as the rest of the records. When the Unload function adds a record, it inserts it in front of the record it just passed to your exit program. After your program adds the record, the Unload function automatically makes the original record it just passed to your program available again.

If you have defined or expanded fields with the *FILL=*nn* parameter in the element list, you may initialize them with your program at either exit point 20 or 30.

The Unload function calls this exit whether you have coded PRESERVE=YES or PRESERVE=NO.

**Processing at Exit Point 30**

Using registers

When the Unload function calls your program at exit point 30, these four registers contain the following information. When your exit program returns to the Unload function, it must restore all registers, except 15, to their contents at entry. Register 15 must contain a return code, as shown below.

Register	Function
R1	Address of a fullword containing the address of the record you want written.
R13	Address of a standard 72-byte save area. In this area, your exit program saves the contents of the registers at entry and restores them from this area at exit.
R14	Return address. Your exit program returns control to the Unload function at this address.
R15	At entry, address of entry point USERE20. At exit, this register contains one of the following return codes: 0 Directs the Unload function to write the record to the output file. 4 Directs the Unload function to add the record and write it to the output file. 8 Directs the Unload function to delete the record.

When the exit program returns a code of 4, it directs the Unload function to add a record. When the Unload function adds records, it must insert them in front of the record it is processing. When the Unload function accesses your program the next time, it automatically makes the original record available again.

The example below illustrates this process. In this case, you want to add Record 8. It must go before Record 10 and after Record 6. The second time the Unload function accesses your program, it passes Record 10 to your program. When your program returns control, it directs the Unload function to add Record 8. After the Unload function writes Record 8 to the output file, it again passes Record 10 to your program for deletion or modification. This time, your program directs Unload to write it to the output file.

Access	Passed	Process	Return code
1 st	Record 6	WRITE RECORD 8	0
2 nd	Record 10	ADD RECORD 8	4
3 rd	Record 10	WRITE RECORD 10	0
4 th	Record 15	WRITE RECORD 15	0

Sample programs for exit point 30

These Assembler and COBOL programs for exit point 30 solve the problem stated in “[Sample programs for exit point 10](#)” on page 271 for exit point 10. This exit program restores the record codes to their original values.

“[Record sequence before and after being sorted with record codes modified at exit points 20 and 30](#)” on page 285 shows the record sequence before and after being sorted with record codes modified by programs for exit points 20 and 30.

The following is an Assembler program:

```

USERE30  CSECT      Entry point.
        USING  *,R15  R15 will be the base register.
        STM R14,R12,12(R13)  Save Unload's registers.
        L   R1,0(,R1)  Pick up address of record.
        LA  R2,TABLE  Address of start of table.
USERE301 DS  0H
        CLC 0(4,R2),HEXFF  End of table?
        BE  USERE302  If invalid code, delete record.
        CLC 2(4,R2),4(R1)  Does table match code in record?
        BE  USERE303  If match, change back to record code.
        LA  R2,4(,R2)  Check next code.
        B   USERE301  Try next entry in table.
USERE302 DS  0H  No match - delete record.
        LM  R14,R12,12(R13)  Restore Unload's registers.
        LA  R15,8  Set return code to delete record.
        BR  R14 Return.
USERE303 DS  0H  Match, unload record.
        MVC 4(2,R1),0(R2)  Change internal code back to record code.
        LM  R14,R12,12(R13)  Restore Unload's registers.
        SR  R15,R15 Set return code to unload record.
        BR  R14 Return.
TABLE DC  C'HR01' Table of record code values.
        DC  C'DT02'
        DC  C'CR03'
        DC  C'DB04'
HEXFF DC  XL4'FFFFFFFF'
        END

```

The following is a COBOL program for exit point 30:

```

IDENTIFICATION DIVISION.
PROGRAM-ID.  USERE30.
DATA DIVISION.

WORKING-STORAGE SECTION.
    01 TABLE.                                Record code conversion table.
        02 FILLER PIC X(4) VALUE 'HR01'.
        02 FILLER PIC X(4) VALUE 'DT02'.
        02 FILLER PIC X(4) VALUE 'CR03'.
        02 FILLER PIC X(4) VALUE 'DB04'.
    01 TABS REDEFINES TABLE.
        02 SEARCH-TAB OCCURS 4 TIMES          Four entries are to be redefined.
        INDEXED BY TAB-1.
            03 TAB-ARG PIC XX.
            03 TAB-PLUG PIC XX.                Value to represent record code in sort.

LINKAGE SECTION.
    01 RECORD-LAYOUT.                          Record passed to exit from Unload.
        02 FILE-NAME PIC XXXX.
        02 RECORD-CODE PIC XX.
        02 KEY PIC X(5).
        02 BALANCE-OF-DATA PIC X(90).         Corresponds to Unload element list.

PROCEDURE DIVISION
    USING RECORD-LAYOUT.

BEGIN.
    SET TAB-1 TO 1.                             Start at beginning of table.
    SEARCH SEARCH-TAB                            Search table for matching record code.
        AT END GO TO ALSO-WRONGO                Go to ALSO-WRONGO if no match.
        WHEN RECORD-CODE EQUALS                Go to FOUNDEM-AGAIN if a match is
            TAB-PLUG(TAB-1)                    found.
        GO TO FOUNDEM-AGAIN.

ALSO-WRONGO.
    MOVE 8 TO RETURN-CODE.                       Set return code to delete record.
    GOBACK.                                       Return.

FOUNDEM-AGAIN.
    MOVE TAB-ARG(TAB-1) TO                       Match found in table. Change
        RECORD CODE.                             value used for sort back to record code.
    MOVE 0 TO RETURN-CODE.                       Set return code to unload this record.
    GOBACK.                                       Return.

```

Loading PDM files

The Version 2 Load Function (CSULOADR) loads SUPRA Directory or PDM files from the output of the Version 2 Unload function. The Load function also automatically updates the linkpath fields with correct data. You can load the files in any format: SUPRA native, SUPRA converted, or Series 80.



Use the Version 2 Load function only with output files created by the Version 2 Unload function. The Version 2 Load function does not operate with files created by the Version 1 Unload utility.

The Version 2 Unload function unloads only Directory and PDM files; it does not unload index files. Therefore, you cannot load index files with the Version 2 Load function.



If your PDM files have secondary keys, you must depopulate them before you unload them and repopulate them after you load them. For more information, see the introduction to this chapter.

Before you load a PDM VSAM file, allocate it with the IDCAMS utility, but do not format it.

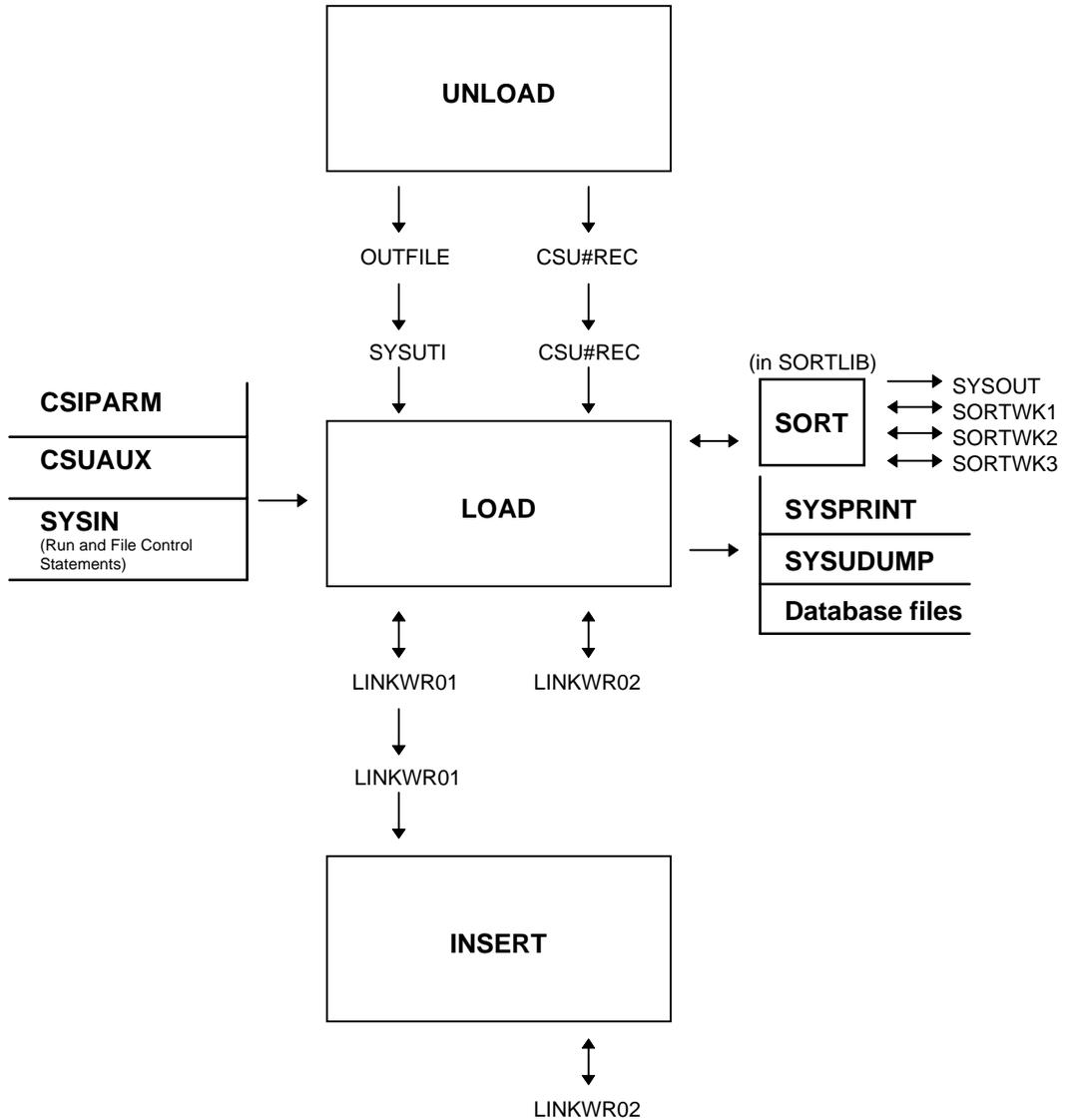
To use the Version 2 Load function, you do not code UCL. Instead, you code the following input:

- ◆ File definitions
- ◆ Run control statements
- ◆ File control statements

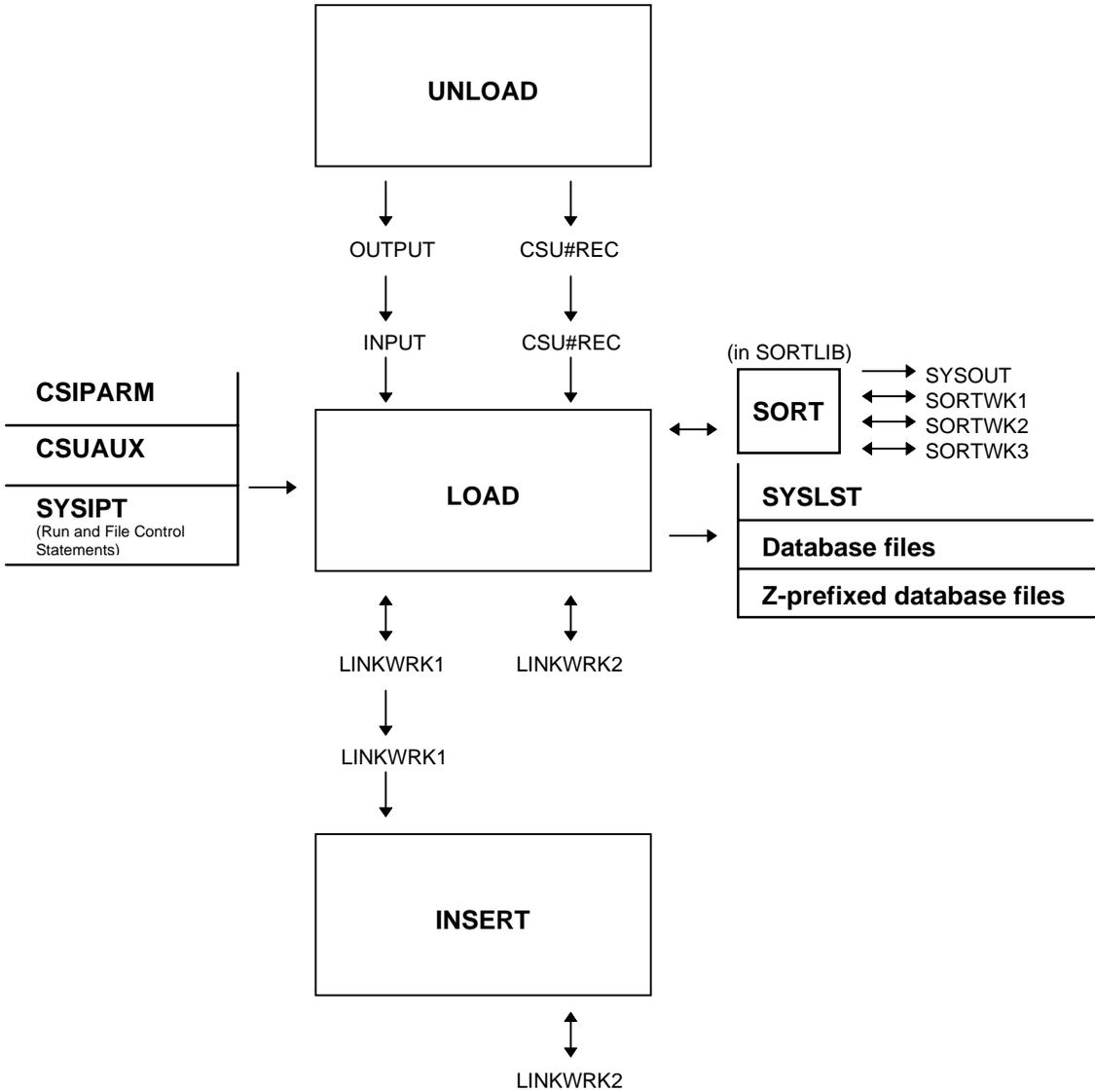
Defining files

To execute the Load function, you must define the files listed in “[File definitions for the Load function](#)” on page 295 in your JCL and execute the Unload program named CSUNLOAD. In OS/390, you can use the cataloged procedure TISUTLOD. If you want to change any of the symbolic parameters in TISUTLOD, refer to the [SUPRA PDM and Directory Administration Guide](#), P26-2250. If you do not want to use the cataloged procedure, you can follow the same procedure as in VSE. The following figures show the files that you must define in OS/390 and VSE.

Files you define in OS/390 JCL to load



Files you define in VSE JCL to load



File definitions for the Load function

To execute the Load function, you must define the files listed below in your JCL and execute CSUNLOAD.

DD or file name	Description	Considerations
OS/390 CSI#WKnn	Identifies the sort work files.	VSE For VSE, see the SORTWKnn definition. If not enough virtual storage is allocated to sort in place, identify the needed sort work files (CSI#WK01, CSI#WK02 and CSI#WK03). Format and space allocation are identical to standard SORTWKnn statements as defined in the appropriate sort manual.
CSIPARM	Identifies the CSIPARM file, which contains control information needed by the PDM.	Code an open mode of NONE in the environment description for all files you want the Unload and Load functions to read. Also, code the task log option as NONE in the environment description or the Utility will abend.
CSU#REC	Holds the number of records the Unload function unloaded.	When you supply the CSU#REC file to the Load function, you must ensure that the file comes from the same run of the Unload utility as the INPUT file. For more information, see “Defining the CSU#REC file” on page 238.
CSUAUX	Holds the auxiliary information for the files that are not in native format.	See “Defining the CSUAUX file” on page 238.
OS/390 fffffff VSE fffffff and Zffffff	Names the files you want loaded.	You may code up to 57 primary files and 57 related files. The file name must be coded in the SUPRA Directory for the schema you are loading. For VSE, you must code primary and related files on two separate DLBL statements. Code each file twice: once for direct access with the file name (ffffff) on the DLBL statement and the second time for sequential access with a Z before the file name. Truncate to seven characters (Zffffff).

DD or file name	Description	Considerations
VSE INPUT	Holds the data for all files you want loaded.	This must be the OUTFILE output file of the Unload execution. See “ Defining the OUTFILE ” on page 241.
OS/390 LINKWK01 VSE LNKWRK1	Indicates the first See linkage work file.	See “ Coding the LINKWK01/LNKWRK1 file ” on page 297.
OS/390 LINKWK02 VSE LNKWRK2	Indicates the second linkage work file.	See “ Coding the LINKWK02/LNKWRK2 file ” on page 300.
SORTLIB	Indicates the library containing the standard sort program.	
VSE SORTWK n	Identifies work files for sorting.	OS/390 For OS/390, see the CSI#WK nn definition. If insufficient virtual storage is allocated to sort in place, identify the required standard sort work files (SORTWK1, SORTWK2, SORTWK3) as defined in the appropriate sort manual.
OS/390 SYSIN VSE SYSIPT	Holds the run control and file control statements.	See “ Coding run control statements for the Load function ” on page 302 and “ Coding the Element List statement ” on page 318.
VSE SYSLST OS/390 SYSPRINT	Identifies the output file for the printed listing of all control statements, diagnostic messages, etc.	
SYSOUT	Identifies the file the standard sort program uses.	
OS/390 SYSUDUMP	Indicates that you want a storage dump taken and written to this file if an abend occurs.	Optional.
OS/390 SYSUT1	Holds the data for all files you want loaded.	This must be the OUTFILE output file of the Unload execution. See “ Defining the OUTFILE ” on page 241.

Coding the LINKWK01/LNKWRK1 file

The first linkage work file is LINKWK01 in OS/390 and LNKWRK1 in VSE. You need the file in these situations:

- ◆ For loading related files
- ◆ For loading primary files that are connected by linkpaths to related files you are also loading.

You use the file to receive linkage data the Load function generates while loading related files. The Load function then inserts the linkage data into the primary files with which they are associated.

If you are loading primary files but not the related files with which they are associated, you can omit this linkage work file because the Load function does not generate any linkage information.

If you are loading related files, you can insert the linkpath data into the associated primary files by executing the Insert Linkpath function. If you do so, you must use a permanent file or pass this linkage work file to the Insert Linkpath function in a later step. The latter possibility is illustrated in [“Files you define in OS/390 JCL to load”](#) on page 293 and [“Files you define in VSE JCL to load”](#) on page 294.

To allocate the linkage work file, you need to know the number of records in the file. You can determine the number in two steps:

1. Execute the File Statistics function to obtain chain statistics for every related file you will load.
2. Add the number of chains in the linkpaths for all the related files you are loading.

When you code your JCL for this file, you need to calculate the logical record length (LRECL). You do that by adding 20 bytes to the value in the MAXKEY parameter in the run control statements. (You may have coded it or allowed it to default.) The purpose of the 20 bytes is shown in the following table. You must calculate and code the LRECL parameter exactly because the Load function calculates this value itself under some conditions and its value must match yours.

Position (bytes)	Contents
1–4	Primary file name
5–8	Home location relative record number (RQLOC) of primary record
9–12	Linkpath name (LKxx)
13–16	Relative record number (RRN) of first record in the related chain
17–20	RRN of last record in the related chain
21– <i>n</i>	Primary file control key. Maximum key length is 256 bytes.

You code the rest of your JCL differently depending on whether you are loading primary or related files:

- ◆ For related files, use DISP=(NEW,KEEP,DELETE)
- ◆ For primary files, use DISP=(OLD,DELETE,KEEP)
- ◆ For related and primary files together, use DISP=(NEW,DELETE,KEEP)

When you load related files, you should code the BLKSIZE parameter for the file. You should make it at least as large as the LRECL parameter you coded. If you make the BLKSIZE parameter larger than the LRECL parameter, it must be a multiple of the LRECL parameter. The Load function uses your value to calculate the number it actually uses. It uses a number that is less than or equal to the BLKSIZE parameter you code. For example, if the record length is 1024 bytes and you code a block size of 10,500 bytes, the Load function uses 10,240.

If you do not code the BLKSIZE parameter, it defaults to 10K for tape devices or the maximum size for the DASD used. In that case, the Load function sets the record format to fixed blocked.

When you load only primary files with linkage data from loading their associated related files, you must code the LRECL and BLKSIZE parameters exactly as they were in the job that loaded these related files.

Coding the LINKWK02/LNKWRK2 file

The second linkage work file is LINKWK02 in OS/390, LNKWRK2 in VSE. The file has different uses depending on whether you are loading related or primary files:

- ◆ For related files, it holds secondary linkage data
- ◆ For primary files, it holds out-of-block synonym records

In both cases, the record format is preset to fixed blocked. The Load function internally calculates the record length (LRECL) at run time by adding 22 bytes to the largest related key length.

VSE

In VSE, we recommend that you code the record length yourself rather than letting it default. Code it in the RECSIZE parameter of the run control statements. To determine the record length, you must make two calculations—one for related and one for primary files—and choose the higher of the two values.

For related files, add at least 22 bytes to the length of the largest key. (See the record formats for the purpose of the 22 bytes below.) Select the largest key from the access and secondary linkpath keys of the related files.

For primary files, use the length of the largest record in the primary files.

If you are loading only primary files, determine the number of records in two steps:

1. Execute the File Statistics function to obtain the number of out-of-block synonyms for each primary file you want to load.
2. If you are not loading the files with new block sizes, code the largest number for any of these files.

If you are decreasing or increasing the block size, you can expect the number of out-of-block synonyms (and therefore the number of records required) to increase or decrease slightly in reverse proportion to the change in block size. That is, if you increase the size of the block, you have fewer out-of-block synonyms and need fewer records. When you change the block size, you cannot arrive at an exact number; you have to estimate the number.

If you are loading only related files or related and primary files, you determine the number of records in three steps:

1. Execute the File Statistics function to obtain the linkpath statistics for all the related files you want to load.
2. Add the number of records containing each secondary linkpath.
3. Use the highest number.

You must code a multiple of the LRECL parameter for the BLKSIZE parameter. In VSE, we recommend you code the block size in the BLKSIZE parameter rather than let it default.

If you use a tape device for this file, use DISP=(NEW,PASS) to avoid unloading and reloading the tape between each primary file.

If you want to estimate the amount of sort work space, consider the format of the records for the first sort:

Position (bytes)	Contents
1–4	Related file name
5–12	Secondary Linkpath name
13–16	Relative position in file (RRN)
17–18	Record code
19–20	Displacement of control key into related file
21–22	Length of primary control key
23– <i>n</i>	Primary record control key

The following is the format of the records for the second sort:

Position (bytes)	Contents
1–4	Related file name
5–8	Relative position in file (RRN)
9–12	Linkpath (Previous)
13–16	Linkpath (Next)
17–18	Displacement of Linkpath into related record

Coding run control statements for the Load function

With run control statements, you can control the execution of the Load function. You may code some of these statements with several parameters and others with only one. You may need to code some statements with more than one record, but you cannot code two different statements on the same record, except in VSE where you can code SORTCORE, SORTNAME, and WORK statements on one record. You must begin each run control statement in position 1.

You must code the RELATED: file list before the PRIMARY: file list, and list the files in ascending order. You are free to code all the other run control statements in any order.

All the run control statements are shown in the following table with a brief description.

Statement	Description	Section
SCHEMA	Indicates the name of the schema you want used to load the database.	“Coding the SCHEMA statement” on page 304
RELATED:	Names the related files you want loaded.	“Coding the RELATED: statement” on page 305
V-E:	Alternate form of RELATED: statement.	“Coding the V-E: statement” on page 306
PRIMARY:	Names the primary files you want loaded.	“Coding the PRIMARY: statement” on page 306
S-E:	Alternate form of PRIMARY: statement.	“Coding the S-E: statement” on page 307
MAXKEY	Indicates the length of the longest primary file control key in the related files you want loaded.	“Coding the MAXKEY statement” on page 307
VSE LINKWK nn	Defines the characteristics of the LNKWRK1 and LNKWRK2 files.	“Coding the LINKWK nn statements (VSE only)” on page 308
VSE RECFORM	Defines the characteristics of the INPUT file.	“Coding the RECFORM statement (VSE only)” on page 311
SORTCORE	Indicates the amount of virtual storage the function can use for the SORT program.	“Coding the SORTCORE statement” on page 314
SORTNAME	Names the sort program if it is not the standard program.	“Coding the SORTNAME statement” on page 315
VSE WORK	Indicates the number of tape devices or disk extents available for intermediate sort storage.	“Coding the WORK statement (VSE only)” on page 316

Coding the SCHEMA statement

With the SCHEMA statement, you name the schema you want to use for loading your database files. If you are loading Directory files, name the bootstrap schema.

$$\text{SCHEMA} = \left\{ \begin{array}{l} \textit{schemaname} \\ \textit{bootschema} \end{array} \right\}$$

$$\text{SCHEMA} = \left\{ \begin{array}{l} \textit{schemaname} \\ \textit{bootschema} \end{array} \right\}$$

Description *Required.* Names the schema you want the function to use when loading your database files.

Format 1–8 alphanumeric characters

Consideration To load your PDM files, the CSIPARM file must have your bootstrap schema and environment description in the DIRECTORY parameter and your schema and environment description in the REALM parameter. When you code this run control statement, you should code your schema. To coordinate coding your run control statements with your CSIPARM file, see [“Coding CSIPARM file and run control statements for PDM files”](#) on page 230.

Coding the RELATED: statement

With the RELATED: statement, you can name the related files you want loaded. If you have no related files to load, you can omit this statement. If you code it, put it before the PRIMARY: statement.

You may use the Series 80 V-E: statement in place of the RELATED: statement. However, use one or the other, not both.

RELATED:rrrr1[rrrr2...rrrrn]END.

or

V-E:vvvv1[vvvv2...vvvvn]END.

RELATED:rrrr1[rrrr2...rrrrn]END.

V-E:vvvv1[vvvv2...vvvvn]END.

Description *Optional.* Identifies the related files you want loaded.

Format

- ◆ Code RELATED: in positions 1–8 of the first record only, or code V-E: in positions 1–4.
- ◆ You may use up to three records for this statement. If you use several records, begin the file names in position 1 of the second and third records.
- ◆ Use all 80 positions unless you are coding the last statement.

Considerations

- ◆ Arrange related file names in ascending order, and define them with JCL statements.
- ◆ Load your PDM files and Directory files in separate jobs.
- ◆ You can name up to 57 files by using several records.
- ◆ You must code END. immediately after the last file name to indicate the end of the file list.

Coding the V-E: statement

The V-E: statement is supported for compatibility with existing Series 80 and TIS 1.x job streams. It serves the same purpose as the RELATED: statement. For coding information, see “Coding the RELATED: statement” on page 305.

Coding the PRIMARY: statement

With the PRIMARY: statement, you can name the primary files you want loaded. You can omit this statement if you have no primary files to load. If you code this statement, put it after the RELATED: statement.

You may use the Series 80 S-E: statement in place of the PRIMARY: statement. However, use one or the other, not both.

PRIMARY:pppp1[pppp2...ppppn]END.

or

S-E:mmmm1[mmmm2...mmmmn]END.

PRIMARY:pppp1[pppp2...ppppn]END.

S-E:mmmm1[mmmm2...mmmmn]END.

Description *Optional.* Identifies the primary files you want loaded.

Format

- ◆ Code PRIMARY: in positions 1–8 of the first record only, or code S-E: in positions 1–4.
- ◆ You can use up to three records for this statement. If you use more than one, begin the file names in position 1 of the second and third records.
- ◆ Fill all 80 positions unless you are coding the last statement.

Considerations

- ◆ Arrange the primary file names in alphabetical order, and define them with JCL statements.
- ◆ Load your PDM files and Directory files in separate jobs.
- ◆ You can name up to 57 primary files by using several records.
- ◆ You must code END. immediately after the last file name to indicate the end of the file list.

Coding the S-E: statement

The S-E: statement is supported for compatibility with existing Series 80 and TIS 1.x job streams. It serves the same purpose as the PRIMARY: statement. For information on coding it, see “Coding the PRIMARY: statement” on page 306.

Coding the MAXKEY statement

With the MAXKEY statement, you can indicate the length of the longest primary file control key in the primary and related files you want loaded. You need to code this statement for various internal calculations and functions.

$$\text{MAXKEY} = \left\{ \begin{array}{c} 256 \\ n \end{array} \right\}$$

$$\text{MAXKEY} = \left\{ \begin{array}{c} 256 \\ n \end{array} \right\}$$

- | | |
|----------------------|---|
| Description | <i>Optional.</i> Indicates the length of the longest primary file control key in the related files you want loaded. |
| Default | 256 |
| Options | 1–256 |
| Consideration | For efficient use of time and space by the sort program, do not use the default if the actual length of the longest control key is much shorter than 256 bytes. |

Coding the LINKWKnn statements (VSE only)

With the LINKWKnn statements, you can define the characteristics of the linkage work files: LNKWRK1 and LNKWRK2. You can code the parameters on more than one line.

If you use this LNKWRK1 file later in the Insert Linkpath function, you must code the same LINKWK01 parameters for the Insert Linkpath function that you code here. For more information on coding the statement for that function, see “Coding the LINKWKnn statements (VSE only)” on page 331.

$$\text{LINKWKnn: } \left[\text{DEVICE} = \left\{ \begin{array}{l} \text{DISK} \\ \text{TAPE} \left[\text{,FILABL} = \left\{ \begin{array}{l} \text{NO} \\ \text{STD} \end{array} \right\} \right] \end{array} \right\} \right]$$

$$\left[\text{,BLKSIZE} = \left\{ \begin{array}{l} 1000 \\ n \end{array} \right\} \right]$$

$$\left[\text{,RECSIZE} = \left\{ \begin{array}{l} 1000 \\ n \end{array} \right\} \right] \left[\text{,DEVADDR} = \left\{ \begin{array}{l} \text{SYS030} \\ \text{SYSnnn} \end{array} \right\} \right]$$

LNKWKnn

- | | |
|----------------------|--|
| Restriction | <i>Required</i> if you code DEVICE, FILABL, BLKSIZE, RECSIZE, or DEVADDR parameters. |
| Description | <i>Required.</i> Identifies the LINKWKnn statement. |
| Options | LINKWK01: The function defines the LNKWRK1 file.
LINKWK02: The function defines the LNKWRK2 file. |
| Format | Code in positions 1–9. |
| Consideration | You can omit the LINKWK01 statement if you are loading primary files without LNKWRK1 information. |

DEVICE = $\left\{ \begin{array}{l} \text{DISK} \\ \text{TAPE} \end{array} \right\}$

Description *Optional.* Use this parameter to indicate the file's device type.

Options

DISK	Disk device
TAPE	Magnetic tape unit

,FILABL = $\left\{ \begin{array}{l} \text{NO} \\ \text{STD} \end{array} \right\}$

Restriction This parameter is valid only when DEVICE=TAPE.

Description *Optional.* Indicates whether the tape contains file labels.

Default NO

Options

NO	Does not contain labels
STD	Contains standard labels

,BLKSIZE = $\left\{ \begin{array}{l} 1000 \\ n \end{array} \right\}$

Restriction *Required* if the RECSIZE is greater than 1000.

Description Indicates the file's block size in bytes. If the RECSIZE is less than or equal to 1000, use this parameter to achieve a higher blocking factor.

Default 1000

Format Use numeric characters.

Consideration To determine the block size, see "[Coding the LINKWK01/LNKWRK1 file](#)" on page 297 and "[Coding the LINKWK02/LNKWRK2 file](#)" on page 300.

,RECSIZE = $\left\{ \begin{array}{l} \mathbf{1000} \\ n \end{array} \right\}$

Description *Required.* Indicates the file's record size in bytes.

Default 1000

Format Use numeric characters.

Consideration To determine the record size, see “Coding the LINKWK01/LNKWRK1 file” on page 297 and “Coding the LINKWK02/LNKWRK2 file” on page 300.

,DEVADDR = $\left\{ \begin{array}{l} \mathbf{SYS030} \\ \mathbf{SYSnnn} \end{array} \right\}$

Description *Optional.* Identifies the device address (SYS number symbolic unit) associated with the file.

Default SYS030

Format *nnn* Must be 3 digits.

Coding the RECFORM statement (VSE only)

With the RECFORM statement, you can define the characteristics of the INPUT file. All parameters are optional since defaults are supplied. If you code any parameters, you must separate them from any other statements. You can continue a parameter on the next line. When you do so, do not code a continuation character.



The parameters must match any parameters you coded on the RECFORM statement in the Unload function. See “[Coding the RECFORM statement \(VSE only\)](#)” on page 249.

$$\left[\text{RECFORM} = \left\{ \begin{array}{l} \text{FIXBLK} \\ \text{FIXUNB} \end{array} \right\} \right] \left[\text{DEVICE} = \left\{ \begin{array}{l} \text{DISK} \\ \text{TAPE} \left[\text{FILABL} = \left\{ \begin{array}{l} \text{NO} \\ \text{STD} \end{array} \right\} \right] \end{array} \right\} \right] \right]$$

$$\left[\text{BLKSIZE} = \left\{ \begin{array}{l} 1000 \\ n \end{array} \right\} \right]$$

$$\left[\text{RECSIZE} = \left\{ \begin{array}{l} 100 \\ n \end{array} \right\} \right] \left[\text{DEVADDR} = \left\{ \begin{array}{l} \text{SYS030} \\ \text{SYSnnn} \end{array} \right\} \right]$$

$$\text{RECFORM} = \left\{ \begin{array}{l} \text{FIXBLK} \\ \text{FIXUNB} \end{array} \right\}$$

Description *Optional.* Indicates the record format of the file.

Default FIXBLK

Options FIXBLK Fixed-length, blocked records

 FIXUNB Fixed-length, unblocked records

,DEVICE = { **DISK** }
 { **TAPE** }

Description *Optional.* Indicates the device type of the file.

Options DISK Disk device
 TAPE Magnetic tape unit

,FILABL = { **NO** }
 { **STD** }

Restriction This statement is valid only when DEVICE=TAPE.

Description *Optional.* Indicates whether the tape contains file labels.

Default NO

Options NO Does not contain labels
 STD Contains standard labels

,BLKSIZE = { **1000** }
 { *n* }

Description *Optional.* Indicates the file's block size in bytes.

Default 1000

Format Use numeric characters.

Consideration You must code a multiple of the record size (RECSIZE). For example, if the record size is 100 bytes, the block size can be 200, 300, 1000, and so on.

$$,RECSIZE = \left\{ \frac{100}{n} \right\}$$

Description *Optional.* Indicates the file's record size in bytes.

Default 100

Format Use numeric characters.

Consideration To determine the value of this parameter, add the following amounts:

- S The sum of the lengths of the data elements you want loaded plus the length of the control key. Calculate this for each file you load and use the largest value.
- +4 The length of the file name. Always add this value.
- +X where "X" is:
 - +2 The length of the record code. Add this if you are loading at least one related file with coded records and are not loading any primary files.
 - +4 The RQLOC. Add this if loading primary files, regardless of the above.
 - +0 Neither of the above.

$$,DEVADDR = \left\{ \begin{array}{l} \text{SYS030} \\ \text{SYS}nnn \end{array} \right\}$$

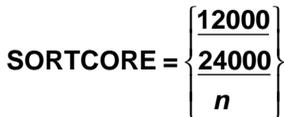
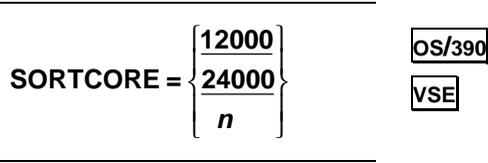
Description *Optional.* Identifies the device address (SYS number symbolic unit) associated with the file.

Default SYS030

Format *nnn* Must be 3 digits

Coding the SORTCORE statement

With the SORTCORE statement, you can indicate the amount of virtual storage the SORT program can use.



Description *Optional.* Indicates the number of bytes of virtual storage the SORT program can use.

Default OS/390 12000 bytes

VSE 24000

Options 12000–999999999

Considerations

- ◆ If you specify less than 12000, the SORT program overrides that figure with 12000 bytes and tries to execute.
- ◆ If you code a value larger than the amount of space available in the address space, the operating system abnormally terminates the Load function.
- ◆ VSE In VSE, you can code the SORTCORE, SORTNAME, and WORK statements on the same record. When you do so, separate the statements with commas.

Coding the SORTNAME statement

With the SORTNAME statement, you can name the sort program you want the Load function to use.

SORTNAME =	IERRCO00	OS/390
	SORT	VSE
	<i>programe</i>	

SORTNAME = { IERRCO00
SORT
programe }

Description *Optional.* Identifies the sort program.

Default OS/390 IERRCO00

 VSE SORT

Format 1–8 alphanumeric characters

Consideration VSE In VSE, you can code the SORTCORE, SORTNAME, and WORK statements on the same record. When you do so, separate the statements with commas.

Coding the WORK statement (VSE only)

With the WORK statement, you can code the number of tape devices or disk extents available for intermediate storage while sorting.

$$\text{WORK} = \left\{ \begin{array}{c} 4 \\ n \end{array} \right\}$$

$$\text{WORK} = \left\{ \begin{array}{c} 4 \\ n \end{array} \right\}$$

Description *Optional.* Indicates the number of tape devices or disk extents available for intermediate storage while sorting.

Default 4

Options 1–9

Consideration You can code the WORK, SORTNAME, and SORTCORE statements on the same record. If you do so, separate the statements with commas.

Coding the file control statements for the Load function

With file control statements, you can control the order of the data elements you are loading. You may code several parameters in some of these statements, while in others you may code only one. Some statements may require more than one record, but you cannot code two different statements on the same record.

You must code the statements for the files in the same order as in the RELATED: and PRIMARY: run control statements. You must code the related file information before the primary file information. When you code the information, you may code a LINKPATH statement for each related file if you like. Then you must code an Element List statement for each related and primary file, in that order.

The following table briefly describes each statement.

Statements	Description	Section
Element List	Indicates the data elements you want loaded for a particular file.	“Coding the Element List statement” on page 318
LINKPATH	Indicates the linkpath you want used as the access linkpath for loading a related file.	“Coding the LINKPATH statement” on page 321

Coding the Element List statement

With the Element List statement, you can indicate the data elements you want loaded from the file (*ffff*) you identified. The figure at the end of this section illustrates the format of the records as they appear on the SYSUT1 file (OS/390) or INPUT file (VSE).

ffff { *element*, [*element*₂...*element*_n]} } **END.**
 { **ALL.** }

ffff

Description *Required.* Identifies the file containing the elements you want loaded.

Format 4-character file name coded in the schema.

Consideration Place the file name in positions 1–4 of each Element List statement.

{ *element*, [*element*₂...*element*_n]} } **END.**
 { **ALL.** }

Description *Required.* Identifies the individual element(s) or all (ALL.) elements you want loaded.

Format

- ◆ You can code from 1 to 100 data elements in a maximum of 12 records. See the last consideration on the following page.
- ◆ If you use ALL., code it in positions 5–8.
- ◆ Code the element list in positions 5–76; anything in positions 77–80 is ignored.

Considerations

- ◆ For related files, you must code the Element List statement immediately after the LINKPATH statement to which it applies.
- ◆ If you coded ALL. in the Unload function, you must also code it in the Load function.
- ◆ All element names you code must have been previously defined for the file in the schema on the SUPRA Directory.
- ◆ You do not need to code all elements of the file in the same order as they appear in the SUPRA Directory. However, you must code the elements in the same position in the Unload and Load functions.
- ◆ To understand how to code your element list to add, delete, expand, or contract elements in your files, see Considerations 6 through 9 of the Unload function's Element List statement in “[Coding the Element List statement](#)” on page 257.
- ◆ If you are loading related files without record codes, you must code the control key associated with the access linkpath first in the Element List statement, regardless of its physical position in the record in the database file. (This does not apply if you are using ALL.)
- ◆ If you are loading coded records, you must put the record code data element immediately before the control key in the Element List statement. (This does not apply if you are using ALL.)
- ◆ When you load related files with an explicit element list, you must code the control keys of all secondary linkpaths for the file as defined in the Directory. When you code the control keys, you help ensure the integrity of the database.
- ◆ When you are loading primary files, you must code *ppppCTRL* first unless you are using ALL.
- ◆ You must code END. immediately after ALL. or the last element to indicate the termination of the element list.
- ◆ Note that if the number of data elements total more than 100, the V2LOADR will get a U1000 error if the 100 plus fields or ALL. are specified. To avoid this situation, have one or more parent Physical Fields defined that encompass the whole record in less than 101 elements that can be used to do the V2LOAD. This error can occur even if you specify ALL. when the elements that make up the record are greater than 100.

ELEM-LIST = Element list you supply

Primary Record Format	▶	ffff PRIMARY FILE NAME	RQLOC VALUE	CONTROL- KEY	All remaining data elements and LINKPATHS as coded on the ELEM-LIST control card(s) (CONTROL-KEY is excluded).

Standard Related Record Format	▶	ffff RELATED FILE NAME	ACCESS-KEY		All remaining data elements as coded on the ELEM-LIST control card(s) (ACCESS-KEY is excluded).

Coded Related Record Format	▶	ffff RELATED FILE NAME	RECORD CODE	ACCESS- KEY	All remaining data elements as coded on the ELEM-LIST control card(s) (ACCESS-KEY is excluded).

ELEM-LIST = ALL. END.

Primary Record Format with Linkpaths in File	▶	ffff PRIMARY FILE NAME	RQLOC VALUE	CONTROL- KEY	All LINKPATHS in the order in which they appear in the internal record	All data elements excluding the root field in the order in which they appear in the internal record control card(s) (CONTROL-KEY excluded).

Primary Record Format without Linkpaths	▶	ffff PRIMARY FILE NAME	RQLOC VALUE	CONTROL- KEY	All data elements excluding the root field in the order in which they appear in the internal record (CONTROL-KEY excluded).	

Standard Related Record Format	▶	ffff RELATED FILE NAME	ACCESS-KEY		All data elements in the order in which they appear in the internal record (ACCESS-KEY excluded).	

Coded Related Record Format	▶	ffff RELATED FILE NAME	RECORD CODE	ACCESS- KEY	All data elements in the order in which they appear in the internal record (RECORD CODE and ACCESS-KEY excluded).	

Coding the LINKPATH statement

With the LINKPATH statement, you can identify the linkpath you want used as the primary linkpath to load a related file.

rrrrLINKPATH=ppppLKxx

Restriction	Use this statement only for related files.
Description	<i>Optional.</i> Identifies the primary linkpath you want used to load a related file.
Format	<i>rrrr</i> 4-character related file name. <i>ppppLKxx</i> The linkpath name as coded in the schema.

Considerations

- ◆ The primary linkpath and its associated control key that you use must be in the base portion of a coded record rather than the redefined portion.
- ◆ This statement must immediately precede the Element List statement of the related file to which it applies.

Loading Directory files

You must load Directory files and PDM files in separate jobs.

If you are loading all the Directory files, the run control statements and file control statements are in data member CSUSLOAD. After installation is complete, you cannot modify this member.

In your CSIPARM file, you must code the bootstrap schema and environment description. When you code the SCHEMA control statement in “Coding the SCHEMA statement” on page 304, you should use the same schema that you coded in the CSIPARM file. “Coding CSIPARM file and run control statements for directory files” on page 231 shows how to coordinate coding your CSIPARM file with your run control statements.

Coding the Insert Linkpath function

The Insert Linkpath function (CSUINSRT) dynamically inserts linkpath data into primary files without your unloading and reloading them. The Insert Linkpath function also inserts linkpath data into Directory or PDM files.

You use this function when you unload and load part of your files. For example, you may unload several related files and one of the primary files to which they are connected, but you may not unload all the primary files to which they are connected. After you reload, you need to insert the connections to the files you did not unload. For more information on when to use this function, see “What to do with linkpaths when you unload and load” on page 228.

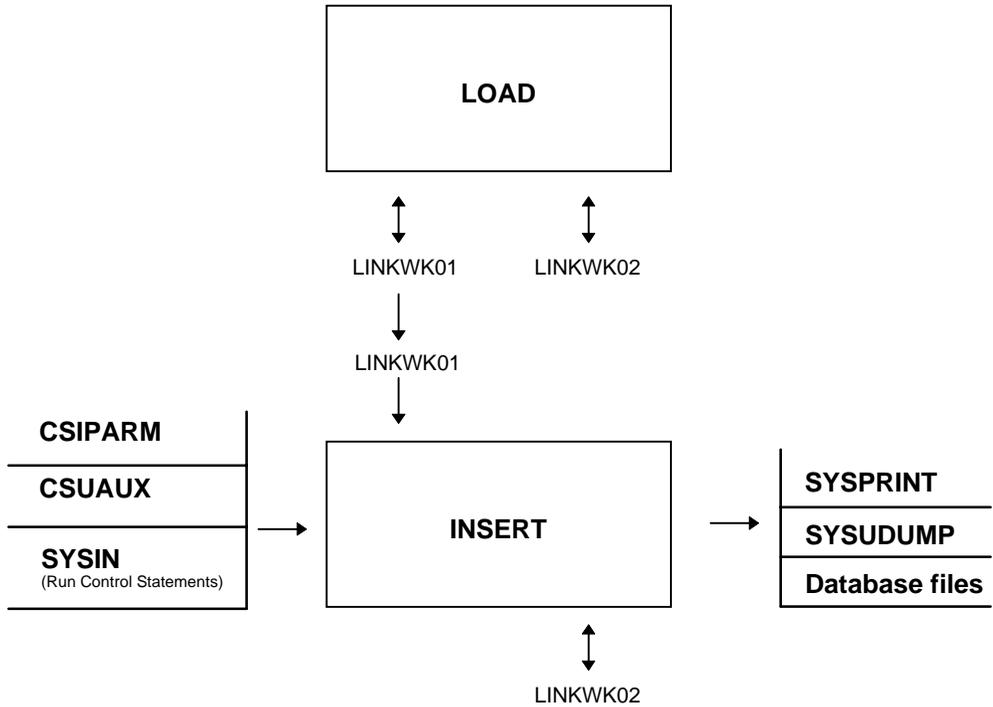
If you are already unloading and loading all the primary files, you do not need to use this function because the Load function inserts linkpath data. The Load function also creates the input to this function when it loads the associated related files. The following figures illustrate the output of the Load function becoming the input to the Insert Linkpath function.



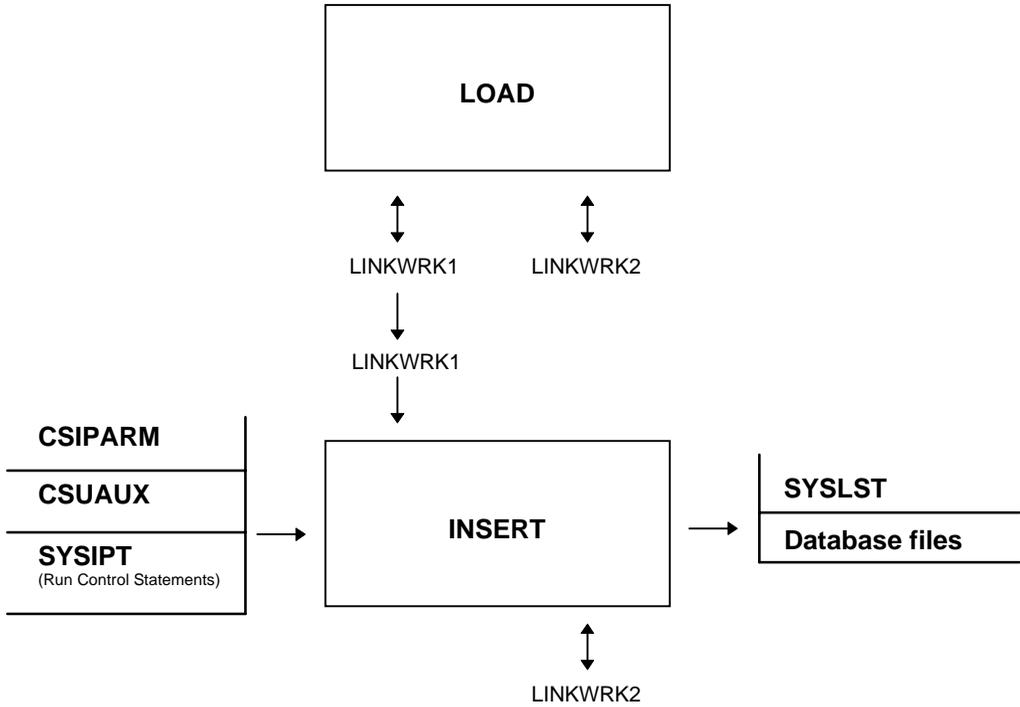
Use this function only with files loaded by the Version 2 Load function.

To use the Insert Linkpath function, you need to code file definitions and control statements. JCL examples follow the explanation of the statements.

OS/390 The following figure shows files you define in OS/390 JCL to insert linkpath data:



VSE The following figure shows files you define in VSE JCL to insert linkpath data:



Defining files

To execute the Insert Linkpath function in OS/390 or VSE, you must define the files listed in the following table in your JCL and execute the Insert Linkpath program, CSUINSRT. In OS/390, rather than coding all the file definitions, you can use the cataloged procedure TISUTINS.

File	Description	Considerations
CSIPARM:	Identifies the CSIPARM file which contains control information that the PDM needs.	See “Coding the CSIPARM file for Unload, Load, and Insert Linkpath” on page 230.
CSUAUX:	Holds the auxiliary information for the files that are not in native format.	See “Defining the CSUAUX file” on page 238.
fffffff	Indicates the SUPRA primary files you want processed by the Insert Linkpath function.	You may specify up to 57 primary files. File names must be specified in the SUPRA Directory for the schema you are using. For VSE, you must always code primary files as direct access on their DLBL statements.
OS/390 LINKWK01 VSE LNKWRK1	Indicates the first linkage work file, which contains linkpath data generated in the related file load for insertion into the associated primary files.	See “Defining the LINKWK01/LNKWRK1 file” on page 326.
OS/390 LINKWK02 VSE LNKWRK2	Indicates the second linkage work file for synonym processing.	See “Defining the LINKWK02/LNKWRK2 file” on page 327.
OS/390 SYSIN VSE SYSIPT	Holds the run control statements.	See “Coding control statements” on page 328.
VSE SYSLST OS/390 SYSPRINT	Indicates the output file for the printed listing of all control statements, diagnostics messages, etc.	
OS/390 SYSUDUMP	Indicates a dump file.	Optional.

Defining the LINKWK01/LNKWRK1 file

The first linkage work file in OS/390 is LINKWK01; in VSE it is LNKWRK1. The Load function creates this file while loading associated related files. It uses the file to hold linkpath pointers to the first and last record for each key on the related files. This file becomes input to the Insert Linkpath function. This function uses the file to update pointers to the first and last records in the corresponding primary records.

Since this is the same file you defined in the Load function, you must code the same LRECL and BLKSIZE parameters that you coded for the Load function.

You must also maintain the same format for the data that you used in the Load function:

Position in bytes	Contents
1–4	Primary file name.
5–8	Home location relative record number (RQLOC value) of primary record.
9–12	Linkpath name (LKxx).
13–16	RRN of first record in the related chain.
17–20	RRN of last record in the related chain.
21– <i>n</i>	Primary file control key of the record you want to load. Maximum key length is 256 bytes.

Defining the LINKWK02/LNKWRK2 file

The second linkage work file in OS/390 is LINKWK02; in VSE it is LNKWRK2. It is used to hold records that have no keys in the Primary file. After the last record for the Primary file is processed, the work file is used as input for dummy record insertions into the Primary file.

For OS/390, LINKWK02 LRECL and BLKSIZE must be the same as LINKWK01. For VSE, LNKWRK2 RECSIZE and BLKSIZE must be the same as LNKWRK1.

This file does not have the same number of records as the corresponding file for the Load function. In this function, determine the number of records for this file in three steps:

1. Execute the File Statistics function to obtain the number of out-of-block synonyms for each primary file you loaded.
2. Multiply each file's number of out-of-block synonyms by the number of linkpaths it contains.
3. Use the largest number for one file.

Coding control statements

The INSERT and LINKWKnn statements control the execution of the Insert Linkpath function. The following table briefly describes each statement. Some statements may require more than one record, but you cannot put two different statements on the same record.

Statement	Description	Section
INSERT	Names the primary files into which you want to insert linkage information.	“Coding the INSERT statement” on page 328
LINKWKnn (VSE only)	Defines the characteristics of the LNKWRK1 and LNKWRK2 files.	“Coding the LINKWKnn statements (VSE only)” on page 331

Coding the INSERT statement

Use the INSERT statement to name the primary files into which you want to insert linkage information. You can put the DBMOD, FILES, and CLEARLKS parameters in any order. You can continue them over several records by placing a continuation character in position 72.

```

INSERT      FILES = ( { ALL.
                    [fff1[fff2...fffn].] } )

            [, CLEARLKS = (ppppLKxx1[ppppLKxx2... ppppLKxxn].) ]

            [, DBMOD = schemaname ]

            ,END

```

INSERT

Description *Required.* Identifies the control statement.

Consideration Place at least one blank character before and after the INSERT parameter.

$$\text{FILES} = \left(\left\{ \begin{array}{l} \text{ALL.} \\ \text{fff}_1[\text{fff}_2 \dots \text{fff}_n]. \end{array} \right\} \right)$$

Description *Required.* Names the file(s) you want to update.

Format File names must be 4 alphanumeric characters. Terminate the list with a period, and surround it with parentheses. Do not separate the file names.

Considerations

- ◆ List the file names in alphabetical order.
- ◆ If you code a list of files in the FILES parameter and you want to clear a linkpath in one particular file, you must name the linkpath in the CLEARLKS parameter. When you code the CLEARLKS parameter, you must replace the *pppp* portion with a file you named in the FILES parameter. If you code linkpaths that do not match a file in the FILES parameter, they are ignored.
- ◆ The function updates only the files you code even if the first linkage work file contains linkage information for other files.

,CLEARLKS=(ppppLKxx1[ppppLKxx2...ppppLKxxn].)

Description *Optional.* Indicates that you want blanks moved into the linkpath you coded for each record of the primary file if there is no linkage data for the linkpath.

Format Linkpath names must be 8 alphanumeric characters. Terminate the list with a period, and surround it with parentheses. Do not separate the linkpath names.

Considerations

- ◆ The linkpath names must be defined in the schema you coded.
- ◆ You must list linkpaths in alphabetical order.
- ◆ This parameter is needed because the linkage work files do not necessarily contain linkage data for every record in the primary file. If there is linkage data for a record, the function inserts it in the appropriate linkpath field. If there is no data and you have coded the linkpath in CLEARLKS, the function sets the linkpath field to blanks.

If there is no data and you have not coded the linkpath in the CLEARLKS parameter, the function leaves the linkpath untouched. In that case, the linkpath field contains the linkage data from before it was unloaded and loaded. Since that linkage data may be invalid, your database may be corrupt.

- ◆ In the FILES parameter, you should list all linkpath names for all files that have linkpath data to be inserted. This will take care of the situation where there is no linkage data for a particular record and the linkpath contains incorrect data.

,DBMOD=schemaname

Description *Optional.* Use this parameter only for compatibility with existing Series 80 job streams; it is ignored.

Format 1–8 alphanumeric characters

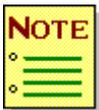
Consideration Do not code this parameter when creating new job streams.

,END.

Description *Required.* Indicates the termination of the control statement.

Coding the LINKWK nn statements (VSE only)

Use the LINKWK nn statements to define the characteristics of the linkage work files: LNKWRK1 and LNKWRK2. You can code any parameter on more than one line.



You must code the parameters identically to those you coded on the LINKWK nn statements in the step that loaded the associated related files. For information on this file in the Load function, see “Coding the LINKWK nn statements (VSE only)” on page 308.

$$\text{LINKWK}_{nn}: \left[\text{DEVICE} = \left\{ \begin{array}{l} \text{DISK} \\ \text{TAPE} \end{array} \right. \left[, \text{FILABL} = \left\{ \begin{array}{l} \text{NO} \\ \text{STD} \end{array} \right\} \right] \right] \right]$$

[,BLKSIZE = n]

[,RECSIZE = n] ,DEVADDR =SYS $nnnn$

LINKWK nn :

Description	<i>Required.</i> Identifies the LINKWK nn statement.
Options	LINKWK01: The function defines file LNKWRK1. LINKWK02: The function defines file LNKWRK2.
Format	Code in positions 1–9.

$$, \text{DEVICE} = \left\{ \begin{array}{l} \text{DISK} \\ \text{TAPE} \end{array} \right\}$$

Description	<i>Optional.</i> Indicates the device type of the file.
Options	DISK Disk device TAPE Magnetic tape unit

,FILABL = $\left\{ \begin{array}{l} \text{NO} \\ \text{STD} \end{array} \right\}$

Restriction	This parameter is valid only when DEVICE=TAPE.	
Description	<i>Optional.</i> Indicates whether the tape contains file labels.	
Default	NO	
Options	NO	Does not contain labels
	STD	Contains standard labels

,BLKSIZE=*n*

Restriction	For LINKWK01, you must code the same value you used in the step that loaded the associated related files.
Description	<i>Optional.</i> Indicates the file's block size in bytes.
Format	Use a numeric character.
Consideration	To determine the size, see “ Defining the LINKWK01/LNKWRK1 file ” on page 326 and “ Defining the LINKWK02/LNKWRK2 file ” on page 327.

,RECSIZE=*n*

Restriction	<i>Required</i> for the LINKWK01 statement. <i>Optional</i> for the LINKWK02 statement. For the LINKWK01 statement, you must code the same value you used in the step that loaded the associated related files.
Description	Indicates the file's record size in bytes.
Format	One numeric character.
Consideration	To determine the size, see “ Defining the LINKWK01/LNKWRK1 file ” on page 326 and “ Defining the LINKWK02/LNKWRK2 file ” on page 327.

,DEVADDR=SYS*nnn*

Description	<i>Required.</i> Indicates the device address (SYS number symbolic unit) associated with the file.
Format	<i>nnn</i> Must be 3 digits

Examples of Unload, Load, and Insert Linkpath functions

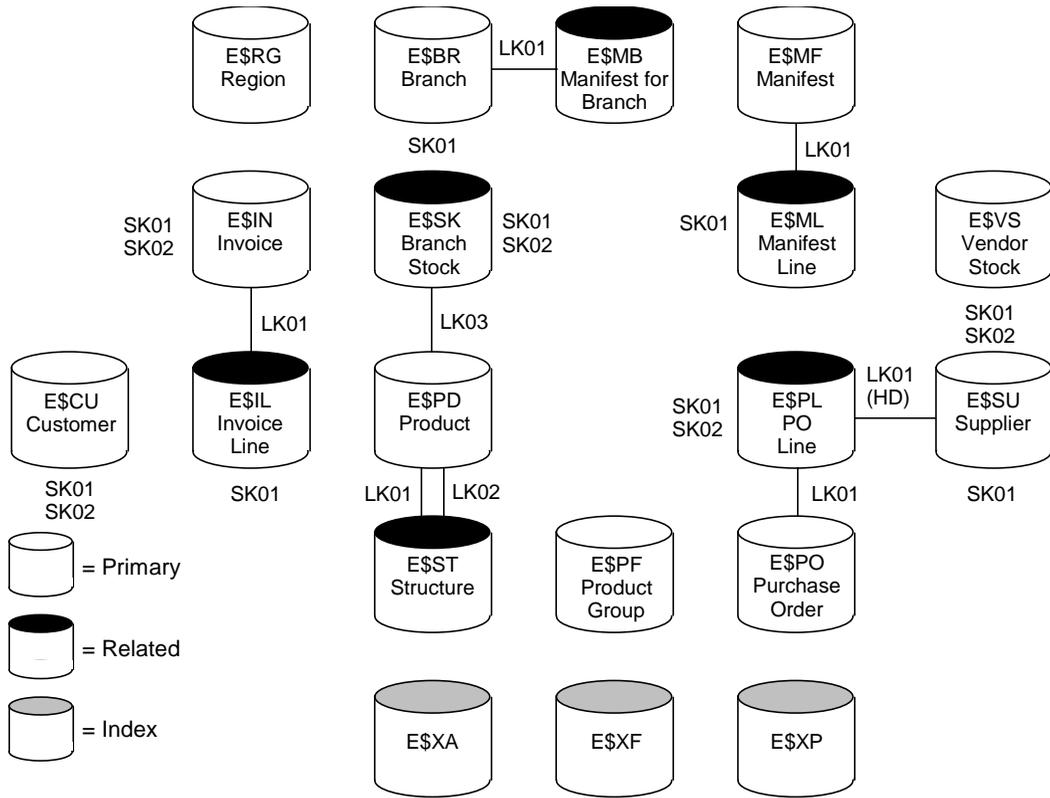
Two examples illustrate how to use the Unload, Load, and Insert Linkpath functions. The first example shows how to unload and reload all the files in the Burry's database. The second one shows how to unload, change the structure, and reload four of the files: two primary and two related. The structure change shows how to use the *FILL parameter to increase the size of elements and an exit program to decrease the size of elements in the files. The second example also shows how to use the Insert Linkpath function. Both examples illustrate when to clear linkpaths.

Since it is necessary to depopulate and repopulate files, those steps are shown in both examples. Since the intention here is to reload immediately, the examples show the index files depopulated before unloading. When you are unloading to get a backup copy that you may never reload, you do not need to depopulate. However, if you ever want to reload the backup copy, you must depopulate first.

To help you understand the examples, The following figure shows the files in Burry's database. "[Internal schema of Burry's database](#)" on page 335 and "[Internal schema of files before unloading](#)" on page 338 show the internal schema of the files. The latter shows the files that will change as they are unloaded. The modified internal schema in "[Internal schema of files after unloading](#)" on page 340 shows the changed files as they are loaded.



The descriptions of the Burry's database files may not match those on the release of SUPRA that you have. Therefore, do not use these descriptions as a basis for decisions you make about Burry's. In addition, these descriptions are not complete; they contain only the information you need to unload and load.



The files in the preceding figure are listed alphabetically in “Internal schema of Burry’s database” on page 335. “Internal schema of files before unloading” on page 338 lists files whose structures change.

Internal schema of Burry's database

Name of file	Type of file	Physical fields	Length of physical fields	Name of secondary keys
E\$BR	Primary	E\$BRROOT	8	E\$BRSK01
		E\$BRCTRL	4	
		E\$BRLK01	8	
		E\$BRNAME	20	
		E\$BRADDR	20	
		E\$BRCITY	13	
		E\$BRSTAT	2	
		E\$BRZIPC	5	
		E\$BRREGN	3	
		E\$BRDRTE	2	
		E\$BRSALQ	9	
		E\$BRSTFQ	5	
		E\$CU	Primary	
E\$CUCTRL	6			E\$CUSK02
E\$CUNAME	20			
E\$CUADDR	20			
E\$CUCITY	13			
E\$CUSTAT	2			
E\$CUZIPC	5			
E\$CUCLAS	2			
E\$CUCRAT	2			
E\$CUCLIM	9			
E\$CUBRAN	4			

Name of file	Type of file	Physical fields	Length of physical fields	Name of secondary keys
E\$IL	Related	E\$ILE\$IN	4	E\$I LSK01
		E\$INLK01	8	
		E\$ILE\$PD	9	
		E\$ILQNTY	5	
		E\$ILPRCE	9	
E\$IN	Primary	E\$INROOT	8	E\$I NSK01
		E\$ICTRLN	4	
		\$INLK01	8	
		E\$INLK04	8	
		E\$I NSLMN	4	
		E\$INTOTL	9	
		E\$INBRAN	4	
		E\$I NDATE	5	
		E\$I NCUST	6	
E\$MB	Related	E\$MBE\$BR	4	none
		E\$BRLK01	8	
		E\$MBE\$MF	5	
		E\$MBFILL	4	
E\$MF	Primary	E\$MFROOT	8	none
		E\$MFCTRL	5	
		E\$MFLK01	8	
		E\$MFTOTL	9	
		E\$MFBRAN	4	
		E\$MFDATE	5	
E\$ML	Related	E\$MLE\$MF	5	E\$MLSK01
		E\$MFLK01	8	
		E\$MLE\$PD	9	
		E\$MLQNTY	5	
		E\$MLVLUE	9	

Name of file	Type of file	Physical fields	Length of physical fields	Name of secondary keys
E\$PG	Primary	E\$PGROOT	8	none
		E\$PGCTRL	2	
		E\$PGDESC	30	
E\$RG	Primary	E\$RGROOT	8	
		E\$RGCTRL	3	
		E\$RGNAME	20	
E\$SK	Related	E\$SKE\$BR	4	E\$SKSK01
		E\$SKE\$PD	9	
		E\$PDLK03	8	
		E\$SKQNTY	5	
		E\$SKBINL	5	
		E\$SKSYTD	9	
E\$SU	Primary	E\$SUROOT	8	E\$SUSK01
		E\$SUCTRL	6	
		E\$SULK01	8	
		E\$SUNAME	20	
		E\$SUADDR	20	
		E\$SUCITY	13	
		E\$SUSTAT	2	
		E\$SUZIPC	5	
E\$VS	Primary	E\$VSROOT	8	E\$VSSK01
		E\$VSCTRL	15	
		E\$VSE\$SU		
		E\$VSE\$PD		
		E\$VSNUMB		
		E\$VSV CST		

To see the change in structure, you need additional information about the files: the logical record length, total logical records, type of physical field, and the number of decimal places. The type of field can be binary, character, or zoned decimal, which is shown as B, C, and Z.

Internal schema of files before unloading

Name of file	Type of file	Physical fields	Length of physical fields	Type of physical field	Decimal places	Name of secondary keys
E\$PD	Primary	E\$PDROOT	8	B	0	E\$PDLK01
		E\$PDCTRL	9	C	0	
		E\$PDLK01	8	B	0	
		E\$PDLK02	8	B	0	
		E\$PDLK03	8	B	0	
		E\$PDDESC	30	C	0	
		E\$PDWQTY	5	Z	0	
		E\$PDPRCE	9	Z	2	
E\$PDPGRP	<u>2</u>	C	0			
LOGICAL RECORD LENGTH =			87			
TOTAL LOGICAL RECORDS =			484			
E\$PO	Primary	E\$POROOT	8	B	0	none
		E\$POCTRL	6	C	0	
		E\$POLK01	8	B	0	
		E\$POTOTL	9	Z	2	
		E\$PODATE	<u>5</u>	Z	0	
LOGICAL RECORD LENGTH =			36			
TOTAL LOGICAL RECORDS =			1177			
E\$ST	Related	E\$STASSM	9	C	0	none
		E\$PDLK01	8	B	0	
		E\$STCOMP	9	C	0	
		E\$PDLK02	8	B		
		E\$STQNTY	<u>5</u>	Z	0	
LOGICAL RECORD LENGTH =			39			
TOTAL LOGICAL RECORDS =			1078			

Name of file	Type of file	Physical fields	Length of physical fields	Type of physical field	Decimal places	Name of secondary keys
E\$PL	Coded Related					
Base portion		E\$PLCODE	2	C	0	E\$PLSK01
		E\$PLE\$PO	6	C	0	E\$PLSK02
		E\$POLK01	8	B	0	
		E\$PLDATA	<u>31</u>	C	0	
LOGICAL RECORD LENGTH =			47			
HD portion redefines E\$PLDATA						
		E\$PLE\$SU	6	C	0	
		E\$SULK01	8	B	0	
		E\$PLDATE	5	Z	0	
		E\$PLFILL	<u>12</u>	C	0	
LOGICAL RECORD LENGTH of Redefined portion =			31			
LN portion redefines E\$PLDATA						
		E\$PLE\$PD	9	C	0	
		E\$PLQNTY	5	Z	0	
		E\$PLCOST	9	Z	2	
		E\$PLFILL	<u>8</u>	C	0	
LOGICAL RECORD LENGTH of Redefined portion =			31			
PD portion redefines E\$PLDATA						
		E\$PLDELN	2	B	0	
		E\$PLDELD	5	Z	0	
		E\$PLDELQ	5	Z	0	
		E\$PLDELP	9	C	0	
		E\$PLFILR	<u>10</u>	C	0	
LOGICAL RECORD LENGTH of Redefined portion =			31			
TOTAL LOGICAL RECORDS =			902			

Internal schema of files after unloading

Name of file	Type of file	Physical fields	Length of physical fields	Type of physical field	Decimal places	Name of secondary keys	
E\$PD	Primary	E\$PDROOT	8	B	0	E\$PDLK01	
		E\$PDCTRL	9	C	0		
		E\$PDLK01	8	B	0		
		E\$PDLK02	8	B	0		
		E\$PDLK03	8	B	0		
		E\$PDDESC	30	C	0		
		E\$PDWQTY	5	Z	0		
		*	E\$PDPRCE	7	Z		0
		*	E\$PDPGRP	12	C		0
*	E\$PDDES2	<u>20</u>	C	0			
* LOGICAL RECORD LENGTH =			115				
TOTAL LOGICAL RECORDS =			484				
E\$PO	Primary	E\$POROOT	8	B	0		
		E\$POCTRL	6	C	0		
		E\$POLK01	8	B	0		
		E\$POTOTL	9	Z	2		
		E\$PODATE	<u>5</u>	Z	0		
LOGICAL RECORD LENGTH =			36				
* TOTAL LOGICAL RECORDS =			1200				
E\$ST	Related	E\$STASSM	9	C	0		
		E\$PDLK01	8	B	0		
		E\$STCOMP	9	C	0		
		E\$PDLK02	8	B	0		
		E\$STQNTY	5	Z	0		
		*	E\$STCOMM	<u>20</u>	C	0	
* LOGICAL RECORD LENGTH =			59				
TOTAL LOGICAL RECORDS =			1078				

Name of file	Type of file	Physical fields	Length of physical fields	Type of physical field	Decimal places	Name of secondary keys
E\$PL	Coded, Related					
Base portion		E\$PLCODE	2	C	0	E\$PLSK01
		E\$PLE\$PO	6	C	0	E\$PLSK02
		E\$POLK01	8	B	0	
		E\$PLDATA	<u>31</u>	C	0	
LOGICAL RECORD LENGTH =			47			
HD portion redefines E\$PLDATA						
		E\$PLE\$SU	6	C	0	
		E\$SULK01	8	B	0	
		E\$PLDATE	5	Z	0	
		E\$PLFILL	<u>12</u>	C	0	
LOGICAL RECORD LENGTH of Redefined portion =			31			
LN portion redefines E\$PLDATA						
		E\$PLE\$PD	9	C	0	
			5	Z	0	
			9	Z	2	
			<u>8</u>	C	0	
LOGICAL RECORD LENGTH of Redefined portion =			31			
PD portion redefines E\$PLDATA						
		E\$PLDELN	2	B	0	
		E\$PLDELD	5	Z	0	
		E\$PLDELQ	5	Z	0	
		E\$PLDELP	9	C	0	
		E\$PLFILR	<u>10</u>	C	0	
LOGICAL RECORD LENGTH of Redefined portion =			31			
* TOTAL LOGICAL RECORDS =			950			

Unloading and loading all of Burry's database files

You unload and reload all of your files to improve performance after many updates have changed the files' structure. You can determine when you need to unload in two ways:

- ◆ When your applications begin finding broken linkpath chains.
- ◆ When your files are no longer structured for best performance, for example, when your primary files have many out-of-block synonyms. You can determine whether you have this problem by executing the File Statistics function regularly.

Unloading and reloading files have the following benefits:

- ◆ Repairing broken linkpath chains
- ◆ Reorganizing the synonym chains in primary files to minimize the number of out-of-block synonyms
- ◆ Reorganizing the linkpath chains in related files to optimize access along the primary linkpath
- ◆ Reorganizing the secondary key tree structure in the index files

The last function is actually a result of depopulating and repopulating—steps you must take before and after you unload and load. You execute four functions when you unload and load: the **Depopulate**, **Unload**, **Load**, and **Sorted-Populate** functions.

Sample UCL and control statements

The following UCL and control statements illustrate how to code these four functions in OS/390.

VSE

In VSE, you need to include other run control statements to unload and load, for example, the RECFORM statement.

The first step in unloading is to depopulate all your secondary keys. The following UCL is for the **Depopulate** function:

```
CONTROL (BEGIN)
*
ENV-DESC (BURRYENN)
SCHEMA (BURRYSCH)
LIST (ALL)
DATA-FORMAT (HEX CHAR)
DIAGNOSTICS (EXTENDED)
*
FUNCTION (DEPOPULATE)
STATISTICS (ALL)
FILE (E$BR)
FILE (E$CU)
FILE (E$IL)
FILE (E$IN)
FILE (E$ML)
FILE (E$PD)
FILE (E$PL)
FILE (E$SK)
FILE (E$SU)
FILE (E$VS)
*
CONTROL (END)
```

After you depopulate your secondary keys, you can unload your files. The following input includes the CSIPARM file, auxiliary file, run control, and file control statements to unload all of Burry's database files and clear all linkpaths:

```
//CSIPARM DD *                                CSIPARM INPUT
DIRECTORY=(SCHEMA=CSTASCHM,ENVDESC=CSTANONE),
REALM=(SCHEMA=BURRYSCH,ENVDESC=BURRYENN),
END.
/*
/*
//CSUAUX DD *                                AUXILIARY INPUT FILE
FILE=E$IL LOAD=NATIVE
FILE=E$MB LOAD=NATIVE
FILE=E$ML LOAD=NATIVE
FILE=E$PL LOAD=NATIVE
FILE=E$SK LOAD=NATIVE
FILE=E$ST LOAD=NATIVE
FILE=E$BR LOAD=NATIVE
FILE=E$CU LOAD=NATIVE
FILE=E$IN LOAD=NATIVE
FILE=E$MF LOAD=NATIVE
FILE=E$PD LOAD=NATIVE
FILE=E$PG LOAD=NATIVE
FILE=E$PO LOAD=NATIVE
FILE=E$RG LOAD=NATIVE
FILE=E$SU LOAD=NATIVE
FILE=E$VS LOAD=NATIVE
/*
//SYSIN DD *                                RUN CONTROL RECORDS
RELATED:E$ILE$MBE$MLE$PLE$SKE$STEND.
PRIMARY:E$BRE$CUE$INE$MFE$PDE$PGE$POE$RGE$SUE$VSEND.
SORTNAME=SORT
DUMP=YES
```

```
/*
/**
//PARM DD * FILE CONTROL RECORDS
E$ILLINKPATH=E$INLK01
E$ILALL.END.
E$MBLINKPATH=E$BRLK01
E$MBALL.END.
E$MLLINKPATH=E$MFLK01
E$MLALL.END.
E$PLLINKPATH=E$POLK01
E$PLALL.END.
E$SKLINKPATH=E$PDLK03
E$SKALL.END.
E$STLINKPATH=E$PDLK01
E$STALL.END.
E$BRALL.END.
E$BRBLANK-LINKS=LK01END.
E$CUALL.END.
E$INALL.END.
E$INBLANK-LINKS=LK01END.
E$MFALL.END.
E$MFBLANK-LINKS=LK01END.
E$PDALL.END.
E$PDBLANK-LINKS=LK01LK02LK03END.
E$PGALL.END.
E$POALL.END.
E$POBLANK-LINKS=LK01END.
E$RGALL.END.
E$SUALL.END.
E$SUBLANK-LINKS=LK01END.
E$VSALL.END.
```

After you unload your files, you can reload them. The following input includes CSIPARM file, auxiliary file, run control, and file control statements for the Load function:

```
//CSIPARM DD *                                CSIPARM INPUT
DIRECTORY=(SCHEMA=CSTASCHM, ENVDESC=CSTANONE),
REALM=(SCHEMA=BURRYSCH, ENVDESC=BURRYENN),
END.
/*
//*
//CSUAUX DD *                                AUXILIARY INPUT FILE
FILE=E$IL LOAD=NATIVE
FILE=E$MB LOAD=NATIVE
FILE=E$ML LOAD=NATIVE
FILE=E$PL LOAD=NATIVE
FILE=E$SK LOAD=NATIVE
FILE=E$ST LOAD=NATIVE
FILE=E$BR LOAD=NATIVE
FILE=E$CU LOAD=NATIVE
FILE=E$IN LOAD=NATIVE
FILE=E$MF LOAD=NATIVE
FILE=E$PD LOAD=NATIVE
FILE=E$PG LOAD=NATIVE
FILE=E$PO LOAD=NATIVE
FILE=E$RG LOAD=NATIVE
FILE=E$SU LOAD=NATIVE
FILE=E$VS LOAD=NATIVE
/*
```

```

/*
/*
//SYSIN      DD *
RELATED: E$ILE$MBE$MLE$PLE$SKE$STEND.
PRIMARY: E$BRE$CUE$INE$MFE$PDE$PGE$POE$RGE$SUE$VSEND.
SCHEMA=BURRYSCH
SORTNAME=SORT
E$ILLINKPATH=E$INLK01
E$ILALL.END.
E$MBLINKPATH=E$BRLK01
E$MBALL.END.
E$MLLINKPATH=E$MFLK01
E$MLALL.END.
E$PLLINKPATH=E$POLK01
E$PLALL.END.
E$SKLINKPATH=E$PDLK03
E$SKALL.END.
E$STLINKPATH=E$PDLK01
E$STALL.END.
E$BRALL.END.
E$CUALL.END.
E$INALL.END.
E$MFALL.END.
E$PDALL.END.
E$PGALL.END.
E$POALL.END.
E$RGALL.END.
E$SUALL.END.
E$VSALL.END.
/*

```

After you reload your files, you need to repopulate all your index files with secondary keys. The following UCL is for the **Sorted-Populate** function:

```
CONTROL(BEGIN)
*
  ENV-DESC (BURRYENN)
  SCHEMA   (BURRYSCH)
  LIST (ALL)
    DATA-FORMAT (HEX CHAR)
  DIAGNOSTICS (EXTENDED)
*
FUNCTION (SORTED- POPULATE)
STATISTICS (ALL)
  FILE (E$BR)
  FILE (E$CU)
  FILE (E$IL)
  FILE (E$IN)
  FILE (E$ML)
  FILE (E$PD)
  FILE (E$PL)
  FILE (E$SK)
  FILE (E$SU)
  FILE (E$VS)
*
CONTROL(END)
```

Sample listings

The following listings illustrate the output you receive as a result of the sample UCL and statements:

```

TTTTTTTT      IIIIII      SSSSSSS
TTTTTTTT      IIIIII      SSSSSSSSS
TT            II          SS      SS
TT            II          SS
TT            II          SSSSSSSS
TT            II          SSSSSSSS
TT            II          SS      SS
TT            II          SS      SS
TT            IIIIII     SSSSSSSS
TT            IIIIII     SSSSSSSS
DDDDDDDD      BBBB BBBB      AAA
DDDDDDDD      BBBB BBBB      AAAA
DD  DD  BB      BB  AA  AA
DD  DD  BB      BB  AA  AA
DD  DD  BBBB BBBB  AA  AA
DD  DD  BBBB BBBB  AAAAAAAAAA
DD  DD  BB      BB  AAAAAAAAAA
DD  DD  BB      BB  AA  AA
DDDDDDDD      BBBB BBBB      AA  AA
DDDDDDDD      BBBB BBBB      AA  AA
UU  UU  TTTTTTTT IIIIII      LL          IIIIII      TTTTTTTT      IIIIII      EEEEEEEEE  SSSSSSS
UU  UU  TTTTTTTT IIIIII      LL          IIIIII      TTTTTTTT      IIIIII      EEEEEEEEE  SSSSSSSSS
UU  UU  TT        II          LL          II          TT          II          EE          SS      SS
UU  UU  TT        II          LL          II          TT          II          EE          SS
UU  UU  TT        II          LL          II          TT          II          EEEEEEE  SSSSSSSS
UU  UU  TT        II          LL          II          TT          II          EEEEEEE  SSSSSSSS
UU  UU  TT        II          LL          II          TT          II          EE          SS      SS
UU  UU  TT        II          LL          II          TT          II          EE          SS      SS
UUUUUUUU      TT          IIIIII      LLLLLLLL      IIIIII      TT          IIIIII      EEEEEEEEE  SSSSSSSS
UUUUUUU      TT          IIIIII      LLLLLLLL      IIIIII      TT          IIIIII      EEEEEEEEE  SSSSSSS
                                F U N C T I O N :   U N L O A D
                                E N V I R O N M E N T :
                                RELATED: E$ILE$MBE$MLE$PLE$SKE$STEND.                00030000
                                PRIMARY: E$BRE$CUE$INE$MFE$PDE$PGE$POE$RGE$SUE$VSEND. 00020000
                                SORTNAME=SORT                                     00050000
                                DUMP=YES                                         00060000
TIS DATABASE ADMINISTRATOR UTILITIES          CINCOM SYSTEMS, INC.          88/309          13:21:24
PAGE          2CSUAUX FILE RECORDS:

FILE=E$IL LOAD=NATIVE                00030000
FILE=E$MB LOAD=NATIVE                00030000
FILE=E$ML LOAD=NATIVE                00030000
FILE=E$PL LOAD=NATIVE                00030000
FILE=E$SK LOAD=NATIVE                00030000
FILE=E$ST LOAD=NATIVE                00030000
FILE=E$BR LOAD=NATIVE                00030000
FILE=E$CU LOAD=NATIVE                00030000
FILE=E$IN LOAD=NATIVE                00030000
FILE=E$MF LOAD=NATIVE                00030000
FILE=E$PD LOAD=NATIVE                00030000
FILE=E$PG LOAD=NATIVE                00030000
FILE=E$PO LOAD=NATIVE                00030000
FILE=E$RG LOAD=NATIVE                00030000
FILE=E$SU LOAD=NATIVE                00030000
FILE=E$VS LOAD=NATIVE                00030000

END OF CSUAUX FILE RECORDS.
NO ERRORS ENCOUNTERED IN THE CSUAUX FILE.
                                BEGINNING THE UNLOAD FUNCTION.                E$ILLINKPATH=E$INLK01
                                E$ILALL.END.
                                E$IL UNLOADED SUCCESSFULLY
                                NUMBER OF RECORDS UNLOADED =                217

```

E\$MBLINKPATH=E\$BRLK01	
E\$MBALL.END.	
E\$MB UNLOADED SUCCESSFULLY	
NUMBER OF RECORDS UNLOADED =	29
E\$MLLINKPATH=E\$MFLK01	
E\$MLALL.END.	
E\$ML UNLOADED SUCCESSFULLY	
NUMBER OF RECORDS UNLOADED =	78
E\$PLLINKPATH=E\$POLK01	
E\$PLALL.END.	
E\$PL UNLOADED SUCCESSFULLY	
NUMBER OF RECORDS UNLOADED =	122
E\$SKLINKPATH=E\$PDLK03	
E\$SKALL.END.	
E\$SK UNLOADED SUCCESSFULLY	
NUMBER OF RECORDS UNLOADED =	2,628
E\$STLINKPATH=E\$PDLK01	
E\$STALL.END.	
E\$ST UNLOADED SUCCESSFULLY	
NUMBER OF RECORDS UNLOADED =	67
E\$BRALL.END.	
E\$BRBLANK-LINKS=LK01END.	
E\$BR UNLOADED SUCCESSFULLY	
NUMBER OF RECORDS UNLOADED =	39
E\$CUALL.END.	
E\$CU UNLOADED SUCCESSFULLY	
NUMBER OF RECORDS UNLOADED =	43
E\$INALL.END.	
E\$INBLANK-LINKS=LK01END.	
E\$IN UNLOADED SUCCESSFULLY	
NUMBER OF RECORDS UNLOADED =	96
E\$MFALL.END.	
E\$MPBLANK-LINKS=LK01END.	
E\$MF UNLOADED SUCCESSFULLY	
NUMBER OF RECORDS UNLOADED =	29
E\$PDALL.END.	
E\$PDBLANK-LINKS=LK01LK02LK03END.	
E\$PD UNLOADED SUCCESSFULLY	
NUMBER OF RECORDS UNLOADED =	88
E\$PGALL.END.	
E\$PG UNLOADED SUCCESSFULLY	
NUMBER OF RECORDS UNLOADED =	6
E\$POALL.END.	
E\$POBLANK-LINKS=LK01END.	
E\$PO UNLOADED SUCCESSFULLY	
NUMBER OF RECORDS UNLOADED =	26
E\$RGALL.END.	
E\$RG UNLOADED SUCCESSFULLY	
NUMBER OF RECORDS UNLOADED =	10
E\$SUALL.END.	
E\$SUBLANK-LINKS=LK01END.	
E\$SU UNLOADED SUCCESSFULLY	
NUMBER OF RECORDS UNLOADED =	14
E\$VSALL.END.	
E\$VS UNLOADED SUCCESSFULLY	
NUMBER OF RECORDS UNLOADED =	191
UNLOAD FUNCTION COMPLETE.	
F U N C T I O N: L O A D	
E N V I R O N M E N T	
RELATED:E\$ILE\$MBE\$MLE\$PLE\$SKE\$STEND.	00030000
PRIMARY:E\$BRE\$CUE\$INE\$MFE\$PDE\$PGE\$POE\$RGE\$SUE\$VSEND.	00020000
SCHEMA=BURRYSCH	00050000
SORTNAME=SORT	00050000

```

CSUAUX FILE RECORDS:
FILE=E$IL LOAD=NATIVE                00030000
FILE=E$MB LOAD=NATIVE                00030000
FILE=E$ML LOAD=NATIVE                00030000
FILE=E$PL LOAD=NATIVE                00030000
FILE=E$SK LOAD=NATIVE                00030000
FILE=E$ST LOAD=NATIVE                00030000
FILE=E$BR LOAD=NATIVE                00030000
FILE=E$CU LOAD=NATIVE                00030000
FILE=E$IN LOAD=NATIVE                00030000
FILE=E$MF LOAD=NATIVE                00030000
FILE=E$PD LOAD=NATIVE                00030000
FILE=E$PG LOAD=NATIVE                00030000
FILE=E$PO LOAD=NATIVE                00030000
FILE=E$RG LOAD=NATIVE                00030000
FILE=E$SU LOAD=NATIVE                00030000
FILE=E$VS LOAD=NATIVE                00030000
END OF CSUAUX FILE RECORDS.
NO ERRORS ENCOUNTERED IN THE CSUAUX FILE.
BEGINNING THE LOAD FUNCTION.( E$IL )
E$ILINKPATH=E$INLK01
E$ILALL.END.
E$IL LOADED SUCCESSFULLY - COUNT =          217( E$MB )
E$MLINKPATH=E$BRLK01
E$MBALL.END.
E$MB LOADED SUCCESSFULLY - COUNT =          29( E$ML )
E$MLLINKPATH=E$MFLK01
E$MLALL.END.
E$ML LOADED SUCCESSFULLY - COUNT =          78( E$PL )
E$PLLINKPATH=E$POLK01
E$PLALL.END.
E$PL LOADED SUCCESSFULLY - COUNT =          122
E$PL - SECONDARY LINKS INSERTED SUCCESSFULLY - COUNT =          26( E$SK )
E$SKLINKPATH=E$PDLK03
E$SKALL.END.
E$SK LOADED SUCCESSFULLY - COUNT =          2,628( E$ST )
E$STLINKPATH=E$PDLK01
E$STALL.END.
E$ST LOADED SUCCESSFULLY - COUNT =          67
E$ST - SECONDARY LINKS INSERTED SUCCESSFULLY - COUNT =          67

( E$BR )
E$BRALL.END.OUT-OF-BLOCK SYNONYM RECORD COUNT =          0
E$BR LOADED SUCCESSFULLY - COUNT =          39
( E$CU )
E$CUALL.END.OUT-OF-BLOCK SYNONYM RECORD COUNT =          0
E$CU LOADED SUCCESSFULLY - COUNT =          43( E$IN )
E$INALL.END.OUT-OF-BLOCK SYNONYM RECORD COUNT =          0
E$IN LOADED SUCCESSFULLY - COUNT =          96( E$MF )
E$MFALL.END.OUT-OF-BLOCK SYNONYM RECORD COUNT =          0
E$MF LOADED SUCCESSFULLY - COUNT =          29( E$PD )
E$PDALL.END.OUT-OF-BLOCK SYNONYM RECORD COUNT =          0
E$PD LOADED SUCCESSFULLY - COUNT =          88( E$PG )
E$PGALL.END.OUT-OF-BLOCK SYNONYM RECORD COUNT =          0
E$PG LOADED SUCCESSFULLY - COUNT =          6( E$PO )
E$POALL.END.OUT-OF-BLOCK SYNONYM RECORD COUNT =          0
E$PO LOADED SUCCESSFULLY - COUNT =          26( E$RG )
E$RGALL.END.OUT-OF-BLOCK SYNONYM RECORD COUNT =          0
E$RG LOADED SUCCESSFULLY - COUNT =          10( E$SU )
E$SUALL.END.OUT-OF-BLOCK SYNONYM RECORD COUNT =          0
E$SU LOADED SUCCESSFULLY - COUNT =          14( E$VS )
E$VSALL.END.OUT-OF-BLOCK SYNONYM RECORD COUNT =          0
E$VS LOADED SUCCESSFULLY - COUNT =          191
LOAD FUNCTION COMPLETE.

```

Unloading, changing, and loading files

You can unload and reload to make changes to the structure of some files. In addition to changing the structure of the file, you also gain the same benefits as when you unload and reload them; that is, you repair broken linkpath chains, minimize the number of out-of-block synonyms, optimize access along primary linkpaths, and reorganize tree structures.

In this example, you are making three changes to the primary file E\$PD:

- ◆ Decreasing the size of element E\$PDPDPRCE from nine to seven bytes by removing the two zoned decimals from the front of the element.
- ◆ Adding the 20-character element E\$PDDES2 to the end of the record.
- ◆ Increasing the size of element E\$PDPGRP from two to 12 bytes by adding ten characters to the front of it.

You are making one change to the related file E\$ST: adding a 20-character element, E\$STCOMM, to the end of it.

You are making one change to the primary file E\$PO: increasing its size from 1177 to 1200 total logical records.

You are making one change to the primary file E\$PL: increasing its size from 902 to 950 total logical records.

To make these changes, you perform the same four steps as in the first example where you unloaded all files: depopulate, unload, load, and populate. However, in this example, you add another step after the Load function: you insert linkpath information into the E\$SU file. Thus, you execute five functions. The following UCL and control statements show each step in OS/390.

VSE

In VSE, you need to include other run control statements to unload and load, like the RECFORM statement.

Depopulating files

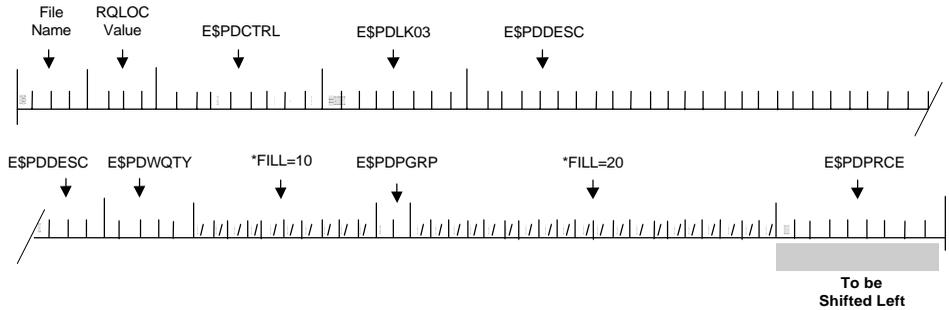
The first step is to depopulate the secondary keys in the files E\$PD and E\$PL. You do not need to depopulate the other two files because they have no secondary keys. The UCL for the **Depopulate** function follows:
(mono)

```
CONTROL(BEGIN)
*
  ENV-DESC (BURRYENN)
  SCHEMA   (BURRYOLD)
  LIST (ALL)
    DATA-FORMAT (HEX CHAR)
  DIAGNOSTICS (EXTENDED)
*
FUNCTION(DEPOPULATE)
  STATISTICS (ALL)
  FILE (E$PD)
  FILE (E$PL)
*
CONTROL(END)
```

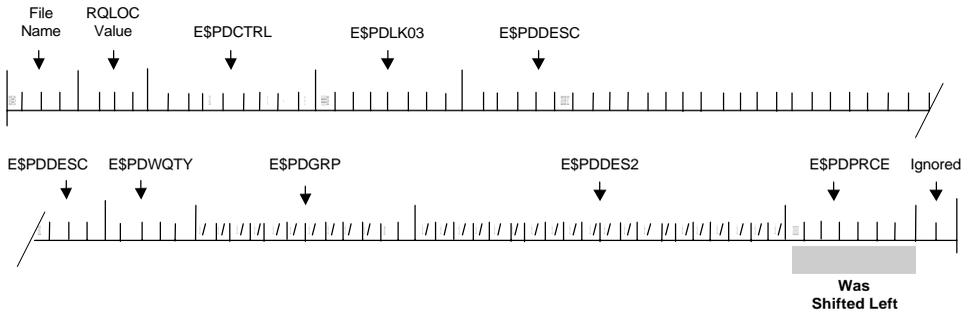
Unloading files

The second step is to unload the four files.

1. To unload E\$PD, you complete the following steps:
 - a. Clear linkpaths LK01 and LK02. You do not clear linkpath LK03 because it connects to a file that is not unloaded, E\$SK. To clear linkpaths LK01 and LK02 implicitly, you do not include them in the element list. Although not shown in this example, you could have included the linkpaths in the element list and coded them in the BLANK-LINKS parameter to clear the linkpaths explicitly.
 - b. Increase the size of the E\$PDPGRP element by adding *FILL=10 to the element list.
 - c. Add the element E\$PDDES2 by adding *FILL=20 to the element list.
 - d. Delete the two zoned decimals from the E\$PDPRCE element by coding exit program USER20 or USER30. Use an exit program to shift the remaining seven zoned decimals in the element to the left. They cover up the first two decimals which essentially deletes them. The following figure shows what you would see in the OUTFILE data record before your exit program:



The following figure shows what your OUTFILE looks like after your exit program.



You must use an exit program because you cannot use *FILL to delete elements in the Version 2 Load function. In the Version 1 Load function, you could code *FILL=02 instead of the exit program.

2. To unload E\$PO, complete the following steps:
 - a. Code ELEMENT (ALL) because you are not changing any elements.
 - b. Since you coded ALL in the element list, code E\$POLK01 in the BLANK-LINKS statement.
3. To unload E\$ST, code *FILL=20 to add the E\$STCOMM element.
4. To unload E\$PL, you can list the record codes separately or as a single element list. This example shows them as a single element list with ALL elements coded.

The following CSIPARM file, auxiliary file, run control, and file control statements illustrate these steps:

```

//CSIPARM DD *                                CSIPARM INPUT
DIRECTORY=(SCHEMA=CSTASCHM,ENVDESC=CSTANONE),
REALM=(SCHEMA=BURRYOLD,ENVDESC=BURRYENN),
END.
/*
//*
//CSUAUX DD *                                AUXILIARY INPUT FILE
FILE=E$PL LOAD=NATIVE
FILE=E$ST LOAD=NATIVE
FILE=E$PD LOAD=NATIVE
FILE=E$PO LOAD=NATIVE
/*
//*
//SYSIN DD *                                RUN CONTROL RECORDS
RELATED:E$PLE$STEND.
PRIMARY:E$PDE$POEND.
NEW-SCHEMA=BURRYSCH,NEW-ENVDESC=BURRYENN
SORTNAME=SORT
DUMP=YES
/*
//*
//PARM DD *                                FILE CONTROL RECORDS
E$PLLINKPATH=E$POLK01
E$PLALL.END.
E$STLINKPATH=E$PDLK01
E$STESTASSME$STQNTYE$STCOMP*FILL=20END.
E$PDE$PDCTRLE$PDLK03E$PDESCE$PDWQTY*FILL=10E$PDPGRPE$PDPREEND.
E$POALL.END.
E$POBLANK-LINKS=LK01END.
/*

```

Loading files

The third step is to load the four files.

1. Load E\$PD using the same element list that you used in the Unload function except you leave out the *FILL entries and add the element E\$PDES2.
2. Load E\$PO by coding ALL in the element list.
3. Load E\$ST by replacing *FILL with the element E\$STCOMM.
4. Load E\$PL by coding ALL in the element list.

The following input statements illustrate these steps:

```
//CSIPARM DD *                                CSIPARM INPUT
DIRECTORY=( SCHEMA=CSTASCHM, ENVDESC=CSTANONE ),
REALM=( SCHEMA=BURRYSCH, ENVDESC=BURRYENN ),
END.
/*
/*
//CSUAUX DD *                                AUXILIARY INPUT FILE
FILE=E$PL LOAD=NATIVE
FILE=E$ST LOAD=NATIVE
FILE=E$PD LOAD=NATIVE
FILE=E$PO LOAD=NATIVE
/*
/*
/*
//SYSIN DD *                                RUN CONTROL AND
                                           FILE CONTROL RECORDS
RELATED:E$PLE$STEND.
PRIMARY:E$PDE$POEND.
SCHEMA=BURRYSCH
SORTNAME=SORT
E$PLLINKPATH=E$POLK01
E$PLALL.END.
E$STLINKPATH=E$PDLK01
E$STES$STASSME$STQNTYE$STCOMPE$STCOMMEND.
E$PDE$PDCTRLE$PDLK03E$PDESCE$PDWQTYE$PDPGRPE$PDES2E$PDPREEND.
E$POALL.END.
/*
```

Inserting linkpath data for a file that was not loaded

The fourth step is to insert linkpath data. When you unload and load the file E\$PL, linkpath information is created for the linkpaths E\$POLK01 and E\$\$SULK01. When you load the file E\$PO, the Load function inserts the linkpath information for E\$POLK01. Since you did not load the file E\$SU, you must use the Insert Linkpath function to insert the information for E\$\$SULK01. The following CSIPARM file, auxiliary file, and run control statements illustrate this step:

```
//CSIPARM DD *                                CSIPARM INPUT
DIRECTORY=(SCHEMA=CSTASCHM,ENVDESC=CSTANONE),
REALM=(SCHEMA=BURRYSCH,ENVDESC=BURRYENN),
END.
/*
/*
//CSUAUX DD *                                AUXILIARY INPUT FILE
FILE=E$PL LOAD=NATIVE
FILE=E$ST LOAD=NATIVE
FILE=E$PD LOAD=NATIVE
FILE=E$PO LOAD=NATIVE
/*
/*
//SYSIN DD *                                RUN CONTROL RECORDS
INSERT FILES=(E$SU.)                            X
,CLEARLKS=(E$$SULK01.)                          X
,END.
/*
```

Populating files

The fifth step is to populate the secondary keys for the files E\$PD and E\$PL. You do not need to populate the other two files because they had no secondary keys. The UCL for the **Sorted-Populate** function follows:

```
CONTROL(BEGIN)
*
  ENV-DESC (BURRYENN)
  SCHEMA   (BURRYSCH)
  LIST (ALL)
  DATA-FORMAT (HEX CHAR)
  DIAGNOSTICS (EXTENDED)
*
FUNCTION (SORTED-
POPULATE)
  STATISTICS (ALL)
  FILE (E$PD)
  FILE (E$PL)
*
CONTROL(END)
```

Sample listing

The following listing shows the output you receive as a result of the sample UCL and statements:

```

TTTTTTTT  IIIIII  SSSSSSS
TTTTTTTT  IIIIII  SSSSSSSS
TT        II     SS    SS
TT        II     SS
TT        II     SSSSSSS
TT        II     SSSSSSSS
TT        II     SS    SS
TT        II     SS    SS
TT        IIIIII SSSSSSSS
TT        IIIIII SSSSSS
DDDDDDDD  BBBB BBBB AAA
DDDDDDDD  BBBB BBBB AAAAA
DD  DD  BB  BB  AA  AA
DD  DD  BB  BB  AA  AA
DD  DD  BBBB BBBB AA  AA
DD  DD  BBBB BBBB AAAAAAAA
DD  DD  BB  BB  AAAAAAAA
DD  DD  BB  BB  AA  AA
DDDDDDDD  BBBB BBBB AA  AA
DDDDDDDD  BBBB BBBB AA  AA
UU  UU  TTTTTT  IIIIII  LL      IIIIII  TTTTTT  IIIIII  EEEEEEEE  SSSSSS
UU  UU  TTTTTT  IIIIII  LL      IIIIII  TTTTTT  IIIIII  EEEEEEEE  SSSSSSSS
UU  UU  TT      II     LL      II     TT      II     EE      SS    SS
UU  UU  TT      II     LL      II     TT      II     EE      SS
UU  UU  TT      II     LL      II     TT      II     EEEEE  SSSSSSS
UU  UU  TT      II     LL      II     TT      II     EEEEE  SSSSSSS
UU  UU  TT      II     LL      II     TT      II     EE      SS    SS
UUUUUUUU  TT      IIIIII  LLLLLLLL  IIIIII  TT      IIIIII  EEEEEEEE  SSSSSSSS
UUUUUUUU  TT      IIIIII  LLLLLLLL  IIIIII  TT      IIIIII  EEEEEEEE  SSSSSS(ep)

                F U N C T I O N :  U N L O A D
                E N V I R O N M E N T :

RELATED: E$PLE$STEND.
PRIMARY: E$PDE$POEND.
NEW-SCHEMA=BURRYSCH,NEW-ENVDESC=BURRYENN
SORTNAME=SORT
DUMP=YES

CSUAUX FILE RECORDS:
FILE=E$PL LOAD=NATIVE
FILE=E$ST LOAD=NATIVE
FILE=E$PD LOAD=NATIVE
FILE=E$PO LOAD=NATIVE
END OF CSUAUX FILE RECORDS.
NO ERRORS ENCOUNTERED IN THE CSUAUX FILE.
CSUAUX FILE RECORDS:
FILE=E$PL LOAD=NATIVE
FILE=E$ST LOAD=NATIVE
FILE=E$PD LOAD=NATIVE
FILE=E$PO LOAD=NATIVE
END OF CSUAUX FILE RECORDS.
NO ERRORS ENCOUNTERED IN THE CSUAUX FILE.

                BEGINNING THE UNLOAD FUNCTION.                E$PLLINKPATH=E$POLK01
                E$PLALL.END.                E$PL UNLOADED SUCCESSFULLY
                NUMBER OF RECORDS UNLOADED =                122

                E$STLINKPATH=E$PDLK01
                E$STES$STASME$STQNTY$STCOMP*FILL=2OEND.
                E$ST UNLOADED SUCCESSFULLY
                NUMBER OF RECORDS UNLOADED =                67
                E$PDE$PDCTRL$E$PDLK03E$PDDESCE$PDWQTY*FILL=10E$PDPGRPE$PDRCEEND.
                E$PD UNLOADED SUCCESSFULLY
                NUMBER OF RECORDS UNLOADED =                88

                E$POALL.END.
                E$POBLANK-LINKS=LK01END.
                E$PO UNLOADED SUCCESSFULLY
                NUMBER OF RECORDS UNLOADED =                26

UNLOAD FUNCTION COMPLETE.
    
```

```

TTTTTTTT IIIIII SSSSSS
TTTTTTTT IIIIII SSSSSSSS
TT II SS SS
TT II SS
TT II SSSSSSS
TT II SSSSSSS
TT II SS SS
TT II SS SS
TT IIIIII SSSSSSSS
TT IIIIII SSSSSS
DDDDDDDD BBBB BBBB AAA
DDDDDDDD BBBB BBBB AAAAA
DD DD BB BB AA AA
DD DD BB BB AA AA
DD DD BBBB BB AA AA
DD DD BBBB BB AAAAAAAA
DD DD BB BB AAAAAAAA
DD DD BB BB AA AA
DDDDDDDD BBBB BBBB AA AA
DDDDDDDD BBBB BBBB AA AA
UU UU TTTTTT IIIIII LL IIIIII TTTTTT IIIIII EEEEEEE SSSSSS
UU UU TTTTTT IIIIII LL IIIIII TTTTTT IIIIII EEEEEEE SSSSSSSS
UU UU TT II LL II TT II EE SS SS
UU UU TT II LL II TT II EEEEE SSSSSS
UU UU TT II LL II TT II EEEEE SSSSSSS
UU UU TT II LL II TT II EE SS
UU UU TT II LL II TT II EE SS SS
UUUUUUUU TT IIIIII LLLLLLLL IIIIII TT IIIIII EEEEEEE SSSSSSSS
UUUUUU TT IIIIII LLLLLLLL IIIIII TT IIIIII EEEEEEE SSSSSS
F U N C T I O N : L O A D
E N V I R O N M E N T :RELATED:ES$PLE$STEND.
PRIMARY:ES$PDE$POEND.
SCHEMA-BURRYSCH
SORTNAME-SORT

CSUAUX FILE RECORDS:
FILE-ESPL LOAD-NATIVE
FILE-ESST LOAD-NATIVE
FILE-ESPD LOAD-NATIVE
FILE-ESPO LOAD-NATIVE
END OF CSUAUX FILE RECORDS.
NO ERRORS ENCOUNTERED IN THE CSUAUX FILE.
BEGINNING THE LOAD FUNCTION.( ESPL )
ESPLLINKPATH-ESPOLK01
ESPLALL.END.
ESPL LOADED SUCCESSFULLY - COUNT - 122
ESPL - SECONDARY LINKS INSERTED SUCCESSFULLY - COUNT - 28( ESST )
ESSTLINKPATH-ESPOLK01
ESSTESASSME$STONTYE$STCOMPE$STCOMMENT.
ESST LOADED SUCCESSFULLY - COUNT - 67
ESPL - SECONDARY LINKS INSERTED SUCCESSFULLY - COUNT - 67( ESPD )
ES$PDE$POCTRLE$PDLK03$PDESCE$PDWQTYE$PDPGRPE$PDESZE$PDPREEND.
OUT-OF-BLOCK SYNONYM RECORD - COUNT - 0
ESPD - LOADED SUCCESSFULLY - COUNT - 88( ESP0 )
ESPOALL.END.OUT-OF-BLOCK SYNONYM RECORD COUNT - 0
ESPO LOADED SUCCESSFULLY - COUNT - 26LOAD FUNCTION COMPLETE.

```

```

TTTTTTT  IIIIII  SSSSSSS
TTTTTTT  IIIIII  SSSSSSSSS
TT       II     SS     SS
TT       II     SS
TT       II     SSSSSSS
TT       II     SSSSSSS
TT       II     SS     SS
TT       II     SSSSSSS
TT       IIIIII SSSSSSS
TT       IIIIII SSSSSSS
DDDDDDDD BBBB BBBB AAA
DDDDDDDD BBBB BBBB AAAAA
DD  DD  BB  BB  AA  AA
DD  DD  BB  BB  AA  AA
DD  DD  BBBB BB  AA  AA
DD  DD  BBBB BB  AAAAAAAA
DD  DD  BB  BB  AAAAAAAA
DD  DD  BB  BB  AA  AA
DDDDDDDD BBBB BB  AA  AA
DDDDDDDD BBBB BB  AA  AA
UU  UU  TTTTTT  IIIIII  LL      IIIIII  TTTTTTT  IIIIII  EEEEEEEE  SSSSSSS
UU  UU  TTTTTTT  IIIIII  LL      IIIIII  TTTTTTT  IIIIII  EEEEEEEE  SSSSSSSS
UU  UU  TT       II     LL      II     TT       II     EE     SS     SS
UU  UU  TT       II     LL      II     TT       II     EEEEE  SSSSSSS
UU  UU  TT       II     LL      II     TT       II     EEEEE  SSSSSSS
UU  UU  TT       II     LL      II     TT       II     EE     SS     SS
UUUUUUUU TT       IIIIII  LLLLLLLL  IIIIII  TT       IIIIII  EEEEEEEE  SSSSSSSS
UUUUUUU  TT       IIIIII  LLLLLLLL  IIIIII  TT       IIIIII  EEEEEEEE  SSSSSSS
F U N C T I O N :   I N S E R T   L I N K P A T H

E N V I R O N M E N T :

INSERT FILES=(E$SU.)           X
,CLEARLKS=(E$SULK01.)         X
,END. CSUAUX FILE RECORDS:

FILE=E$PL LOAD=NATIVE
FILE=E$ST LOAD=NATIVE
FILE=E$PD LOAD=NATIVE
FILE=E$PO LOAD=NATIVE

END OF CSUAUX FILE RECORDS.
NO ERRORS ENCOUNTERED IN THE CSUAUX FILE.UTL-000 ** NO EDIT ERRORS **

UTL-000 ** BEGINNING THE INSERT LINKPATH FUNCTION.

UTL-075 FILE NOT SPECIFIED TO BE UPDATED; SKIPPING THIS FILE **E$PD**

UTL-075 FILE NOT SPECIFIED TO BE UPDATED; SKIPPING THIS FILE **E$PO**UTL-092 COUNT OF RECORDS UPDATED *****09**

UTL-093 PROCESSING COMPLETE FOR FILE **E$SU**UTL-000 ** INSERT LINKPATH FUNCTION COMPLETE.

```

12

Coding the Print function

Use the Print function when you want to print records from a database file. For related files, you may not print linkpaths. For primary files, you may print linkpaths, but you may not print root fields.

Coding the UCL for the Print function

After you code the control section as shown in “[Coding the control section](#)” on page 57, you can code the Print function as shown in the following format. For UCL examples, see “[Print examples](#)” on page 377.

FUNCTION (PRINT)

[STANDARD-EXIT (*exit-name*)]

FILE ({ ALL
 file-mode }) ...

[OPEN - MODE ([READ
 IUPD
 SUPD
 EUPD])]

[CLOSE ([NO
 YES])]

<p>QUALIFIER ($\left[\begin{array}{l} \text{DIRECT} \\ \text{SERIAL} \\ \text{SEQUENTIAL} \end{array} \right]$)</p> <p>[LINKPATH (<i>access-linkpath</i>)]</p> <p>$\left[\text{KEY} \left(\left\{ \begin{array}{l} \text{D' dec-string' } \\ \text{X' hex-string' } \\ \text{C' char-string' } \end{array} \right\} \right) \right]$</p> <p>[RRN (<i>record-rrn</i>)]</p> <p>$\left[\text{RRN-RANGE} \left(\left[\begin{array}{l} \textit{low-rrn} \\ \text{--high-rrn} \\ \textit{low-rrn --high-rrn} \end{array} \right] \right) \right]$</p> <p>$\left[\text{MAXIMUM} \left(\left[\begin{array}{l} \textit{b} \\ \textit{record-count} \end{array} \right] \right) \right]$</p> <p>[CRITERIA (<i>element</i>₁[<i>element</i>₂ ,..., <i>element</i>_n] .<i>operator</i> . <i>datavalue</i>₁ [. <i>datavalue</i>₂ ... <i>datavalue</i>_n]end.)</p>

<p>RECORD ($\left\{ \begin{array}{l} \text{ALL} \\ \textit{record-code} \end{array} \right\}$)</p> <p style="text-align: right;">...</p> <p>ELEMENT ($\left\{ \begin{array}{l} \text{ALL} \\ \textit{element-list} \end{array} \right\}$)</p>

FUNCTION (PRINT)

Description *Required.* Invokes the Print function.

STANDARD-EXIT (*exit-name*)

Description *Optional.* Names an exit program you want to invoke.

Format 1–8 alphanumeric characters. The first character must be alphabetic.

Considerations

- ◆ If you code this statement, you must put it before the FILE statements.
 - ◆ You must make your exit program available to be loaded by the function. That is, it must reside in your execution library.
 - ◆ Only one exit at a time resides in memory. If you code a new exit name in a subsequent function, the function deletes the current exit before it loads the new exit.
-

FILE ({ **ALL
file-name }) ...**

Description *Required.* Indicates the file you want printed.

Format 4 alphanumeric characters. The first character must be alphabetic.

Considerations

- ◆ If you code FILE (ALL), the function prints the files in alphabetical order with primary files first and then the related files. A message stating you cannot print index files precedes the primary files.
- ◆ If you code FILE (ALL), the function prints all your PDM files in the schema. To print Directory files, code FILE (*file-name*).
- ◆ You cannot code FILE (*file-name-list*).

OPEN - MODE (

READ
IUPD
SUPD
EUPD

)

- Description** *Optional.* Indicates how you want the function to open the file.
- Default** READ
- Options** READ The function opens the file for read-only access.
- IUPD The function opens the file with intent to update.
- SUPD The function opens the file for shared update.
- EUPD The function opens the file for exclusive update.

Consideration If you code the OPEN-MODE statement, you must put it before any RECORD statements.

CLOSE (

NO
YES

)

- Description** *Optional.* Indicates whether you want the file explicitly closed after printing.
- Default** YES

QUALIFIER ($\left[\begin{array}{l} \text{DIRECT} \\ \text{SERIAL} \\ \text{SEQUENTIAL} \end{array} \right]$)

Description	<i>Optional.</i> Indicates the access mode you want the function to use to print the current file.	
Default	SERIAL	
Options	DIRECT	Reads a specific record either by RRN or by key. For more information, see the second consideration, below.
	SERIAL	Accesses the file serially without regard to chain sequence.
	SEQUENTIAL	Accesses the related file sequentially by a specific linkpath.

Considerations

- ◆ Do not code QUALIFIER (SEQUENTIAL) for a primary file.
- ◆ As shown in the format, QUALIFIER has three options. Depending on the type of file (primary (P) or related (R)) and the access mode you code, the following statements are either required (r), optional (o), or invalid (i):

QUALIFIER:	DIRECT		SERIAL		SEQUENTIAL	
File Type:	P	R	P	R	P	R
RRN	i	r	i	i	i	i
LINKPATH	i	i	i	i	i	r
KEY	r	i	i	i	i	o
RRN-RANGE	i	i	o	o	i	i
MAXIMUM	i	i	o	o	i	o

- ◆ If you code the QUALIFIER statement, you must put it before the RECORD statements.
- ◆ If you code QUALIFIER (DIRECT), the function prints only one record.
- ◆ If you code QUALIFIER (SEQUENTIAL) and the LINKPATH and KEY statements, the function prints only the chain containing the key you coded.
- ◆ If you code QUALIFIER (SEQUENTIAL) and the LINKPATH statement, but not the KEY statement, the function prints all the chains associated with the linkpath.

LINKPATH (*access-linkpath*)

Restrictions

- ◆ Use this statement only for sequential access to related files.
- ◆ You can only use this statement after a QUALIFIER statement.

Description *Optional.* Determines the access linkpath for a related file you are reading sequentially.

Format *fffLKxx*, where *fff* is a primary file linked to a related file through linkpath *LKxx*

Considerations

- ◆ The access linkpath may exist in either the base or the redefined portion of a coded record.
- ◆ If you code RECORD (ALL) with an access linkpath, that linkpath must exist in all records.
- ◆ You cannot code LINKPATH (*linkpath-list*).

```
KEY ( { D' ec-string'
      X' hex-string'
      C' char-string' } )
```

Restriction You can only use this statement after a QUALIFIER statement.

Description *Optional.* Indicates the key you want the function to use for direct access to a primary file or sequential access to a related file.

Format

<i>D' dec-string</i>	A decimal string of 1 to 256 bytes preceded by a D and surrounded by single quotes. This string is a 1–256 byte key that must match the actual key length.
<i>X' hex-string</i>	A hexadecimal string of 2 to 512 bytes preceded by an X and surrounded by single quotes. This string is a 1–256 byte key. The length must be an even number and twice the actual key length.
<i>C' char-string</i>	A character string of 1 to 256 bytes preceded by a C and surrounded by single quotes. This string is a 1–256 byte key and must match the actual key length.

Considerations

- ◆ If a character string contains more than one quote, you must code two quotes for each actual quote. For example, you must code ABCD'EF'G as C'ABCD"EF"G'.
- ◆ Any key you code must be the correct length for the file.

RRN (*record-rrn*)

Restrictions

- ◆ Use this statement only for direct access to related files.
- ◆ You can only use this statement after a QUALIFIER statement.

Description *Optional.* Selects a relative record number in a non-KSDS related file that you want the function to read directly.

Format 1–9 decimal characters

Considerations

- ◆ You must code a *record-rrn* if you are printing a related file and you coded QUALIFIER (DIRECT).
- ◆ You must code a *record-rrn* that is within the boundary of the file.
- ◆ Do not code FILE (ALL) with the RRN statement if any files are key-sequenced data sets.

```
RRN-RANGE (  $\left[ \begin{array}{l} \textit{low-rrn} \\ - \textit{high-rrn} \\ \textit{low-rrn} - \textit{high-rrn} \end{array} \right]$  )
```

Restrictions

- ◆ Do not use with key-sequenced data sets.
- ◆ You can only use this statement after a QUALIFIER statement.

Description *Optional.* Indicates a range of relative record numbers you want retrieved. The function does not retrieve records outside the range you code.

Format 1–9 decimal characters for each *rrn*

Options

<i>low-rrn</i>	Retrieves records having RRNs from <i>low-rrn</i> through the end of the file.
<i>-high-rrn</i>	Retrieves records having RRNs from the beginning of the file to <i>high-rrn</i> .
<i>low-rrn-high-rrn</i>	Retrieves records having RRNs from <i>low-rrn</i> through <i>high-rrn</i> .

Considerations

- ◆ Do not code FILE (ALL) with the RRN-RANGE statement if any files are key-sequenced data sets.
- ◆ If the *low-rrn* you code is not a valid data record, the function accesses the first data record with a higher RRN than the one you coded.
- ◆ If you are printing a related file and the RRN you code is in the middle of a chain, the function does not print prior records on that chain.
- ◆ Do not code the RRN-RANGE statement if you coded QUALIFIER (DIRECT) or QUALIFIER (SEQUENTIAL).

MAXIMUM ($\left[\begin{array}{l} \underline{b} \\ \text{record-count} \end{array} \right]$)

Restriction You can only use this statement after a QUALIFIER statement.

Description *Optional.* Determines the maximum number of records to print.

Default *b*

Format 1–9 decimal characters

Considerations

- ◆ If you code MAXIMUM (*b*), the function prints all records.
- ◆ To code MAXIMUM, you must have already coded QUALIFIER.
- ◆ If you code the MAXIMUM statement, you must put it must before the RECORD statements.
- ◆ The function includes in this count only valid data records that pass all other selection criteria. For example, if you specify an argument (via the CRITERIA statement), all data records must first pass the argument validation before the function adds them to the maximum record counter.
- ◆ If you code a number that exceeds the total number of records in the file, the function stops processing at the end of the file.
- ◆ If you code a value that exceeds the total number of records in this file, the function stops at the end of the file.

**CRITERIA (*element1*[,*element2*,...,*elementn*].*operator*.*datavalue1*
[.*datavalue2*...*datavaluen*]END.)**

Restriction You can only use this statement after a QUALIFIER statement.

Description *Optional.* Establishes an argument string to select the records you want printed.

Considerations

- ◆ If you code this statement, put it before the RECORD statements.
- ◆ You can code any number of spaces before the element list, after END., and on either side of the separating commas.
- ◆ If you do not code END, the function considers the rest of the program as data.

Format for element

One or more 8 alphanumeric character element names. You must make the first character in each name alphabetic and separate the names with commas.

Considerations

- ◆ If you code an element name in the argument, you must also code it in the ELEMENT statement unless you code ELEMENT (ALL).
- ◆ You cannot code a null element list in the criteria argument.
- ◆ If you name an element in the criteria argument, it must be in all the records you want printed from the file.

Format for operator

.EQ. Equal
 .NE. Not equal
 .GT. Greater than
 .LT. Less than
 .GE. Greater than or equal to
 .LE. Less than or equal to

Consideration You must code a period before and after the Boolean operator. Only one operator may be specified.

Format for datavalue

Any valid EBCDIC value. You can code the actual hexadecimal representation of any value of any data type in your UCL statement. You must put a period before data value and END. after it.

Considerations

- ◆ You must make data values the same length as the element lengths in the element list.
- ◆ Do not put spaces between data values.
- ◆ Your data may cross input line boundaries if necessary. You must stop in column 72 and continue on the next line in column 1. (If you put data in columns 73–80, it is lost.)

RECORD { ALL
[*record-code*] }

Description *Optional.* Indicates the records you want printed.

Default ALL

Format 2 alphanumeric characters

Considerations

- ◆ If you code this statement, you must code the ELEMENT statement.
- ◆ Do not code RECORD (ALL) if you intend to code the element list with redefined element names for a coded related file.
- ◆ For primary files, always code RECORD (ALL).
- ◆ The record code you indicate must be in the file to be printed.
- ◆ If you code multiple RECORD statements, do not code ALL with a specific record code.
- ◆ If you code FILE (ALL), do not code RECORD (*record-code*).
- ◆ If you code RECORD (), the function prints no records.

ELEMENT ({ ALL
element-list })

Restriction *Required if you code the RECORD statement.*

Description Indicates the data elements you want printed.

Default ALL

Format Element names must be 8 alphanumeric characters. The first character must be alphabetic. Separate names with commas.

Considerations

- ◆ If you code FILE (ALL), do not code ELEMENT (*element-list*).
- ◆ You can use ELEMENT (ALL) in conjunction with an *element-list*. For example, the following is correct:

```
RECORD ( 01 )
ELEMENT ( ELEMENT1 , ELEMENT2 )
RECORD ( 02 )
ELEMENT ( ALL )
```

- ◆ The key element for a related file is the one associated with the access linkpath you coded.
- ◆ Additional constraints apply in the following situations:

In this context:

ELEMENT (*element-list*) must conform to these rules:

FILE(<i>primary-file</i>)	Do not include the root element in <i>element-list</i> .
FILE(<i>related-file</i>) no access-linkpath	Do not include linkpaths in <i>element-list</i> .
FILE(<i>related-file</i>)	Do not include linkpaths in <i>element-list</i> . LINKPATH(<i>ffffLKxx</i>)
FILE(<i>coded-file</i>) no access-linkpath	First entry in <i>element-list</i> must be <i>ffffCODE</i> where <i>ffff</i> is the coded file name. Do not include linkpaths in <i>element-list</i> .
FILE(<i>coded-file</i>) LINKPATH(<i>ffffLKxx</i>)	First entry in <i>element-list</i> must be <i>ffffCODE</i> where <i>ffff</i> is the coded file name. Do not include linkpaths in <i>element-list</i> .
RECORD(<i>record-code</i>)	Names appearing in the element list must exist in record code.
FILE(<i>coded-file</i>) RECORD(ALL)	Do not include redefined element names in <i>element-list</i> .

Writing exit programs

You can use the exit point from the Print function to examine the records you are printing. The function invokes your exit program after it prints the record. Therefore, there are no return codes from this exit. Your program can collect statistics or data on the records it is passed, but it cannot modify, delete or add records.

For information on how your exit programs are loaded, how they operate, the languages you can use to write them, and the register conventions you must follow, see “[Inserting exit programs into functions](#)” on page 49. For example, you must code the parameter list addresses in register 1. For a description of parameter list addresses, see the following table.

Parameter	Data type	Contents before exit (passed to exit program)	Contents after exit (passed from exit program)
Record	<i>n</i> bytes of data	Data record	Must be unchanged
Function Name	8 bytes character	PRINT bbb	Must be unchanged

If your exit program changes anything it is not authorized to change, the results are unpredictable.

Print examples

Example 1 The following example prints the elements in the records of all files in the schema coded in the control section:

```
CONTROL (BEGIN)
  ENV-DESC (MYDESC)
  SCHEMA (MYSHEMA)
FUNCTION (PRINT)
  FILE (ALL)
  RECORD (ALL)
  ELEMENT (ALL)
CONTROL (END)
```

Example 2 This example prints data from the CUST file and invokes the exit program EXIT0001. You request the record in the CUST file by coding its key, the decimal string C'001234'. You request the file be opened for read-only and accessed directly. Since CUST is a primary file, the function prints all record codes, specifically the CUSTCTRL and CUSTNAME elements. The function reads the second file, PORD, sequentially by way of the CUSTLKPO linkpath. The function prints various elements in the HD and IT records.

```
CONTROL (BEGIN)
  ENV-DESC (MYDESC)
  SCHEMA (MYSHEMA)
FUNCTION (PRINT)
  STANDARD-EXIT (EXIT0001)
  FILE (CUST)
  OPEN-MODE (READ)
  QUALIFIER (DIRECT)
  KEY (C'001234')
  RECORD (ALL)
  ELEMENT (CUSTCTRL, CUSTNAME)
  FILE (PORD)
  QUALIFIER (SEQUENTIAL)
  LINKPATH (CUSTLKPO)
  RECORD (HD)
  ELEMENT (PORDCMNT, PORDDATE, PORDVEND, PORDCARR)
  RECORD (IT)
  ELEMENT (PORDITEM, PORDIQTY)
CONTROL (END)
```

Example 3 These following examples show sample input and output of the Print function:

```

CSUL0101I : COMMENCING COMMAND VALIDATION.
1 CONTROL(BEGIN)
2 *****
3 *
4 * PRINT EXAMPLE #1 DESCRIPTION
5 *
6 * OBJECTIVE: PRINT RECORDS FROM THE DIRECTORY.
7 *
8 * NOTES:
9 *
10 * 1. FROM THE C$-# FILE, PRINT THE SPECIFIED
11 * ELEMENTS FROM THE RECORDS ONLY IF C$-#NAME
12 * SATISFIES THE STATED CRITERIA.
13 *
14 * 2. MOVE SERIALLY THROUGH FILE C$-S, AND PRINT
15 * A MAXIMUM OF 10 RECORDS.
16 *
17 *
18 *****
19 ENV-DESC(CINDIREN)
20 SCHEMA(CINDIRSC)
21 DIAGNOSTICS(EXTENDED)
22 LIST()
23 HEADER(YES)
24 EXTENSION('PRINT EXAMPLE 1')
25 FUNCTION(PRINT)
26 FILE(C$-#)
27 CRITERIA (C$-#NAME.EQ.01 END.)
28 RECORD(ALL)
29 ELEMENT(C$-#CODE,
30 C$-#NAME,
31 C$-#ATM,
32 C$-#ATUI)
33 FILE(C$-S)
34 QUALIFIER(SERIAL)
35 MAXIMUM(10)
36 RECORD(ALL)
37 ELEMENT(ALL)
38 CONTROL(END)
CONTENTS OF SOURCE LINES OUTSIDE 1..72 MARGINS IGNORED.
0 SYNTAX ERRORS DETECTED.
38 COMMAND LINES READ.
1 CONTROL SECTIONS ANALYZED.
1 FUNCTION COMMANDS ANALYZED.

PRINT EXAMPLE 1

CSUL0102I : COMMENCING COMMAND EXECUTION.
CSUL0301I : COMMENCING CONTROL SECTION USING ENVIRONMENT DESCRIPTION CINDIREN AND SCHEMA CINDIRSC.
CSUL0302I : COMMENCING PRINT PROCESS.
CSUL0311I : COMMENCING PRINT AGAINST FILE C$-#.
REFER = 00000865
C$-#CODE C$-#NAME C$-#ATM C$-#ATUI
24 01 112511 DIR. MIGR. SL 2114 ->SL 2116
REFER = 00000925
C$-#CODE C$-#NAME C$-#ATM C$-#ATUI
16 01 120154 CSI-DBA
REFER = 0000093F
C$-#CODE C$-#NAME C$-#ATM C$-#ATUI
16 01 112512 DIR. MIGR. SL 2114 ->SL 2116
REFER = 00000993
C$-#CODE C$-#NAME C$-#ATM C$-#ATUI
16 01 150956 CSI-DBA
REFER = 00000B36
C$-#CODE C$-#NAME C$-#ATM C$-#ATUI
16 01 120342 CSI-DBA

```

```

REFER = 00000ECE
C$-#CODE      C$-#NAME          C$-#ATTM      C$-#ATUI
24            01                110310        CSI-DBA
REFER = 00001230
C$-#CODE      C$-#NAME          C$-#ATTM      C$-#ATUI
16            01                112518        DIR. MIGR. SL 2114 ->SL 2116
REFER = 000014F6
C$-#CODE      C$-#NAME          C$-#ATTM      C$-#ATUI
16            01                151526        CSI-DBA
REFER = 00001B0A
C$-#CODE      C$-#NAME          C$-#ATTM      C$-#ATUI
24            01                112535        DIR. MIGR. SL 2114 ->SL 2116
CSUL0349I : END-OF-FILE HAS BEEN ENCOUNTERED ON THE CURRENT FILE.
PRINT EXAMPLE 1
CSUL0321I : PRINT                PROCESSING AGAINST FILE C$-# TERMINATING NORMALLY.
PRINT EXAMPLE 1      CSUL0311I : COMMENCING PRINT                AGAINST FILE C$-S.
REFER = 00000001
DT /1213                >
REFER = 00000002
HD 2224
REFER = 00000003
DT 2224
REFER = 00000004
HD /i0709
REFER = 00000005
DT /i0709 / NONESLFB      / /
REFER = 0000001B
HD /1213
REFER = 00000059
DT 1213                / %R / !
REFER = 0000005A
DT 1213                / %/ /
REFER = 0000005B
DT /1612 /
REFER = 0000005C
DT /1612 /                ? )
CSUL0346I : THE SPECIFIED MAXIMUM NUMBER OF RECORDS FOR THE CURRENT FILE HAVE BEEN PROCESSED.
CSUL0321I : PRINT                PROCESSING AGAINST FILE C$-S TERMINATING NORMALLY.
CSUL0303I : PRINT                PROCESS TERMINATING.
CSUL0305I : CONTROL SECTION TERMINATING.
CSUL0306I : SUMMARY DATA FOR TERMINATING CONTROL SECTION :
CSUL0101I : COMMENCING COMMAND VALIDATION.
 1 CONTROL(BEGIN)
 2 *****
 3 *
 4 * PRINT EXAMPLE #2 DESCRIPTION
 5 *
 6 * OBJECTIVE: PRINT ELEMENTS FROM THE DATABASE FILES. *
 7 *
 8 *
 9 * NOTES:
10 *
11 * 1. FOR THE C$-T FILE, PRINT ELEMENTS FOR ALL
12 * RECORDS IN THE SPECIFIED LINKPATH CHAIN.
13 *
14 * 2. MOVE SERIALY THROUGH THE C$-D FILE AND:
15 *
16 * A. PRINT THE 2 SPECIFIED ELEMENTS FOR
17 * RECORD CODE 07 RECORDS WITHIN RRN-RANGE
18 * 5900-6810.
19 *
20 * B. PRINT ALL ELEMENTS FOR RECORD CODE 14
21 * RECORDS WITHIN RRN-RANGE 5900-6810.
22 *
23 *
24 *****
25 ENV-DESC(CINDIREN)
26 SCHEMA(CINDIRSC)
27 DIAGNOSTICS(EXTENDED)
28 LIST()
29 HEADER(YES)
30 EXTENSION('PRINT EXAMPLE 2')

```

```

31 FUNCTION(PRINT)
32 FILE(C$-T)
33 QUALIFIER(SEQUENTIAL)
34 LINKPATH(C$-#LKTT)
35 KEY('000000C0')
36 RECORD(ALL)
37 ELEMENT(ALL)
38 FILE(C$-D)
39 QUALIFIER(SERIAL)
40 RRN-RANGE(5900-6810)
41 RECORD(07)
42 ELEMENT(C$-DCODE,
43 C$-D07TP)
44 RECORD(14)
45 ELEMENT(ALL)
46 CONTROL(END)
CONTENTS OF SOURCE LINES OUTSIDE 1...72 MARGINS IGNORED.
0 SYNTAX ERRORS DETECTED.
46 COMMAND LINES READ.
1 CONTROL SECTIONS ANALYZED. 1 FUNCTION COMMANDS ANALYZED.PRINT EXAMPLE 2

CSUL0102I : COMMENCING COMMAND EXECUTION.
CSUL0301I : COMMENCING CONTROL SECTION USING ENVIRONMENT DESCRIPTION CINDIREN AND SCHEMA CINDIRSC.
CSUL0302I : COMMENCING PRINT PROCESS.
CSUL0311I : COMMENCING PRINT AGAINST FILE C$-T.
REFER = 000027F9
ST |P[ /CALCULATES AVERAGE, MAXIMUM, MINIMUM & TOTAL. 5
REFER = 000027F5
LT |P[ s c /STATISTICS IS A TERM USED WITH THE STATISTICS COLLECTION COMMANDS (WHEN 9 6
REFER = 000027F6
LT |P[ s c HCHANGES AND WHEN FINISHED). WHEN SPECIFIED, QUERY WILL CALCULATE 5 7
REFER = 000027F7
LT |P[ s c AND DISPLAY THE AVERAGE, MAXIMUM, MINIMUM AND TOTAL OF THE 6 8
REFER = 000027F8
LT |P[ s c /ASSOCIATED VALUE. 7 /
REFER = 000027FA
LT |P[ s c 4- 8 /
REFER = 000027FB
LT |P[ s c /WHEN FINISHED PRINT STATISTICS OF SALES /
CSUL0348I : END-OF-CHAIN HAS BEEN ENCOUNTERED ON THE CURRENT FILE AND SPECIFIED LINKPATH.
CSUL0321I : PRINT PROCESSING AGAINST FILE C$-T TERMINATING NORMALLY.
PRINT EXAMPLE 2 CSUL0311I : COMMENCING PRINT AGAINST FILE C$-D.
REFER = 0000171B
C$-DCODE C$-D07TP
07 BP TLFB N
REFER = 00001722
14 / s cB HC$-N,CSI-DIR-RPTR-NAME-
QUAL FIND DEFN FROM INPUT US NAME / /
REFER = 00001723
14 / s cB C$-#.NADF,CSI-DIR-NM-DEFN FIND CAT. CODE FROM US DEFN
REFER = 00001724
14 / s cB /C$-T,C$-#LKTT FIND US TEXT FROM US DEFN / /
o
o
o
C$-DCODE C$-D07TP
07 BP / /USRM N
REFER = 00001A6E
C$-DCODE C$-D07TP
07 BP SLFB N
CSUL0347I : THE SPECIFIED LAST RECORD FOR THE CURRENT FILE HAS BEEN PROCESSED.
CSUL0321I : PRINT PROCESSING AGAINST FILE C$-D TERMINATING NORMALLY.
CSUL0303I : PRINT PROCESS TERMINATING.
CSUL0305I : CONTROL SECTION TERMINATING.

```

13

Coding the Modify function

Coding the Modify function

Use the Modify function to update records in database files. You can update all or some database elements in each type of record. You can print records before, after, or both before and after you update them.

You can update all but the following elements:

- ◆ The ROOT field in a primary file
- ◆ Control keys in primary or related files
- ◆ Code element in a coded file
- ◆ Linkpaths in a related file

While you cannot update linkpaths in a related file, you can update linkpaths in a primary file. This enables you to unload and load a related file, but not the primary files with which it is associated. To do this, you use the Modify function to clear the linkpaths in the primary files as in “[Examples of Unload, Load, and Modify functions](#)” on page 191. The procedure involves the following three steps:

1. Using the [Version 1 Unload](#) function to unload the related file.
2. Using the Modify function to change the linkpath elements in the primary file(s) so they contain eight blanks.
3. Using the [Version 1 Load](#) function to reload the related file.

As the Load function processes the related file, it recreates the linkpath information and stores it in the blanked linkpath element(s). The UCL to clear the linkpaths is shown in the second example in “[Modify examples](#)” on page 394.

If you are using the [Version 2 Unload and Load](#) functions, you cannot use the Modify function to clear the linkpaths. You must use the [Insert Linkpath](#) function.

Coding the UCL for the Modify function

After you code the control section as shown in “[Coding the control section](#)” on page 57, you can code the Modify function as shown in the following format. For UCL examples, see “[Modify examples](#)” on page 394.

FUNCTION (MODIFY)

STANDARD-EXIT (*exit-name*)

FILE (*file-name*) ...

[OPEN - MODE ([SUPD]
[EUPD])]

[CLOSE ([NO]
[YES])]

QUALIFIER ({ SERIAL
DIRECT
SEQUENTIAL })

[RRN (*record-rrn*)]

[LINKPATH (*access-linkpath*)]

[KEY ({ D' *dec-string*' }
{ X' *hex-string*' })
{ C' *char-string*' })]

[MAXIMUM ([b]
[*record-count*])]

[CRITERIA (*element*₁, [*element*₂, ..., *element*_n].*operator*.*datavalue*₁
[*datavalue*₂...*datavalue*_n].END.)]

[RECORD ({ ALL
[*record - code*] })
ELEMENT (*element - list*) ...
DATA (*.data - string*END.)]

FUNCTION (MODIFY)

Description *Required.* Invokes the Modify function.

Consideration If the PDM returns a bad status, the Modify function closes and locks the file.

STANDARD-EXIT (*exit-name*)

Description *Optional.* Indicates the name of an exit program you want to invoke to process each modified record. For information on coding an exit program, see “[Writing exit programs](#)” on page 393.

Format 1–8 alphanumeric characters. The first character must be alphabetic.

Considerations

- ◆ If you code this statement, you must put it before the FILE statements.
- ◆ You must make your exit program available to be loaded by the Modify function. That is, it must reside in your execution library.
- ◆ Only one exit at a time resides in memory. If you code a new exit name in a subsequent function, the function deletes the current exit before it loads the new one.

FILE (*file-name*)

Description *Required.* Indicates the file you want the Modify function to access.

Format 4 alphanumeric characters. The first character must be alphabetic.

Considerations

- ◆ You cannot code FILE (*file-name-list*) or FILE (ALL).
- ◆ You cannot use the Modify function to change an index file.

OPEN - MODE ($\left[\begin{array}{c} \text{SUPD} \\ \text{EUPD} \end{array} \right]$)

Description *Optional.* Indicates how you want the file opened for processing.

Default SUPD

Options SUPD Opens the file for shared update.

EUPD Opens the file for exclusive update.

Consideration If you code this statement, you must put it before any RECORD statements.

CLOSE ($\left[\begin{array}{c} \text{NO} \\ \text{YES} \end{array} \right]$)

Description *Optional.* Indicates whether you want the function to explicitly close the file after modification.

Default YES

QUALIFIER ($\left\{ \begin{array}{c} \text{SERIAL} \\ \text{DIRECT} \\ \text{SEQUENTIAL} \end{array} \right\}$)

Description *Required.* Indicates the access mode you want the function to use when it modifies the current file.

Options SERIAL Accesses the file serially without regard to chain sequence.

DIRECT Accesses a specific record either by RRN or key.

SEQUENTIAL Accesses a related file sequentially by a specific linkpath.

Considerations

- ◆ Do not code QUALIFIER (SEQUENTIAL) for a primary file.
- ◆ As shown in the format, the QUALIFIER statement has three options. Depending on the type of file (primary (P) or related (R)) and the access mode you select, the statements are required (r), optional (o), or invalid (i) as follows:

QUALIFIER	DIRECT		SERIAL		SEQUENTIAL	
	P	R	P	R	P	R
RRN	i	r	i	i	i	i
LINKPATH	i	r	i	i	i	r
KEY	r	r	i	i	i	r
MAXIMUM	i	i	o	o	i	o

- ◆ If you code QUALIFIER (SEQUENTIAL) and the LINKPATH and KEY statements, the function modifies only the chain containing the key you code.
- ◆ If you code QUALIFIER (DIRECT), the function modifies only one record.
- ◆ If you code the QUALIFIER statement, you must put it before the RECORD statements.
- ◆ If you code QUALIFIER (SERIAL), you can modify many elements. The function modifies the elements you code for the record code you indicate. If you code RECORD (ALL), the function modifies every record.
- ◆ If you modify a coded related file, there must be at least one linkpath in the base portion of the file.

RRN (*record-rrn*)

Restrictions

- ◆ Use this statement for direct access to related files.
- ◆ You can only use this statement after a QUALIFIER statement.

Description *Optional.* Indicates a relative record number in a related file you want read directly.

Format 1–9 decimal characters

Consideration You must code this statement if you are modifying a related file and you coded QUALIFIER (DIRECT).

LINKPATH (*access-linkpath*)

Restrictions

- ◆ Use this statement for related files with direct or sequential access.
- ◆ You can only use this statement after a QUALIFIER statement.

Description *Optional.* Determines the access linkpath for a related file.

Format *ffff*LKxx, where *ffff* is a primary file linked to a related file through linkpath LKxx

Considerations

- ◆ The access linkpath may exist either in the base or the redefined portion of a coded record.
- ◆ If you code RECORD (ALL) with an access linkpath, that linkpath must exist in all records.
- ◆ You cannot code LINKPATH (*access-linkpath-list*).

```
KEY ( { D' dec-string'
      X' hex-string'
      C' char-string' } )
```

Restrictions

- ◆ Invalid for sequential access to a primary file or any serial access.
- ◆ You can only use this statement after a QUALIFIER statement.

Description *Optional.* Selects a key you want the function to use for direct access to a primary or related file, or sequential access to a related file.

Format

<i>D' dec-string'</i>	A decimal string of 1 to 256 bytes preceded by a D and surrounded by single quotes. This string must match the actual key length.
<i>X' hex-string'</i>	A hexadecimal string of 2 to 512 bytes that is preceded by an X and surrounded by single quotes. The length must be an even number and exactly twice the actual key length.
<i>C' char-string'</i>	A character string of 1 to 256 bytes preceded by a C and surrounded by single quotes. This string must match the actual key length. If a character string contains quotes, you must code two quotation marks for each quote. For example, you must code ABC D'EF'G as C'ABC"EF"G'.

$$\left[\text{MAXIMUM} \left(\left[\begin{array}{c} \underline{b} \\ \text{record-count} \end{array} \right] \right) \right]$$
Restrictions

- ◆ You can use this statement for serial access to primary or related files, or sequential access to related files. You cannot use it for direct access to primary or related files or for sequential access to primary files.
- ◆ You can only use this statement after a QUALIFIER statement.

Description *Optional.* Determines the maximum number of records to modify.

Default *b*

Format 1–9 decimal characters

Considerations

- ◆ If you code MAXIMUM (*b*), the function modifies all records.
- ◆ To code the MAXIMUM statement, you must have already coded the QUALIFIER statement.
- ◆ If you code the MAXIMUM statement, you must put it before the RECORD statements.
- ◆ The function counts only valid data records that pass all other selection criteria. For example, if you code an argument with the CRITERIA statement, all data records must first pass the argument validation before the function adds them to the maximum record counter.
- ◆ If you code a number that exceeds the total number of records in this file, the function stops processing at the end of the file.
- ◆ If you code MAXIMUM (0), no records are processed.

**CRITERIA (*element1*[,*element2*,...,*elementn*].*operator.datavalue1*
[.*datavalue2*...*datavalue*n]END.)**

Restriction You can only use this statement after a QUALIFIER statement.

Description *Optional.* Establishes an argument string to select the records you want modified.

Considerations

- ◆ If you code this statement, put it before the RECORD statements.
- ◆ You can code any number of spaces before the element list, after END., or on either side of the separating commas.
- ◆ If you do not code END., the function considers the rest of the program as data.

Format for element

One or more 8 alphanumeric character element names. You must make the first character in each name alphabetic and separate the names with commas.

Considerations

- ◆ If you code an element name in the argument, you must code the element name in the ELEMENT statement unless you code ELEMENT (ALL).
- ◆ You cannot code a null element list.
- ◆ If you name an element in the CRITERIA argument, it must be in all records you want modified in the file.

Format for operator

Only one Boolean operator may be specified. The following operators are valid:

- .EQ. Equal
- .NE. Not equal
- .GT. Greater than
- .LT. Less than
- .GE. Greater than or equal to
- .LE. Less than or equal to

Consideration You must code a period before and after each Boolean operator.

Format for datavalue

Any valid EBCDIC value. You can code the actual hexadecimal representation of any value of any data type in your UCL statement. You must put a period before datavalue and END. after it.

Considerations

- ◆ You must make the data values the same length as element names.
- ◆ You cannot put spaces between data values.
- ◆ Your data may cross input line boundaries if necessary. If so, you must stop in column 72 and continue on the next line in column 1. (If you put data in columns 73–80, it is lost.)

RECORD ($\left\{ \begin{array}{l} \underline{\text{ALL}} \\ \text{record-code} \end{array} \right\}$)

Description *Optional.* Indicates the record you want modified.

Default ALL

Format 2 alphanumeric characters

Considerations

- ◆ If you code this statement, you must code ELEMENT and DATA statements.
- ◆ If you code RECORD (ALL), the function modifies all record codes.
- ◆ Do not code RECORD (ALL) if you intend to code the element list with redefined element names for a coded related file.
- ◆ For primary files, always code RECORD (ALL).
- ◆ If you code a *record-code* here, it must be in the file you are modifying.
- ◆ If you code several RECORD statements, do not code ALL with a specific *record-code*.
- ◆ If you code RECORD (), the function modifies no records.

ELEMENT (*element-list*)

- Restriction** *Required* if you code the RECORD statement.
- Description** Indicates the data elements you want modified.
- Format** Element names must be 8 alphanumeric characters. The first character must be alphabetic. Separate names with commas.

Considerations

- ◆ The key element for a related file is the one associated with the access linkpath you coded.
- ◆ Additional constraints apply in the following situations:

In this context:	ELEMENT (<i>element-list</i>) must conform to these rules:
FILE(<i>primary-file</i>)	Do not include the root element in the element-list.
FILE(<i>related-file</i>)	Do not code linkpaths in the element list.
FILE(<i>related-file</i>) LINKPATH(<i>ffffLKxx</i>)	Do not code linkpaths in the element list.
FILE(<i>coded-file</i>) no access-linkpath	First entry in the element list must be <i>ffffCODE</i> where <i>ffff</i> is the coded file name. Do not code linkpaths in the element list.
FILE(<i>noncoded-file</i>)	The key must be the first element in the list.
FILE(<i>coded-file</i>) LINKPATH (<i>ffffLKxx</i>)	First entry in the element list must be <i>ffffCODE</i> where <i>ffff</i> is the coded file name. Do not code linkpaths in the element list.
RECORD(<i>record-code</i>)	Names appearing in the element list must exist in record code.
FILE(<i>coded-file</i>) RECORD(ALL)	Do not include redefined element names in the element list.

DATA (.*data-string*END.)

Description *Required.* Provides a string of data you want the function to use to modify an existing database record.

Format Any valid EBCDIC value. You can code the actual hexadecimal value of any data type in your UCL statement. You must put a period before *data-string* and END. after it.

Considerations

- ◆ You must make the data values in the string the same type and length as the data they replace. They must correspond on a one-to-one basis with the list you coded in the ELEMENT statement.
- ◆ Do not put blanks between data values.
- ◆ Your data may cross input line boundaries if necessary. If you code multiple lines, you must stop in column 72 and continue on the next line in column 1. (If you put data in columns 73–80, it is lost.)

Writing exit programs

You can use the exit point from the Modify function to examine the records you are modifying. The function invokes your exit program after it modifies the record. Therefore, there are no return codes. Your program can collect statistics or data on the records it is passed, but you cannot modify, delete, or add records.

For information on how your exit programs are loaded, how they operate, the languages you can use to write them, and the register conventions you must follow, see “[Inserting exit programs into functions](#)” on page 49. For example, you must code the parameter list addresses in register 1. For a description of the parameter list addresses, see the following table.

Parameter	Data type	Contents before exit (passed to exit program)	Contents after exit (passed from exit program)
Record	<i>n</i> bytes of data	Data record	Must be unchanged
Function Name	8 bytes character	MODIFY bb	Must be unchanged

If your exit program changes anything it is not authorized to change, the results are unpredictable.

Modify examples

Example 1 The following example shows you how to modify the CUST file and invoke an exit program EXIT0004. You code the UCL so the function opens the file for exclusive update and access it directly. You indicate the record you want to modify the CUST file by coding its key, the decimal string C'001234'. Since CUST is a primary file, the function processes all record codes, modifying specifically the CUSTNMBR and CUSTNAME elements. In the DATA statement, you indicate the data string that you want to take the place of the elements listed in the ELEMENT statement.

```
CONTROL (BEGIN)
  ENV-DESC (MYDESC)
  SCHEMA (MYSHEMA)
FUNCTION (MODIFY)
  STANDARD-EXIT (EXIT0004)
  FILE (CUST)
  OPEN-MODE (EUPD)
  QUALIFIER (DIRECT)
  KEY (C'001234')
  RECORD (ALL)
  ELEMENT (CUSTNMBR, CUSTNAME)
  DATA (.01209JOHN PAUL JONES END.)
CONTROL (END)
```

Invokes Modify function

Indicates your exit program

Indicates CUST file

Indicates CUST open mode

Indicates CUST access mode

Indicates CUST key value

All records required for primary file

Indicates CUST elements

Indicates CUST data

Example 2 The following example shows how to code the function section to clear linkpaths in primary files. Clearing linkpaths with the Modify function enables you to unload and load related files, but not the primary files with which they are associated.

```
FUNCTION (MODIFY)
FILE (CUST)
  QUALIFIER (SERIAL)
  RECORD (ALL)
  ELEMENT (CUSTLK010)
  DATA (. END.)
```

Example 3 The following example shows sample input and output.

```

CSUL0101I : COMMENCING COMMAND VALIDATION.
 1 CONTROL(BEGIN)
*****
 3 *
 4 * MODIFY EXAMPLE #1 DESCRIPTION
 5 *
 6 * OBJECTIVES:
 7 *
 8 * 1. FOR THE PRIMARY FILE PANM RECORD WITH KEY 'PKEY8'
 9 * CHANGE THE ELEMENT PANMDATA TO THE NEW VALUE.
10 *
11 *
12 * 2. FOR THE RELATED FILE RANV RECORD WITH RRN 4,
13 * CHANGE THE ELEMENT RANVFIL3 TO THE NEW VALUE.
14 *
15 *
*****
16 ENV-DESC(UTED00US)
17 SCHEMA(UTILSCHM)
18 LIST(ALL)
19 HEADER(YES)
20 EXTENSION(' MODIFY EXAMPLE #1')
21 **
22 FUNCTION(MODIFY)
23 FILE(PANM)
24 QUALIFIER(DIRECT)
25 KEY(C'PKEY8')
26 RECORD(ALL)
27 ELEMENT(PANMDATA)
28 DATA(. * THE NEW DATA NAME *END.)
29 **
30 FILE(RANV)
31 QUALIFIER(DIRECT)
32 RRN(0004)
33 LINKPATH(PANMLK01)
34 KEY(C'PKEY8')
35 RECORD(03)
36 ELEMENT(RANVFIL3)
37 DATA(. ***** THE NEW DATA FIELD FOR RANVFIL3 *****END.)
38 **
39 CONTROL(END)
CONTENTS OF SOURCE LINES OUTSIDE 1...72 MARGINS IGNORED.
 0 SYNTAX ERRORS DETECTED.
40 COMMAND LINES READ.
 1 CONTROL SECTIONS ANALYZED.
 1 FUNCTION COMMANDS ANALYZED.

MODIFY EXAMPLE #1

CSUL0102I : COMMENCING COMMAND EXECUTION.
CSUL0301I : COMMENCING CONTROL SECTION USING ENVIRONMENT DESCRIPTION UTED00US AND SCHEMA UTILSCHM.
CSUL0302I : COMMENCING MODIFY PROCESS.
CSUL0311I : COMMENCING MODIFY AGAINST FILE PANM.
CSUL1400I : RECORD IMAGE BEFORE MODIFY IS :

PANMDATA
DATAFORPANMPKEY8
CSUL1401I : RECORD IMAGE AFTER MODIFY IS :

PANMDATA
* THE NEW DATA NAME *
CSUL0345I : THE SPECIFIED RECORD IN THE CURRENT FILE HAS BEEN DIRECTLY ACCESSED AND PROCESSED.
CSUL0321I : MODIFY PROCESSING AGAINST FILE PANM TERMINATING NORMALLY.
MODIFY EXAMPLE #1

```

```
CSUL0311I : COMMENCING MODIFY          AGAINST FILE RANV.
CSUL1400I : RECORD IMAGE BEFORE MODIFY IS :

RANVFIL3

CSUL1401I : RECORD IMAGE AFTER MODIFY IS :

RANVFIL3
***** THE NEW DATA FIELD FOR RANVFIL3 *****
CSUL0345I : THE SPECIFIED RECORD IN THE CURRENT FILE HAS BEEN DIRECTLY ACCESSED AND PROCESSED.
CSUL0321I : MODIFY                      PROCESSING AGAINST FILE RANV TERMINATING NORMALLY.
CSUL0303I : MODIFY                      PROCESS TERMINATING.
CSUL0305I : CONTROL SECTION TERMINATING.
CSUL0306I : SUMMARY DATA FOR TERMINATING CONTROL SECTION :
CSUL0361I : NUMBER OF READS ISSUED TO THE PDM =                2
CSUL0362I : NUMBER OF RECORDS RECEIVED FROM THE PDM =         2
CSUL0363I : NUMBER OF RECORDS PROCESSED =                     2
CSUL0364I : NUMBER OF RECORDS PRINTED =                       4
CSUL0365I : NUMBER OF RECORDS UPDATED =                       2
CSUL0366I : NUMBER OF WRITES ISSUED TO PDM =                 2
CSUL0307I : ALL CONTROL SECTIONS PROCESSED.
CSUL0308I : CUMULATIVE SUMMARY DATA FOR ALL CONTROL SECTIONS :
CSUL0361I : NUMBER OF READS ISSUED TO THE PDM =                2
CSUL0362I : NUMBER OF RECORDS RECEIVED FROM THE PDM =         2
CSUL0363I : NUMBER OF RECORDS PROCESSED =                     2
CSUL0364I : NUMBER OF RECORDS PRINTED =                       4
CSUL0365I : NUMBER OF RECORDS UPDATED =                       2
CSUL0366I : NUMBER OF WRITES ISSUED TO PDM =                 2
CSUL0103I : DATABASE UTILITIES SUCCESSFUL TERMINATION.
```

Example 4 The following example shows sample input and output.

```

CSUL0101I : COMMENCING COMMAND VALIDATION.
 1 CONTROL(BEGIN)
 2 *****
 3 *
 4 * MODIFY EXAMPLE #2 DESCRIPTION *
 5 * *
 6 * OBJECTIVE: *
 7 * *
 8 * 1. FOR THE RELATED FILE RANV, READ SEQUENTIALLY ALONG *
 9 * THE SPECIFIED LINKPATH AND MODIFY THE ELEMENT *
10 * SPECIFIED FOR EACH STATED RECORD CODE. *
11 * *
12 * *
13 *****
14 ENV-DESC(UTED00US)
15 SCHEMA(UTILSCHM)
16 LIST(ALL)
17 HEADER(YES)
18 EXTENSION(' MODIFY EXAMPLE #2')
19 DATA-FORMAT(CHAR HEX)
20 FUNCTION(MODIFY)
21 FILE(RANV)
22 QUALIFIER(SEQUENTIAL)
23 LINKPATH(PANMLK01)
24 KEY(C'PKEY8')
25 RECORD(01)
26 ELEMENT(RANVFIL1)
27 DATA(
28 .*** THE NEW DATA FIELD FOR RANVFIL1 **END.)
29 RECORD(02)
30 ELEMENT(RANVFIL2)
31 DATA(
32 .**** THE NEW DATA FIELD FOR RANVFIL2 ****END.)
33 RECORD(03)
34 ELEMENT(RANVFIL3)
35 DATA(.***** THE NEW DATA FIELD FOR RANVFIL3 *****END.)
36 CONTROL(END)
CONTENTS OF SOURCE LINES OUTSIDE 1...72 MARGINS IGNORED.
 0 SYNTAX ERRORS DETECTED.
36 COMMAND LINES READ.
 1 CONTROL SECTIONS ANALYZED.
 1 FUNCTION COMMANDS ANALYZED.
MODIFY EXAMPLE #2

CSUL0102I : COMMENCING COMMAND EXECUTION.
CSUL0301I : COMMENCING CONTROL SECTION USING ENVIRONMENT DESCRIPTION UTED00US AND SCHEMA UTILSCHM.
CSUL0302I : COMMENCING MODIFY PROCESS.
CSUL0311I : COMMENCING MODIFY AGAINST FILE RANV.
CSUL1400I : RECORD IMAGE BEFORE MODIFY IS :

```


14

Coding the PDM Termination utility

Coding the PDM Termination utility

Use the PDM Termination utility to shut down the PDM by executing a single function.

To execute the PDM Termination utility:

- ◆ Code the appropriate input statements.
- ◆ **OS/390** In OS/390, execute the JCL sample TXJSHUTP, or use the cataloged procedure TISDBTMC.
- ◆ **VSE** In VSE, submit the JCL sample TXJSHUTP.

This utility does not use UCL. Instead, it has its own form of input statements. For information on defining files, see “[Defining files for the PDM Termination utility](#)” on page 40.



If the PDM Termination utility encounters any errors when validating the input, it does not proceed to the PDM with the termination process. The utility calls the PDM only if the input is valid.

Coding the input statements for the PDM Termination utility

Enter the following statements in the INPUT file to designate how you want the PDM shut down. You can put the statements in any order. You can code all of the statements on one line, or you may put one statement on a line.

PASSWORD (*shutdown-password*)

$$\left[\text{FORCE} \left(\left\{ \begin{array}{c} \text{NO} \\ \text{YES} \end{array} \right\} \right) \right]$$

[DBMNAME (*dbmname*)]

$$\left[\text{CONSOLE} \left(\left\{ \begin{array}{c} \text{NO} \\ \text{YES} \end{array} \right\} \right) \right]$$

PASSWORD (*shutdown-password*)

- Restriction** *Required* if there is a password in the environment description.
- Description** Identifies the shutdown password. With this statement, you can restrict the use of this utility to the DBA or an authorized person.
- Format** 1–8 byte password as defined in the environment description.
- Considerations**
- ◆ If you enter an invalid password, the PDM returns an error status instead of terminating.
 - ◆ If there is no password in the environment description, do not code this statement.

```
FORCE ( { NO } )
```

Description	<i>Optional.</i> Indicates whether the PDM should be shut down while tasks are still logged on.	
Default	NO	
Options	NO	If all tasks are signed off, the PDM shuts down. If tasks are signed on, the PDM does not shut down and a message is returned.
	YES	The utility shuts down the PDM even if tasks are signed on, but after all current functions are executed. The utility closes files without unlocking them. If task logging is active, all tasks signed on are reset. If task logging is not active, all tasks are signed off.

DBMNAME (*dbmname*)

Description	<i>Optional.</i> Identifies the PDM you want shut down.	
Default	If you do not code <i>dbmname</i> , the PDM uses the name in the CSIPARM file.	
Format	1–8 alphanumeric characters. The first character must be alphabetic.	

```
CONSOLE ( { NO } )
```

Description	<i>Optional.</i> Indicates whether the utility is to display a message on the operator's console stating that the PDM shutdown was successful or unsuccessful.	
Default	NO	

PDM termination example

The following is an example of input:

```
FORCE (YES)    CONSOLE (NO)    DBMNAME (PDMTEST)
```

The following messages result from the input:

```
CSUL4000I      : THE FOLLOWING INPUT CONTROL STATEMENTS WERE SPECIFIED
FORCE-OPTION   : YES
DBMNAME        : PDMTEST
CONSOLE OPTION : NO
CSUL4011I      : DATABASE SHUTDOWN SUCCESSFUL. STATUS = ****
```

15

Coding the Execution Statistics utility for release 2.1.6

Coding the Execution Statistics utility for release 2.1.6

Use the Execution Statistics utility (CSUXSTAT) to generate a statistics report. This report shows the contents of the statistics file and calculations based on the contents.

The PDM places a group of statistics records in a statistics file at these times:

- ◆ When the PDM finishes initializing
- ◆ When an application issues a Read Statistics (RSTAT) PDML command
- ◆ When the PDM terminates

The Execution Statistics utility generates a report from a PDM statistics file. For more information on generating a PDM statistics file, refer to the *SUPRA Server PDM DML Programming Guide (OS/390 & VSE)*, P26-4340.

Defining the files

The Execution Statistics utility does not use UCL or access Directory or database files. To execute the utility, you need to define only two files: INPUT and STATS. The INPUT file describes the record size and block size of the STATS file. The STATS file is created during PDM execution and contains statistics records. When you define the INPUT file, you must code record size in positions 1–4, a blank in position 5, and block size in positions 6–9. Use leading zeros in both fields.

When you code the record size, you must make it at least 512 bytes. You must set the record size through Directory Maintenance so that Directory Maintenance automatically calculates the block size. Make sure the block size is an even multiple of the record size and then add 4 because the file is blocked. The block size must be at least 516 bytes.

The following formula shows the calculation:

$$\text{the block size} = (\text{the record size} * n) + 4$$

The record size and block size must exactly match the file definition you used when you executed the Execution Statistics utility. That is, the record size and block size must match the values used in the JCL in OS/390 and VSE. The file definition in the Execution Statistics utility must match the file definition used when the file was created during execution of the PDM. Therefore, in your JCL for the PDM, utilities, and INPUT file, set your record size to 512 and block size to 516.

To get Directory Maintenance to set the block size to 516, you may need to set the record size to 516 in Directory Maintenance and 512 in the JCL.

To execute the utility in OS/390 or VSE, see sample JCL member TXJPSTAT. For information on the TIS/XA Selection Facility, refer to the [SUPRA Server PDM and Directory Administration Guide \(OS/390 & VSE\)](#), P26-2250.

See “[Defining files for the Execution Statistics utility](#)” on page 38 for more information on defining files.

Arrangement of the statistics report

An execution statistics report consists of:

- ◆ A SUPRA DBA utilities title page
- ◆ A DBA execution statistics report title page
- ◆ A Physical Data Manager identification page
- ◆ At least one group report
- ◆ An execution statistics termination page

Each group report shows a single group of statistics that was placed in the statistics file at the same time. A group report contains:

- ◆ A group identification page
- ◆ A system statistics page
- ◆ A set of file statistics pages (one page for each file defined to the PDM)
- ◆ A file statistics totals page

The following sections show samples of each type of page.

Sample of the Physical Data Manager identification page

A Physical Data Manager identification page similar to the one shown below is printed once at the beginning of each execution statistics report.

```

FUNCTION = EXECUTION STATISTICS
P H Y S I C A L   D A T A   M A N A G E R
I D E N T I F I C A T I O N

PDM NAME                EE73EX01
BOOT SCHEMA             CINDIRSC
BOOT ENVIRONMENT DESCRIPTION CINDIRTU
USER SCHEMA             UTILSCHM
USER ENVIRONMENT DESCRIPTION UTED50EX

DATE AND TIME OF PDM INITIALIZATION NOV. 01, 1992
10:15:10
    
```

Sample of the group identification page

The group identification page shown below is printed at the beginning of each group report. The table following this sample gives more information on each statistic.

```

FUNCTION = EXECUTION STATISTICS

G R O U P   I D E N T I F I C A T I O N
F O R   G R O U P   2

G1.01  STATISTICS GROUP TYPE                PDM TERMINATION
G2.01  DATE AND TIME EXECUTION STATISTICS WERE ISSUED  NOV. 01, 1992 10:18:13
    
```

Statistic identifier	Explanation
G1.01	This shows the reason the PDM placed a group of records in the statistics file. For example, the PDM completes initialization, an RSTAT command is issued with the FILE option, or the PDM completes termination.
G2.01	This shows the date and time the PDM placed the group of statistics records in the statistics file.

Sample of the system statistics page

A system statistics page similar to the one shown below is the first set of statistics in each group report. These PDM system-level statistics cover the time from when the statistics were last reset (S1.01) until they were placed in the statistics file (G2.01). They are not specific to the task that issued the RSTAT command and caused the statistics to be placed in the file.

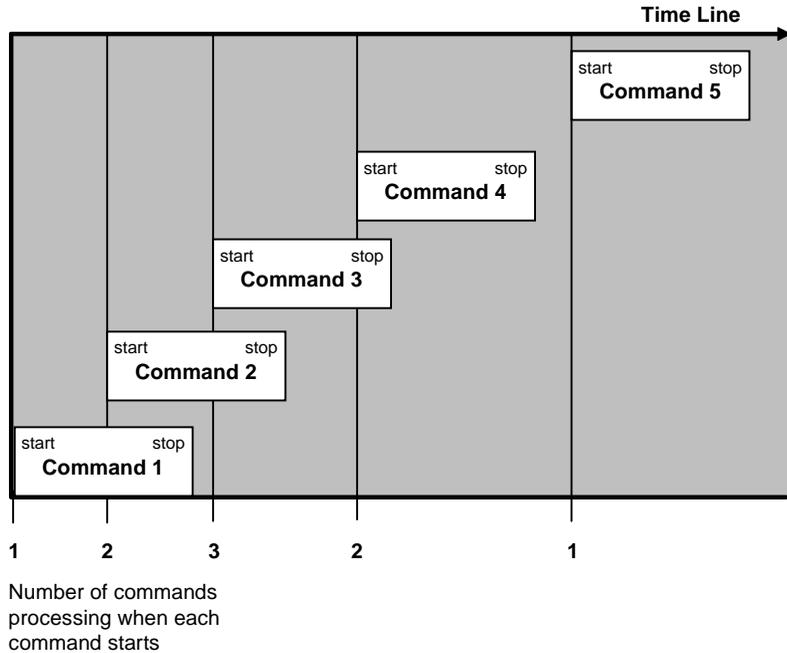
For more information on each statistic, see the table following this sample.

FUNCTION = EXECUTION STATISTICS			
			GROUP 2.1
S Y S T E M S T A T I S T I C S			
S1.01	DATE AND TIME STATISTICS WERE LAST RESET	NOV. 01, 1992	10:16:07
S1.02	FIRST GROUP OF EXECUTION STATISTICS SINCE RESET?	YES	
S2.01	TOTAL TASKS		2
S2.02	MAXIMUM CONCURRENT TASKS		5
S3.01	TOTAL HELD RECORDS		1,439
S3.02	MAXIMUM HELD RECORDS		584
S3.03	TOTAL RECORDS HELD BY OTHER TASKS		0
S3.04	TOTAL HELD RECORDS STOLEN BY ANOTHER TASK		0
S4.01	TOTAL READ COMMANDS		121
	21.96%		
S4.02	TOTAL UPDATE COMMANDS		9
	1.63%		
S4.03	TOTAL ADD AND DELETE COMMANDS		389
	70.60%		
S4.04	TOTAL OTHER COMMANDS		30
	5.44%		
S5.01	TOTAL COMMANDS ISSUED TO THE PDM		551
S5.02	MAXIMUM NUMBER OF COMMANDS AT COMMAND STARTS		4
S5.03	SUM OF COMMANDS AT COMMAND STARTS		1,402
S5.04	AVERAGE NUMBER OF COMMANDS AT COMMAND STARTS	(S5.03/S5.01)	
	2.54AVG.		
S5.05	TOTAL ELAPSED TIME ON COMMANDS ISSUED TO THE PDM		00:03:20.676
S5.06	AVERAGE ELAPSED TIME PER COMMAND ISSUED TO THE PDM	(S5.05/S5.01)	00:00:00.364
S5.07	MAXIMUM ELAPSED TIME FOR ANY COMMAND ISSUED TO THE PDM		00:00:11.086
S6.01	NUMBER OF TIMES PDM WAS INACTIVE		3,175
S6.02	AVERAGE NUMBER OF TIMES PDM WAS INACTIVE PER COMMAND	(S6.01/S5.01)	
	5.76AVG.		
S7.01	AMOUNT OF TIME PDM WAS ACTIVE	(HH:MM:SS.SSS)	00:00:25.587
	20.28%		
S7.02	AMOUNT OF TIME PDM WAS INACTIVE		00:01:40.575
	79.72%		
S7.03	TOTAL PDM TIME	(S7.01+S7.02)	00:02:06.162
	100.00%		
S7.04	AVERAGE AMOUNT OF TIME PDM WAS ACTIVE PER COMMAND	(S7.01/S5.01)	00:00:00.046

Statistics identifier	Additional information
S1.01	The date and time the memory storage area (where statistics are accumulated) was last reset.
S1.02	First set of execution statistics since reset? Yes or No.
S2.01	The total number of SINON commands issued to the PDM. It includes sign-ons that failed and sign-ons used to reconnect an active task after a task or system failure (task-level recovery only).
S2.02	The maximum number of tasks that were signed on at any one time.
S3.01	The number of times a record was automatically reserved, explicitly reserved, or locked.
S3.02	The maximum number of records automatically reserved, explicitly reserved, or locked simultaneously.
S3.03	The number of requests by a batch task for a record already held by a TP task in a non-task level recovery environment.
S3.04	The number of times a record was stolen from a CICS task for use by another task. When task logging is active and when you are using a batch PDM, this value is always 0.
S4.01	The number of times read commands were issued to the PDM by PDM interfaces. The statistic includes commands that failed in the PDM, but does not include commands issued to a PDM interface and not passed to the PDM.
S4.02	The number of times write commands were issued to the PDM by PDM interfaces. The statistic includes commands that failed in the PDM, but does not include commands issued to a PDM interface and not passed to the PDM.
S4.03	The number of times add and delete commands were issued to the PDM by PDM interfaces. The statistic includes commands that failed in the PDM, but does not include commands that the PDM interface did not pass to the PDM.
S4.04	The number of commands other than reads, writes, adds, or deletes that the PDM interfaces issued to the PDM. The statistic includes commands that failed in the PDM, but does not include commands that the PDM interfaces did not pass to the PDM.
S5.01	The number of commands that PDM interfaces issued to the PDM. The statistic does not include commands the PDM interfaces did not pass to the PDM.
S5.02	The maximum number of commands processing simultaneously (see the figure following this table). A command is processing if it has been passed to the PDM and not yet returned to the interface.

Statistics identifier	Additional information
S5.03	Sum of commands processing when a command starts processing, including commands starting and already processing. Use this statistic to calculate the average number of commands being processed by the PDM when commands start processing (see the figure following this table).
S5.04	The sum of commands at command starts (S5.03) divided by the total number of commands issued to the PDM (S5.01).
S5.05	The total elapsed time of all commands. The elapsed time for a single command begins when the PDM interface issues the command to the PDM and ends when the PDM interface receives notice that the command completed. This statistic is given in hours, minutes, seconds, and milliseconds.
S5.06	The elapsed time on commands issued to the PDM (S5.05) divided by total commands issued (S5.01). This statistic is given in hours, minutes, seconds, and milliseconds.
S5.07	The maximum elapsed time to process a single command out of all the commands in this group report. This statistic is given in hours, minutes, seconds, and milliseconds.
S6.01	The number of times the PDM issued an operating system wait because there was no processing to do.
S6.02	The average number of times the PDM was inactive per command. This value is calculated by dividing the number of times the PDM was inactive (S6.01) by the total commands issued to the PDM (S5.01).
S7.01	The amount of time, in hours, minutes, seconds, and milliseconds, that the PDM was executing.
S7.02	The amount of time, in hours, minutes, seconds, and milliseconds, that the PDM was in an operating system wait mode.
S7.03	Total amount of active (S7.01) and inactive PDM time (S7.02), in hours, minutes, seconds, and milliseconds.
S7.04	The average time the PDM took to execute a command, in hours, minutes, seconds, and milliseconds. This number is calculated by dividing the amount of time the PDM was active (S7.01) by the number of commands issued by the PDM (S5.01).

The following figure illustrates how statistics for command starts are gathered:



The PDM counts the number of commands processing each time a new command starts. The PDM counts the command that is starting as the first command. In this example, only one command, Command 1, is processing when Command 1 starts. When Command 2 starts, two commands are processing: Commands 1 and 2. When Command 3 starts, three commands are processing. However, when Command 4 starts, Commands 1 and 2 have finished processing, so only two commands are processing: Commands 3 and 4. When Command 5 starts, it is the only command processing.

To arrive at the maximum number of commands processing when each command starts (S5.02), the PDM picks the highest number from the figures along the base line (3).

To arrive at the sum of commands processing when commands start (S5.03), the PDM adds the numbers along the base line (1+2+3+2+1=9).

Sample of the file statistics page

A file statistics page, like the one shown below, is included in the group report. There is one page for each file defined to the PDM when the statistics record group was placed in the statistics file. The files defined to the PDM include all database, Directory, log, and statistics files. The statistics describe the activity for the file from the time the statistics were last reset (S1.01) until the statistics record group was placed in the statistics file (G2.01). These statistics are at the PDM file level, and, therefore, are not specific to the task which issued the RSTAT command. For more information on each statistic, see the table following this sample.

FUNCTION = EXECUTION STATISTICS		GROUP 2.18	
FILE STATISTICS			
FOR FILE P002			
FILE TYPE	PRIMARY	LOGICAL RECORD LENGTH	57
FILE CODED	NO	BLOCK SIZE	9,405
FILE DDNAME	P002	BLOCKS PER TRACK	15
ACCESS METHOD	BDAM	RECORDS PER BLOCK	165
BUFFER POOL	USRM	TOTAL LOGICAL RECORDS	660
	CONTROL INTERVAL SIZE	0	
F1.01	TOTAL LOGICAL READS		132
F1.02	TOTAL PHYSICAL READS		7
F1.03	TOTAL IN-MEMORY HITS	(F1.01-F1.02)	125
F1.04	TOTAL IN-MEMORY HITS ON UPDATED BUFFER		0
F1.05	TOTAL PHYSICAL UPDATES FORCED BY A PHYSICAL READ		6
F2.01	AVERAGE LOGICAL READS PER PHYSICAL READ	(F1.01/F1.02)	18.86AVG.
F2.02	% OF LOGICAL READS WHICH WERE IN-MEMORY HITS	((F1.03/F1.01)*100)	94.70%
F2.03	% OF IN-MEMORY HITS WHICH WERE TO AN UPDATED BUFFER	((F1.04/F1.03)*100)	0.00%
F2.04	% OF PHYSICAL READS FORCING A PHYSICAL UPDATE	((F1.05/F1.02)*100)	4.55%
F3.01	TOTAL LOGICAL UPDATES		129
F3.02	TOTAL PHYSICAL UPDATES		9
F3.03	TOTAL MULTIPLE LOGICAL UPDATES TO THE SAME BUFFER		0
F3.04	AVERAGE LOGICAL UPDATES PER PHYSICAL UPDATE	(F3.01/F3.02)	14.33AVG.
F3.05	% OF PHYSICAL UPDATES WHICH WERE MULTIPLE UPDATES	((F3.03/F3.02)*100)	0.00%
F4.01	TOTAL LOGICAL I/O	(F1.01+F3.01)	261
F4.02	TOTAL PHYSICAL I/O	(F1.02+F3.02)	16
F4.03	AVERAGE LOGICAL I/O PER PHYSICAL I/O	(F4.01/F4.02)	16.31AVG.
F5.01	% OF LOGICAL I/O WHICH WERE LOGICAL READS	((F1.01/F4.01)*100)	50.57%
F5.02	% OF LOGICAL I/O WHICH WERE LOGICAL UPDATES	((F3.01/F4.01)*100)	49.43%
F5.03	% OF PHYSICAL I/O WHICH WERE PHYSICAL READS	((F1.02/F4.02)*100)	43.75%
F4	% OF PHYSICAL I/O WHICH WERE PHYSICAL UPDATES	((F3.02/F4.02)*100)	56.25%
F6.01	NUMBER OF TIMES PHYSICAL WRITES TO A DATABASE FILE WERE DELAYED BECAUSE A LOG FILE BLOCK HAD TO BE WRITTEN		1

Identifier	Additional information
F1.01	The number of times a record was logically read.
F1.02	The number of times a record was physically read.
F1.03	The number of times a record was logically read and was already in memory (F1.01-F1.02).
F1.04	The number of logical reads that found the desired block of data in a updated storage buffer.
F1.05	The number of forced physical updates that occurred because a logical read required a buffer for a physical read.
F2.01	The number of logical reads (F1.01) divided by the number of physical reads (F1.02).
F2.02	The percentage of logical reads (F1.01) that were in-memory hits (F1.03).
F2.03	The percentage of in-memory hits (F1.03) that were hits on an updated storage buffer (F1.04).
F2.04	The number of physical reads (F1.02) that forced a physical write to obtain a buffer (F1.05).
F3.01	The number of times a record was logically updated. A record is logically updated when it is logically added to the file, deleted from the file, or changed in the file.
F3.02	The number of times the contents of a buffer were physically written. For Task and System Log Files, F3.02 may exceed F3.01 because a single logical record may span several physical blocks. This can cause several buffers to be physically written for a single logical write.
F3.03	The number of logical updates to storage buffers that have already been updated.
F3.04	The number of logical updates (F3.01) divided by the number of physical updates (F3.02).
F3.05	The number of updates to previously updated buffers (F3.03) as a percentage of physical updates (F3.02).
F4.01	The number of logical read and update (change, add, or delete) commands received by the PDM.
F4.02	The number of physical read and update operations performed by the PDM.
F4.03	The number of logical I/O transactions (F4.01) divided by the number of physical I/O transactions (F4.02).
F5.01	The logical read (F1.01) commands as a percentage of logical I/O commands (F4.01).
F5.02	The logical update (F3.01) commands as a percentage of logical I/O commands (F4.01).
F5.03	The physical reads (F1.02) as a percentage of physical I/Os (F4.02).
F5.04	The physical updates (F3.02) as a percentage of physical I/Os (F4.02).
F6.01	The number of times physical writes to a database file were delayed because a System or Task Log File block had to be physically written first. This statistic is 0 for files other than primary, related, and index files.

Sample of the file statistics totals for group

A file statistics totals page, like the one shown below, is included in each group report following the individual file statistic pages. Except for T1.00 and T6.02–T6.05, these statistics summarize the corresponding statistics on the individual file statistics pages. These statistics summarize PDM activity at the file level from the time the statistics were last reset (S1.01) until the statistics record group was placed in the statistics file (G2.01). These statistics summarize Directory files, log files, and statistics files in addition to database files. They are not specific to the task that issued the RSTAT command. For more information on each statistic, see the table following this sample.

FUNCTION = EXECUTION STATISTICS		GROUP 2.28	
FILE STATISTICS			
TOTALS FOR GROUP 2			
T1.00	NUMBER OF FILES FOR WHICH STATISTICS WERE ACCUMULATED	26	
T1.01	TOTAL LOGICAL READS	3,931	
T1.02	TOTAL PHYSICAL READS	527	
T1.03	TOTAL IN-MEMORY HITS	(T1.01-T1.02)	3,404
T1.04	TOTAL IN-MEMORY HITS ON UPDATED BUFFER	2,832	
T1.05	TOTAL PHYSICAL UPDATES FORCED BY A PHYSICAL READ	301	
T2.01	AVERAGE LOGICAL READS PER PHYSICAL READ	(T1.01/T1.02)	7.46AVG.
T2.02	% OF LOGICAL READS WHICH WERE IN-MEMORY HITS	((T1.03/T1.01)*100)	86.59%
T2.03	% OF IN-MEMORY HITS WHICH WERE TO AN UPDATED BUFFER	((T1.04/T1.03)*100)	83.20%
T2.04	% OF PHYSICAL READS FORCING A PHYSICAL UPDATE	((T1.05/T1.02)*100)	7.66%
T3.01	TOTAL LOGICAL UPDATES	7,602	
T3.02	TOTAL PHYSICAL UPDATES	2,558	
T3.03	TOTAL MULTIPLE LOGICAL UPDATES TO THE SAME BUFFER	17	
T3.04	AVERAGE LOGICAL UPDATES PER PHYSICAL UPDATE	(T3.01/T3.02)	2.97AVG.
T3.05	% OF PHYSICAL UPDATES WHICH WERE MULTIPLE UPDATES	((T3.03/T3.02)*100)	0.66%
T4.01	TOTAL LOGICAL I/O	(T1.01+T3.01)	11,533
T4.02	TOTAL PHYSICAL I/O	(T1.02+T3.02)	3,085
T4.03	AVERAGE LOGICAL I/O PER PHYSICAL I/O	(T4.01/T4.02)	3.74AVG.
T5.01	% OF LOGICAL I/O WHICH WERE LOGICAL READS	((T1.01/T4.01)*100)	34.08%
T5.02	% OF LOGICAL I/O WHICH WERE LOGICAL UPDATES	((T3.01/T4.01)*100)	65.92%
T5.03	% OF PHYSICAL I/O WHICH WERE PHYSICAL READS	((T1.02/T4.02)*100)	17.08%
T5.04	% OF PHYSICAL I/O WHICH WERE PHYSICAL UPDATES	((T3.02/T4.02)*100)	82.92%
T6.01	NUMBER OF TIMES PHYSICAL WRITES TO A DATABASE FILE WERE DELAYED BECAUSE A LOG FILE BLOCK HAD TO BE WRITTEN	144	94.12%
T6.02	NUMBER OF TIMES LOGICAL WRITES TO A LOG FILE WERE DELAYED BECAUSE A LOG FILE BLOCK HAD TO BE WRITTEN	9	5.88%

T6.03	NUMBER OF TIMES WRITES WERE DELAYED BECAUSE A LOG FILE BLOCK HAD TO BE WRITTEN	(T6.01+T6.02)	153 100.00%
T6.04	TOTAL NUMBER OF WRITES TO THE LOG FILE(S)	5,440	
T6.05	AVERAGE LOGICAL LOG FILE BLOCK WRITES PER LOG FILE DELAY	(T6.04/T6.03)	35.56AVG.

Statistics identifier	Additional information
T1.00	The number of files defined to the PDM when the statistics record group was placed in the statistics file.
T1.01	The number of times a record was logically read.
T1.02	The number of times a record was physically read.
T1.03	The number of times a record was logically read and was already in memory.
T1.04	The number of logical reads that found the desired block of data in a storage buffer.
T1.05	The number of forced physical updates that occurred because a logical read required a buffer for a physical read.
T2.01	The number of logical reads (T1.01) divided by the number of physical reads (T1.02).
T2.02	The percentage of logical reads (T1.01) that were in-memory hits (T1.03).
T2.03	The percentage of in-memory hits (T1.03) that were hits on an updated buffer (T1.04).
T2.04	The number of physical reads (T1.02) that forced a physical write to obtain a buffer (T1.05).
T3.01	The number of times a record was logically updated. A record is logically updated when it is logically added to the file, deleted from the file, or changed in the file.
T3.02	The number of times the contents of a buffer were physically written.
T3.03	The number of logical updates to storage buffers that have already been updated.
T3.04	The number of logical updates (T3.01) divided by the number of physical updates (T3.02).
T3.05	The number of updates to previously updated buffers (T3.03) as a percentage of physical updates (T3.02).
T4.01	The number of logical read and update (change, add, or delete) commands received by the PDM.
T4.02	The number of times physical read and update operations were performed by the PDM.
T4.03	The number of logical I/O transactions (T4.01) divided by the number of physical I/O transactions (T4.02).

Statistics identifier	Additional information
T5.01	The logical read (T1.01) commands as a percentage of logical I/O commands (T4.01).
T5.02	The logical update (T3.01) commands as a percentage of logical I/O commands (T4.01).
T5.03	The physical read (T1.02) commands as a percentage of physical I/Os (T4.02).
T5.04	The physical updates (T3.02) as a percentage of physical I/Os (T4.02).
T6.01	The number of times physical writes to a database file were delayed because a System or Task Log File block had to be physically written first.
T6.02	The number of times a log file record (or when spanning records, a log record segment) could not be logically written because a System or Task Log File block had to be physically written.
T6.03	The number of times logical writes to a log file and physical writes to a database file were delayed because a System or Task Log File block had to be written (T6.01 + T6.02).
T6.04	The number of logical writes to the System and Task Log Files. When spanning records, each block in the record is considered a separate logical write.
T6.05	The number of records logically written to the log file(s) (T6.04) divided by the number of delays (T6.03).

Sample of the termination page

The termination page appears after at least one group report.

```
FUNCTION = EXECUTION STATISTICS

EXECUTION STATISTICS
TERMINATION

PDM NAME                EE73EX01
BOOT SCHEMA             CINDIRSC
BOOT ENVIRONMENT DESCRIPTION CINDIRTU
USER SCHEMA             UTILSCHM
USER ENVIRONMENT DESCRIPTION UTED50EX

TOTAL PDM EXECUTION STATISTICS GROUPS      2
TOTAL RECORDS ON STATISTICS FILE          56
```

16

Coding the Execution Statistics utility for release 2.4

Coding the Execution Statistics utility for release 2.4

Use the Execution Statistics utility (CSUXSTAT) to generate a statistics report. This report shows the contents of the statistics file and calculations based on the contents.

The PDM places a group of statistics records in a statistics file at these times:

- ◆ When the PDM finishes initializing
- ◆ When an application issues a Read Statistics (RSTAT) PDML command
- ◆ When the PDM terminates

The Execution Statistics utility generates a report from a PDM statistics file. For more information on generating a PDM statistics file, refer to the *SUPRA Server PDM DML Programming Guide (OS/390 & VSE)*, P26-4340.

Defining the files

The Execution Statistics utility does not use UCL or access Directory or database files. To execute the utility, you need to define only two files: INPUT and STATS. The INPUT file describes the record size and block size of the STATS file. The STATS file is created during PDM execution and contains statistics records. When you define the INPUT file, you must code record size in positions 1–4, a blank in position 5, and block size in positions 6–9. Use leading zeros in both fields.

When you code the record size, you must make it at least 512 bytes. You must set the record size through Directory Maintenance so that Directory Maintenance automatically calculates the block size. Make sure the block size is an even multiple of the record size and then add 4 because the file is blocked. The block size must be at least 516 bytes.

The following formula shows the calculation:

$$\text{the block size} = (\text{the record size} * n) + 4$$

The record size and block size must exactly match the file definition you used when you executed the Execution Statistics utility. That is, the record size and block size must match the values used in the JCL. The file definition in the Execution Statistics utility must match the file definition used when the file was created during execution of the PDM. Therefore, in your JCL for the PDM, utilities, and INPUT file, set your record size to 512 and block size to 516.

To get Directory Maintenance to set the block size to 516, you may need to set the record size to 516 in Directory Maintenance and 512 in the JCL.

To execute the utility, see sample JCL member TXJPSTAT. For information on the TIS/XA Selection Facility, refer to the *SUPRA Server PDM and Directory Administration Guide (OS/390 & VSE)*, P26-2250.

See “[Defining files for the Execution Statistics utility](#)” on page 38 for more information on defining files.

Arrangement of the statistics report

An execution statistics report consists of:

- ◆ A SUPRA DBA utilities title page
- ◆ A DBA execution statistics report title page
- ◆ A Physical Data Manager identification page
- ◆ At least one group report
- ◆ An execution statistics termination page

Each group report shows a single group of statistics that was placed in the statistics file at the same time. A group report contains:

- ◆ A group identification page
- ◆ A system statistics page
- ◆ A set of file statistics pages (one page for each file defined to the PDM)
- ◆ A file statistics totals page

The following sections show samples of each type of page.

Sample of the Physical Data Manager identification page

A Physical Data Manager identification page similar to the one shown below is printed once at the beginning of each execution statistics report.

```

FUNCTION = EXECUTION STATISTICS

P H Y S I C A L   D A T A   M A N A G E R
I D E N T I F I C A T I O N

PDM NAME                EE73EX01
BOOT SCHEMA             CINDIRSC
BOOT ENVIRONMENT DESCRIPTION CINDIRTU
USER SCHEMA             UTILSCHM
USER ENVIRONMENT DESCRIPTION UTED50EX

DATE AND TIME OF PDM INITIALIZATION NOV. 01, 1992
10:15:10
    
```

Sample of the group identification page

A group identification page similar to the one shown below is printed at the beginning of each group report. The table following this sample gives more information on each statistic.

```

FUNCTION = EXECUTION STATISTICS

G R O U P   I D E N T I F I C A T I O N
F O R   G R O U P   2

G1.01  STATISTICS GROUP TYPE                PDM TERMINATION
G2.01  DATE AND TIME EXECUTION STATISTICS WERE ISSUED NOV. 01, 1992 10:18:13
    
```

Statistic identifier	Explanation
G1.01	This shows the reason the PDM placed a group of records in the statistics file. For example, the PDM completes initialization, an RSTAT command is issued with the FILE option, or the PDM completes termination.
G2.01	This shows the date and time the PDM placed the group of statistics records in the statistics file.

Sample of the system statistics page

A system statistics page similar to the one shown below is the first set of statistics in each group report. These PDM system-level statistics cover the time from when the statistics were last reset (S1.01) until they were placed in the statistics file (G2.01). They are not specific to the task that issued the RSTAT command and caused the statistics to be placed in the file.

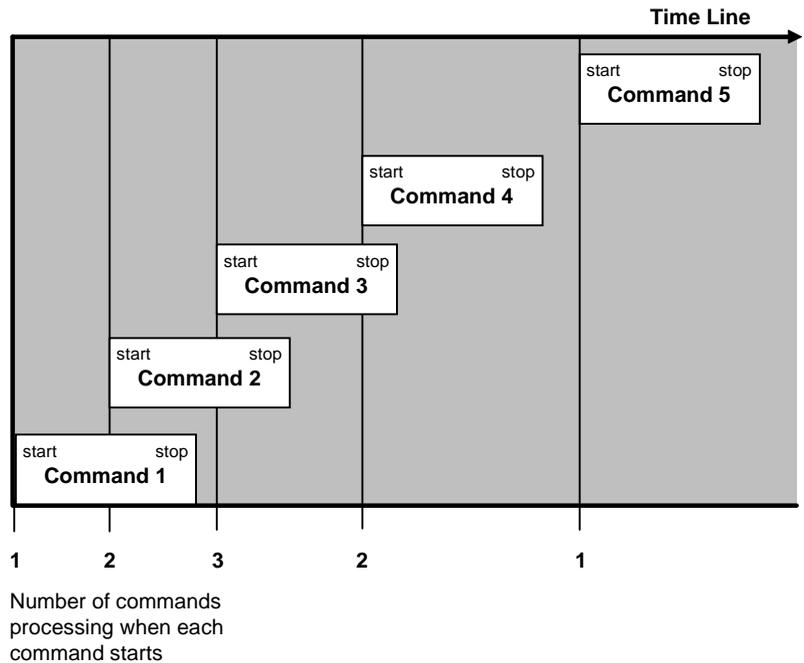
For more information on each statistic, see the table following this sample.

FUNCTION = EXECUTION STATISTICS			
GROUP 1.1			
SYSTEM STATISTICS			
S1.01	DATE AND TIME STATISTICS WERE LAST RESET	NOV. 12, 1992	10:35:24
S1.02	FIRST GROUP OF EXECUTION STATISTICS SINCE RESET?	YES	
S2.01	TOTAL TASKS		1
S2.02	MAXIMUM CONCURRENT TASKS		1
S3.01	CURRENT RECORD HOLDING ENTRIES IN USE		0
S3.02	MAXIMUM RECORD HOLDING ENTRIES USED		0
S3.03	CURRENT MONITOR ENTRIES (READ LOCKS) IN USE		0
S3.04	MAXIMUM MONITOR ENTRIES (READ LOCKS) USED		0
S4.01	TOTAL READ COMMANDS		516 99.61%
S4.02	TOTAL UPDATE COMMANDS		0 0.00%
S4.03	TOTAL ADD AND DELETE COMMANDS		0 0.00%
S4.04	TOTAL OTHER COMMANDS		2 0.39%
S5.01	TOTAL COMMANDS ISSUED TO THE PDM		518
S5.02	MAXIMUM NUMBER OF COMMANDS AT COMMAND STARTS		1
S5.03	SUM OF COMMANDS AT COMMAND STARTS		518
S5.04	AVERAGE NUMBER OF COMMANDS AT COMMAND STARTS	(S5.03/S5.01)	1.00AVG.
S6.01	TOTAL ELAPSED TIME ON COMMANDS ISSUED TO THE PDM		00:00:00.000
S6.02	AVERAGE ELAPSED TIME PER COMMAND ISSUED TO THE PDM	(S6.01/S5.01)	00:00:00.000
S6.03	MAXIMUM ELAPSED TIME FOR ANY COMMAND ISSUED TO THE PDM		00:00:00.000
S7.01	NUMBER OF TIMES PDM WAS INACTIVE		757
S7.02	AVERAGE NUMBER OF TIMES PDM WAS INACTIVE PER COMMAND	(S7.01/S5.01)	1.46AVG.
S8.01	AMOUNT OF TIME PDM WAS ACTIVE	(HH:MM:SS.SS)	00:00:13.717 53.00%
S8.02	AMOUNT OF TIME PDM WAS INACTIVE		00:00:12.166 47.00%
S8.03	TOTAL PDM TIME	(S8.01+S8.02)	00:00:25.883 100.00%
S8.04	AVERAGE AMOUNT OF TIME PDM WAS ACTIVE PER COMMAND	(S8.01/S5.01)	00:00:00.026 S9.01 TOTAL
BYTES OF MEMORY (IN K)		1,460	
S9.02	TOTAL BYTES OF MEMORY USED AT PRESENT TIME (IN K)		464 31.78%
S9.03	MAXIMUM BYTES OF MEMORY USED (IN K)		464 31.78%
S9.04	THRESHOLD MEMORY		90%
S10.01	TOTAL BYTES OF XA MEMORY (IN K)		6.144
S10.02	TOTAL BYTES OF XA MEMORY USED AT PRESENT TIME (IN K)		148 2.41%
S10.03	MAXIMUM BYTES OF XA MEMORY USED (IN K)		148 2.41%

Statistics identifier	Additional information
S1.01	The date and time the memory storage area (where statistics are accumulated) was last reset.
S1.02	First set of execution statistics since reset? Yes or No.
S2.01	The total number of SINON commands issued to the PDM. It includes sign-ons that failed and sign-ons used to reconnect an active task after a task or system failure (task-level recovery only).
S2.02	The maximum number of tasks that were signed on at any one time.
S3.01	The number of record holding entries in use by the PDM.
S3.02	The highest number of record holding entries with read locks in use by the PDM since statistics were last reset.
S3.03	The current number of monitored entries with read locks in use by the PDM.
S3.04	The highest number of monitored entries with read locks in use by the PDM since statistics were last reset.
S4.01	The number of times read commands were issued to the PDM by PDM interfaces. The statistic includes commands that failed in the PDM, but does not include commands issued to a PDM interface and not passed to the PDM.
S4.02	The number of times write commands were issued to the PDM by PDM interfaces. The statistic includes commands that failed in the PDM, but does not include commands issued to a PDM interface and not passed to the PDM.
S4.03	The number of times add and delete commands were issued to the PDM by PDM interfaces. The statistic includes commands that failed in the PDM, but does not include commands that the PDM interface did not pass to the PDM.
S4.04	The number of commands other than reads, writes, adds, or deletes that the PDM interfaces issued to the PDM. The statistic includes commands that failed in the PDM, but does not include commands that the PDM interfaces did not pass to the PDM.
S5.01	The number of commands that PDM interfaces issued to the PDM. The statistic does not include commands the PDM interfaces did not pass to the PDM.
S5.02	The maximum number of commands processing simultaneously (see the figure following this table). A command is processing if it has been passed to the PDM and not yet returned to the interface.

Statistics identifier	Additional information
S5.03	Sum of commands processing when a command starts processing, including commands starting and already processing. Use this statistic to calculate the average number of commands being processed by the PDM when commands start processing (see the figure following this table).
S5.04	The sum of commands at command starts (S5.03) divided by the total number of commands issued to the PDM (S5.01).
S6.01	The total elapsed time of all commands. The elapsed time for a single command begins when the PDM interface issues the command to the PDM and ends when the PDM interface receives notice that the command completed. This statistic is given in hours, minutes, seconds, and milliseconds.
S6.02	The elapsed time on commands issued to the PDM (S6.01) divided by total commands issued (S5.01). This statistic is given in hours, minutes, seconds, and milliseconds.
S6.03	The maximum elapsed time to process a single command out of all the commands in this group report. This statistic is given in hours, minutes, seconds, and milliseconds.
S7.01	The number of times the PDM issued an operating system wait because there was no processing to do.
S7.02	The average number of times the PDM was inactive per command. This value is calculated by dividing the number of times the PDM was inactive (S7.01) by the total commands issued to the PDM (S5.01).
S8.01	The amount of time, in hours, minutes, seconds, and milliseconds, that the PDM was executing.
S8.02	The amount of time, in hours, minutes, seconds, and milliseconds, that the PDM was in an operating system wait mode.
S8.03	Total amount of active (S8.01) and inactive PDM time (S8.02), in hours, minutes, seconds, and milliseconds.
S8.04	The average time the PDM took to execute a command, in hours, minutes, seconds, and milliseconds. This number is calculated by dividing the amount of time the PDM was active (S8.01) by the number of commands issued by the PDM (S5.01).
S9.01	The amount of memory specified to the PDM to be allocated as work space. Refer to the <i>SUPRA Server PDM and Directory Administration Guide (OS/390 & VSE)</i> , P26-2250, for additional information.

Statistics identifier	Additional information
S9.02	The amount of memory that is currently being used.
S9.03	The highest amount of memory used (S9.02) since statistics were last reset.
S9.04	The percentage (S9.02/S9.01*100) of allocated memory used at which the PDM will begin releasing noncritical memory. Refer to the <i>SUPRA Server PDM and Directory Administration Guide (OS/390 & VSE)</i> , P26-2250, for additional information.
S10.01	The amount of memory specified to the PDM to be allocated as work space. Refer to the <i>SUPRA Server PDM and Directory Administration Guide (OS/390 & VSE)</i> , P26-2250, for additional information.
S10.02	The amount of memory above the 16-megabyte line that is currently being used.
S10.03	The highest amount of memory above the 16-megabyte line used (S10.02) since statistics were last reset.



The PDM counts the number of commands processing each time a new command starts. The PDM counts the command that is starting as the first command. In this example, only one command, Command 1, is processing when Command 1 starts. When Command 2 starts, two commands are processing: Commands 1 and 2. When Command 3 starts, three commands are processing. However, when Command 4 starts, Commands 1 and 2 have finished processing, so only two commands are processing: Commands 3 and 4. When Command 5 starts, it is the only command processing.

To arrive at the maximum number of commands processing when each command starts (S5.02), the PDM picks the highest number from the figures along the base line (3).

To arrive at the sum of commands processing when commands start (S5.03), the PDM adds the numbers along the base line ($1+2+3+2+1=9$).

Sample of the file statistics page

A file statistics page, like the one shown below, is included in the group report. There is one page for each file defined to the PDM when the statistics record group was placed in the statistics file. The files defined to the PDM include all database, Directory, log, and statistics files. The statistics describe the activity for the file from the time the statistics were last reset (S1.01) until the statistics record group was placed in the statistics file (G2.01). These statistics are at the PDM file level, and, therefore, are not specific to the task which issued the RSTAT command.

For more information on each statistic, see the table following this sample.

FUNCTION = EXECUTION STATISTICS			
			GROUP 1.10
FILE STATISTICS			
FOR FILE M002			
FILE TYPE	PRIMARY	LOGICAL RECORD LENGTH	61
FILE CODED	NO	BLOCK SIZE	9,394
FILE DDNAME	M002	BLOCKS PER TRACK	2
ACCESS METHOD	BDAM	RECORDS PER BLOCK	154
BUFFER POOL	USRM	TOTAL LOGICAL RECORDS	616
		CONTROL INTERVAL SIZE	0
F1.01	TOTAL LOGICAL READS		4
F1.02	TOTAL PHYSICAL READS		3
F1.03	TOTAL IN-MEMORY HITS	(F1.01-F1.02)	1
F1.04	TOTAL IN-MEMORY HITS ON UPDATED BUFFER		1
F1.05	TOTAL PHYSICAL UPDATES FORCED BY A PHYSICAL READ		0
LOGICAL READS PER PHYSICAL READ		(F1.01/F1.02)	1.33AVG.
F2.02	% OF LOGICAL READS WHICH WERE IN-MEMORY HITS	((F1.03/F1.01)*100)	25.00%
F2.03	% OF IN-MEMORY HITS WHICH WERE TO AN UPDATED BUFFER	((F1.04/F1.03)*100)	100.00%
F2.04	% OF PHYSICAL READS FORCING A PHYSICAL UPDATE	((F1.05/F1.02)*100)	0.00%
F3.01	TOTAL LOGICAL UPDATES		1
F3.02	TOTAL PHYSICAL UPDATES		1
F3.03	TOTAL MULTIPLE LOGICAL UPDATES TO THE SAME BUFFER		0
F3.04	AVERAGE LOGICAL UPDATES PER PHYSICAL UPDATE	(F3.01/F3.02)	1.00AVG.
F3.05	% OF PHYSICAL UPDATES WHICH WERE MULTIPLE UPDATES	((F3.03/F3.02)*100)	0.00%
F4.01	TOTAL LOGICAL I/O	(F1.01+F3.01)	5
F4.02	TOTAL PHYSICAL I/O	(F1.02+F3.02)	4
F4.03	AVERAGE LOGICAL I/O PER PHYSICAL I/O	(F4.01/F4.02)	1.25AVG.
F5.01	% OF LOGICAL I/O WHICH WERE LOGICAL READS	((F1.01/F4.01)*100)	80.00%
F5.02	% OF LOGICAL I/O WHICH WERE LOGICAL UPDATES	((F3.01/F4.01)*100)	20.00%
F5.03	% OF PHYSICAL I/O WHICH WERE PHYSICAL READS	((F1.02/F4.02)*100)	75.00%
F5.04	% OF PHYSICAL I/O WHICH WERE PHYSICAL UPDATES	((F3.02/F4.02)*100)	25.00%
F6.01	TOTAL PHYSICAL UPDATE WAITS DUE TO LOG FILE UPDATES		0
F7.01	TOTAL STOLEN LOCKED RECORDS		0
F7.02	TOTAL SUCCESSFUL LOCKS WITHOUT WAITING		2
F7.03	TOTAL SUCCESSFUL LOCKS APTER WAITING		0
F7.04	TOTAL SUCCESSFUL LOCKS	(F7.01+F7.02+F7.03)	2
F7.05	TOTAL FAILED LOCKS WITHOUT WAITING		0
F7.06	TOTAL FAILED LOCKS APTER WAITING		1
F7.07	TOTAL EMBRACES DETECTED		0
F7.08	TOTAL FAILED LOCKS	(F7.05+F7.06+F7.07)	1
F8.01	TOTAL LOCKS REQUESTED	(F7.04+F7.08)	3
F8.02	% OF TOTAL LOCKS WHICH WERE SUCCESSFUL	((F7.04/F8.01)*100)	66.67%
F8.03	% OF TOTAL LOCKS WHICH FAILED	((F7.08/F8.01)*100)	33.33%

Statistics identifier	Additional information
F1.01	The number of times a record was logically read.
F1.02	The number of times a record was physically read.
F1.03	The number of times a record was logically read and was already in memory (F1.01–F1.02).
F1.04	The number of logical reads that found the desired block of data in a updated storage buffer.
F1.05	The number of forced physical updates that occurred because a logical read required a buffer for a physical read.
F2.01	The number of logical reads (F1.01) divided by the number of physical reads (F1.02).
F2.02	The percentage of logical reads (F1.01) that were in-memory hits (F1.03).
F2.03	The percentage of in-memory hits (F1.03) that were hits on an updated storage buffer (F1.04).
F2.04	The number of physical reads (F1.02) that forced a physical write to obtain a buffer (F1.05).
F3.01	The number of times a record was logically updated. A record is logically updated when it is logically added to the file, deleted from the file, or changed in the file.
F3.02	The number of times the contents of a buffer were physically written. For Task and System Log Files, F3.02 may exceed F3.01 because a single logical record may span several physical blocks. This can cause several buffers to be physically written for a single logical write.
F3.03	The number of logical updates to storage buffers that have already been updated.
F3.04	The number of logical updates (F3.01) divided by the number of physical updates (F3.02).
F3.05	The number of updates to previously updated buffers (F3.03) as a percentage of physical updates (F3.02).
F4.01	The number of logical read and update (change, add, or delete) commands received by the PDM.
F4.02	The number of physical read and update operations performed by the PDM.
F4.03	The number of logical I/O transactions (F4.01) divided by the number of physical I/O transactions (F4.02).

Statistics identifier	Additional information
F5.01	The logical read (F1.01) commands as a percentage of logical I/O commands (F4.01).
F5.02	The logical update (F3.01) commands as a percentage of logical I/O commands (F4.01).
F5.03	The physical reads (F1.02) as a percentage of physical I/Os (F4.02).
F5.04	The physical updates (F3.02) as a percentage of physical I/Os (F4.02).
F6.01	The number of times physical writes to a database file were delayed because a System or Task Log File block had to be physically written first. This statistic is 0 for files other than primary, related, and index files.
F7.01	The number of times ownership of a held record was changed from one task to another without the first task voluntarily releasing the record. This can occur only when task logging is set to NO. Refer to the <i>SUPRA Server PDM and Directory Administration Guide (OS/390 & VSE)</i> , P26-2250, for additional information.
F7.02	The number of times a request to lock a record was immediately granted.
F7.03	The number of times a request to lock a record was granted after waiting for the record to become available.
F7.04	The total number of times a request to lock a record was granted.
F7.05	The number of times a request to lock a record was immediately denied.
F7.06	The number of times a request to lock a record was denied after exceeding the TP monitor delay time or the batch delay time. Refer to the <i>SUPRA Server PDM Directory Online User's Guide (OS/390 & VSE)</i> , P26-1260, for additional information.
F7.07	The number of times a request to lock a record was denied because granting the request would cause an unresolvable locking conflict.
F7.08	The total number of times a request to lock a record was denied.
F8.01	The total number of lock requests.
F8.02	The percentage of lock requests (F8.01) that were granted (F7.04).
F8.03	The percentage of lock requests (F8.01) that were denied (F7.08).

Sample of the file statistics totals for group

A file statistics totals page, like the one shown below, is included in each group report following the individual file statistic pages. Except for T1.00 and T6.02–T6.05, these statistics summarize the corresponding statistics on the individual file statistics pages. These statistics summarize PDM activity at the file level from the time the statistics were last reset (S1.01) until the statistics record group was placed in the statistics file (G2.01). These statistics summarize Directory files, log files, and statistics files in addition to database files. They are not specific to the task that issued the RSTAT command.

For more information on each statistic, see the table following this sample.

FUNCTION = EXECUTION STATISTICS			
FILE STATISTICS			GROUP 1.21
TOTALS FOR GROUP 1			
T1.00	NUMBER OF FILES FOR WHICH STATISTICS WERE ACCUMULATED		19
T1.01	TOTAL LOGICAL READS		885
T1.02	TOTAL PHYSICAL READS		67
T1.03	TOTAL IN-MEMORY HITS	(T1.01-T1.02)	818
T1.04	TOTAL IN-MEMORY HITS ON UPDATED BUFFER		417
T1.05	TOTAL PHYSICAL UPDATES FORCED BY A PHYSICAL READ		1
T2.01	AVERAGE LOGICAL READS PER PHYSICAL READ	(T1.01/T1.02)	13.21AVG.
T2.02	% OF LOGICAL READS WHICH WERE IN-MEMORY HITS	((T1.03/T1.01)*100)	92.43%
T2.03	% OF IN-MEMORY HITS WHICH WERE TO AN UPDATED BUFFER	((T1.04/T1.03)*100)	50.98%
T2.04	% OF PHYSICAL READS FORCING A PHYSICAL UPDATE	((T1.05/T1.02)*100)	1.49%
T3.01	TOTAL LOGICAL UPDATES		42
T3.02	TOTAL PHYSICAL UPDATES		123
T3.03	TOTAL MULTIPLE LOGICAL UPDATES TO THE SAME BUFFER		0
T3.04	AVERAGE LOGICAL UPDATES PER PHYSICAL UPDATE	(T3.01/T3.02)	0.34AVG.
T3.05	% OF PHYSICAL UPDATES WHICH WERE MULTIPLE UPDATES	((T3.03/T3.02)*100)	0.00%
T4.01	TOTAL LOGICAL I/O	(T1.01+T3.01)	927
T4.02	TOTAL PHYSICAL I/O	(T1.02+T3.02)	190
T4.03	AVERAGE LOGICAL I/O PER PHYSICAL I/O	(T4.01/T4.02)	4.88AVG.
T5.01	% OF LOGICAL I/O WHICH WERE LOGICAL READS	((T1.01/T4.01)*100)	95.47%
T5.02	% OF LOGICAL I/O WHICH WERE LOGICAL UPDATES	((T3.01/T4.01)*100)	4.53%
T5.03	% OF PHYSICAL I/O WHICH WERE PHYSICAL READS	((T1.02/T4.02)*100)	35.26%
T5.04	% OF PHYSICAL I/O WHICH WERE PHYSICAL UPDATES	((T3.02/T4.02)*100)	64.74%
T6.01	TOTAL PHYSICAL UPDATE WAITS DUE TO LOG FILE UPDATES		0 0.00%
T6.02	TOTAL LOGICAL UPDATE WAITS DUE TO LOG FILE UPDATES		1 100.00%
T6.03	TOTAL UPDATE WAITS DUE TO LOG FILE UPDATES	(T6.01+T6.02)	1 100.00%
T6.04	TOTAL NUMBER OF UPDATES TO THE LOG FILES(S)		29
T6.05	AVERAGE LOGICAL LOG FILE UPDATES PER LOG FILE WAIT	(T6.04/T6.03)	29.00AVG.
T7.01	TOTAL STOLEN LOCKED RECORDS		0
T7.02	TOTAL SUCCESSFUL LOCKS WITHOUT WAITING		2
T7.03	TOTAL SUCCESSFUL LOCKS AFTER WAITING		0
T7.04	TOTAL SUCCESSFUL LOCKS	(T7.01+T7.02+T7.03)	2
T7.05	TOTAL FAILED LOCKS WITHOUT WAITING		0
T7.06	TOTAL FAILED LOCKS AFTER WAITING		1
T7.07	TOTAL EMBRACES DETECTED		0
T7.08	TOTAL FAILED LOCKS	(T7.05+T7.06+T7.07)	1
T8.01	TOTAL LOCKS REQUESTED	(T7.04+T7.08)	3
T8.02	% OF TOTAL LOCKS WHICH WERE SUCCESSFUL	((T7.04/T8.01)*100)	66.67%
T8.03	% OF TOTAL LOCKS WHICH FAILED	((T7.08/T8.01)*100)	33.33%

Statistics identifier	Additional information
T1.00	The number of files defined to the PDM when the statistics record group was placed in the statistics file.
T1.01	The number of times a record was logically read.
T1.02	The number of times a record was physically read.
T1.03	The number of times a record was logically read and was already in memory.
T1.04	The number of logical reads that found the desired block of data in a storage buffer.
T1.05	The number of forced physical updates that occurred because a logical read required a buffer for a physical read.
T2.01	The number of logical reads (T1.01) divided by the number of physical reads (T1.02).
T2.02	The percentage of logical reads (T1.01) that were in-memory hits (T1.03).
T2.03	The percentage of in-memory hits (T1.03) that were hits on an updated buffer (T1.04).
T2.04	The number of physical reads (T1.02) that forced a physical write to obtain a buffer (T1.05).
T3.01	The number of times a record was logically updated. A record is logically updated when it is logically added to the file, deleted from the file, or changed in the file.
T3.02	The number of times the contents of a buffer were physically written.
T3.03	The number of logical updates to storage buffers that have already been updated.
T3.04	The number of logical updates (T3.01) divided by the number of physical updates (T3.02).
T3.05	The number of updates to previously updated buffers (T3.03) as a percentage of physical updates (T3.02).
T4.01	The number of logical read and update (change, add, or delete) commands received by the PDM.
T4.02	The number of times physical read and update operations were performed by the PDM.
T4.03	The number of logical I/O transactions (T4.01) divided by the number of physical I/O transactions (T4.02).
T5.01	The logical read (T1.01) commands as a percentage of logical I/O commands (T4.01).

Statistics identifier	Additional information
T5.02	The logical update (T3.01) commands as a percentage of logical I/O commands (T4.01).
T5.03	The physical read (T1.02) commands as a percentage of physical I/Os (T4.02).
T5.04	The physical updates (T3.02) as a percentage of physical I/Os (T4.02).
T6.01	The number of times physical writes to a database file were delayed because a System or Task Log File block had to be physically written first.
T6.02	The number of times a log file record (or when spanning records, a log record segment) could not be logically written because a System or Task Log File block had to be physically written.
T6.03	The number of times logical writes to a log file and physical writes to a database file were delayed because a System or Task Log File block had to be written (T6.01 + T6.02).
T6.04	The number of logical writes to the System and Task Log Files. When spanning records, each block in the record is considered a separate logical write.
T6.05	The number of records logically written to the log file(s) (T6.04) divided by the number of delays (T6.03).
T7.01	The number of times ownership of a held record was changed from one task to another without the first task voluntarily releasing the record. This can occur only when task logging is set to NO. Refer to the <i>SUPRA Server PDM and Directory Administration Guide (OS/390 & VSE)</i> , P26-2250, for additional information.
T7.02	The number of times a request to lock a record was immediately granted.
T7.03	The number of times a request to lock a record was granted after waiting for the record to become available.
T7.04	The total number of times a request to lock a record was granted.
T7.05	The number of times a request to lock a record was immediately denied.
T7.06	The number of times a request to lock a record was denied after exceeding the TP monitor delay time or the batch delay time. Refer to the <i>SUPRA Server PDM Directory Online User's Guide (OS/390 & VSE)</i> , P26-1260, for additional information.
T7.07	The number of times a request to lock a record was denied because granting the request would cause an unresolvable locking conflict.
T7.08	The total number of times a request to lock a record was denied.
T8.01	The total number of lock requests.
T8.02	The percentage of lock requests (T8.01) that were granted (T7.04).
T8.03	The percentage of lock requests (T8.01) that were denied (T7.08).

Sample of the termination page

The termination page appears after at least one group report.

```
FUNCTION = EXECUTION STATISTICS

EXECUTION STATISTICS
TERMINATION

PDM NAME                EE73EX01
BOOT SCHEMA             CINDIRSC
BOOT ENVIRONMENT DESCRIPTION CINDIRTU
USER SCHEMA             UTILSCHM
USER ENVIRONMENT DESCRIPTION UTED50EX

TOTAL PDM EXECUTION STATISTICS GROUPS      2
TOTAL RECORDS ON STATISTICS FILE          56
```

17

Coding the Inter-Directory Copy utility

Coding the Inter-Directory Copy utility

Use the Inter-Directory Copy utility to copy information from one SUPRA Directory to another. Inter-Directory Copy creates batch Directory Maintenance transactions from the source Directory. These transactions can be passed to batch Directory Maintenance to be applied to a target Directory. In the target Directory, Directory Maintenance will mark as inconsistent every copied Access Set, Conceptual Schema, File, Internal Record, Logical View, Relation, Schema, or Secondary Key entity.

This utility does not use UCL. [“Executing the Inter-Directory Copy utility”](#) on page 445 describes how to execute this utility.

This utility copies the following entities in the hierarchy shown and includes all attribute data, relationships, and short and long text:

- ◆ Schema
 - External fields
 - Files
 - Internal Records
 - Physical Fields
 - Secondary Keys
 - Key Codes
 - Environment Descriptions
 - Buffer Pools
 - Log Groups
 - Logical Views
 - Access Sets
- ◆ Security Groups
 - Maintenance Restrictions
- ◆ Conceptual Schemas
 - Relation
 - Primary Key
 - Foreign Keys
 - Attributes
- ◆ Domain
- ◆ Users
 - Procedures

Before executing the batch Directory Maintenance transactions, you must enter a valid ID and password. When processing the transactions, you may use any of the run options such as +SYNTAX to check syntax. At this point, you can optionally edit the transactions prior to executing batch Directory Maintenance. This utility can be run to generate transactions to be held and input later, or for logical back-up. For details, refer to *SUPRA Server PDM Directory Batch User's Guide (OS/390 & VSE)*, P26-1261.

OS/390

(OS/390 only) You cannot use members of a partitioned data set to hold the output from DIRCOPYP. Write the output data sets as sequential data sets, and then use the IEBCOPY utility to load them onto the partitioned data sets.

Coding the input statements for the Inter-Directory Copy utility

When you submit input statements for the Inter-Directory Copy utility, they are echoed, as they appear, to your transaction file. You must submit them in the following order:

1. The batch Directory Maintenance statements are optional and can go in any order. The only exception is the +SIGNON statement, which is required and must go last. You can code it only once.
2. If you code the user input statement, you must put it next.
3. If you code the security group statement, you must code it after the user input statement.
4. If you code the schema input statement, you must code it after the user input statement.
5. If you code the conceptual schema input statement, you must code it after the schema input statement.
6. You can code the copy table, edit mask, and domain input statements anywhere. Their order is not important.

If you code statements in the wrong order, you receive error messages. For example, you must copy a user before you copy the user's security group. That is, you must identify the users before you can give them security clearance.

While the order of the statements is important, the number is not. There is no limit to the number of statements you can code. For example, you can code several users and several security groups. However, if you do not code any input statements, the utility copies nothing. The following table shows the relationship between your input statements and the transaction file to which they are echoed.

Statement input	Ordering	Notes
BDM + statements	You must put all the statements you use in the first group.	+SIGNON is the only required statement.
COPY-TA	The order does not matter.	Optional
COPY-EM	The order does not matter.	Optional
COPY-US	Must precede COPY-SG and COPY-SC.	Optional
COPY-SG	Must follow COPY-US.	Optional
COPY-SC	Must follow COPY-US.	Optional
COPY-CS	Must follow COPY-SC.	Optional
COPY-DM	Must follow COPY-SC.	Optional

Optional input statements

Positions 1–7

Any of the batch Directory Maintenance run option definition statements:

+DATA

+NODATA

+PAGING

+NOPAGING

+SEQUENCE

+NOSEQUENCE

+ERRCONT

+NULL

+SYNTAX

Description *Optional.* The run option definition statements offer processing and printing options. For example, you can check syntax or continue processing after an error.

Consideration These statements are for batch Directory Maintenance processing. The Inter-Directory Copy utility simply includes them among the transactions created. You should code these statements before any other input statements. For more information about run option definition statements, refer to the *SUPRA Server PDM Directory Batch User's Guide (OS/390 & VSE)*, P26-1261.

Signon input statement

Positions 1–7

+SIGNON

Description *Required.* Signs on the user for this utility and for batch Directory Maintenance.

Consideration The +SIGNON statement becomes part of the output of the Inter-Directory Copy utility.

Positions 9–38

user-id

Description *Required.* Identifies an existing user on the target Directory.

Format 1–30 alphanumeric or special characters (#, \$, and -)

Consideration The ID you code in the User-ID field must exist on the Directory and must be related to the Directory-Copy Logical View. (You may use the RELATE command to establish this relationship.) For security, you must be an authorized user (DBA) to copy Security Group and User entities.

Positions 40–69

password

Description *Required.* Indicates the password assigned to user.

Format 1–30 alphanumeric characters

Copy table input statement

Positions 1–7

COPY-TA

Description *Optional.* Copies the specified Table.

Positions 9–38

table name

Description *Required.* Identifies the Table you want to copy from the source Directory.

Format 1–30 alphanumeric or special characters (#, \$, and -)

Edit mask input statement

Positions 1–7

COPY-EM

Description *Optional.* Copies the specified Edit Mask.

Positions 9–38

edit mask name

Description *Required.* Identifies the Edit Mask you want to copy from the source Directory.

Format 1–30 alphanumeric or special characters (#, \$, and -)

User input statement

Positions 1–7

COPY-US

Restriction If you do not sign on as a DBA, the utility ignores this statement.

Description *Optional.* Copies the specified User.

Consideration Must precede all COPY-SC statements.

Positions 9–38

source-user-name

Description *Required.* Identifies the name of the User you want to copy from the source Directory.

Format 1–30 alphanumeric or special characters (#, \$, and -)

Positions 40–69

target-user-name

Description *Optional.* Identifies the User name for the target Directory.

Format 1–30 alphanumeric or special characters (#, \$, and -)

Consideration When you change the name of a user, the old password remains in effect. To change the password, edit the batch Directory Maintenance transactions that result from this utility or process an additional transaction in a subsequent batch Directory Maintenance run.

Security group input statement

Positions 1–7

COPY-SG

- Restriction** If you do not sign on as a DBA, the utility ignores this statement.
- Description** *Optional.* Copies a Security Group and all child Maintenance Restrictions.
- Consideration** Must follow the COPY-US statement(s).
-

Positions 9–38

security-group-name

- Description** *Required.* Identifies the Security Group you want to copy from the source Directory.
- Format** 1–30 alphanumeric or special characters (#, \$, and -)
- Consideration** Maintenance Restrictions for the Security Group are also copied.
-

Positions 40–69

target-security-group-name

- Description** *Optional.* Identifies the Security Group name for the target Directory.
- Format** 1–30 alphanumeric or special characters (#, \$, and -)

Schema input statement

Positions 1–7

COPY-SC

Description *Optional.* Copies a schema and all related entities.

Consideration Must follow the COPY-US statement(s).

Positions 9–38

source-schema-name

Description *Required.* Identifies the schema you want to copy from the source Directory.

Format 1–8 alphanumeric or special characters (#, \$, and @)

Positions 40–69

target-schema-name

Description *Optional.* Identifies the schema name for the target Directory.

Format 1–8 alphanumeric or special characters (#, \$, and @)

Conceptual schema input statement

Positions 1–7

COPY-CS

Description *Optional.* Copies a Conceptual Schema and generates all relationships between Schema and Conceptual Schema.

Consideration Must follow the COPY-SC statement(s).

Positions 9–38

source-conceptual-schema-name

Description *Required.* Identifies the Conceptual Schema you want to copy from the source Directory.

Format 1–30 alphanumeric or special characters (#, \$, and -)

Positions 40–69

target-conceptual-schema-name

Description *Optional.* Identifies the Conceptual Schema name for the target Directory.

Format 1–30 alphanumeric or special characters (#, \$, and -)

Consideration When you code COPY-CS, this utility generates AD DM batch Directory Maintenance statements, which add the Domains needed for the Attributes. If you copy multiple Conceptual Schemas that are related to a common Domain, the utility generates duplicate AD DM statements. The second and subsequent AD DM transactions cause Directory Maintenance to return the error message, "Entity Already Exists." For Directory Maintenance to continue processing after this error, you must code the +ERRCONT run option definition statement.

Domain input statement

Positions 1–7

COPY-DM

Description *Optional.* Copies a Domain.

Positions 9–38

source-domain

Description *Required.* Identifies the Domain you want to copy from the source Directory.

Format 1–30 alphanumeric or special characters (#, \$, and -)

Positions 40–69

target-domain

Description *Optional.* Identifies the Domain name for the target Directory.

Format 1–30 alphanumeric or special characters (#, \$, and -)

Executing the Inter-Directory Copy utility

OS/390

To execute the Inter-Directory Copy utility in OS/390, submit sample JCL member TXJDRCPY. This member uses the cataloged procedure TISDMCPY. All sample JCL members are in the SUPRA product library on your installation tape.

The Inter-Directory Copy utility writes the output (the Batch Directory Maintenance transactions) to a data set with the dname LTRX. The utility produces a listing that shows all the input records and any error messages.



You cannot use members of a partitioned data set to hold the output from this utility. Direct the output to sequential data sets and, if you want, use the IBM IEBCOPY utility to load them into partitioned data set(s).

VSE

To execute the Inter-Directory Copy utility in VSE, submit the sample JCL member TXJDRCPY. This member uses the cataloged procedure TISDMCPY. All sample JCL members are located in the SUPRA product library on your installation tape.

To create the trigger file, code the input statements as OBJMAINT input in the first step of the sample.

The Inter-Directory Copy utility writes the output (the Batch Directory Maintenance transactions) to a data set with the filename LTRX. The utility produces a listing that shows all the input records and any error messages.

Inter-Directory Copy example

Sample input for the Inter-Directory Copy utility:

```
+SIGNON userid                                password
+NOPAGING
+NODATA
+ERRCONT
+NOSEQUENCE
COPY-TA CSIA
COPY-TA CSIB
COPY-TA CSIC
COPY-TA CSID
COPY-TA CSIE
COPY-EM A
COPY-EM B
COPY-EM C
COPY-EM D
COPY-EM E
COPY-US CINCOM                                NEW-CINCOM
COPY-SC CSISCH20                              NEWSCH20
```

18

Coding the Recover, Restore, and Log-Print utilities

Coding the Recover, Restore, and Log-Print functions

When you cannot use the Task Log File to recover the database, you can use the Recover function to back off updates to the last commit. The Recover function has two phases. In the analysis phase, the Recover function reads the System Log File from the beginning, collecting and optionally printing information. In the image application phase, the Recover function reads the System Log File backwards from the end, applying before-images to the database. The Recover function stops reading and applying images either at the last commit for each task or at the start of the file.

When PDM files are lost or damaged, you can use the Restore function to reapply updates to the last commit. You must reload the affected files from your backup copies before running the Restore function. The Restore function has two phases. In the analysis phase, the Restore function reads the System Log File from the beginning, collecting and optionally printing information. In the image application phase, the Restore function reads the System Log File forward from the beginning, applying after-images to the database. The Restore function stops reading and applying images either at the last commit for each task or at the end of the file.

You can use the Log-Print function to print selected information from the System Log File without updating the database. The Log-Print function has one phase, the analysis phase, which corresponds to the analysis phase of the Recover and Restore functions. The Log-Print function reads the System Log File from the beginning, collecting and optionally printing information.

A UCL program can invoke a combination of the Recover, Restore, and Log-Print functions. A UCL program that invokes a combination of these three functions may not invoke any other function.

For additional information on PDM system logging and the Recover, Restore, and Log-Print functions, refer to the *SUPRA Server PDM Logging and Recovery Guide (OS/390 & VSE)*, P26-2223.

Coding the UCL for the Recover and Restore functions

After you code the control section as shown in “[Coding the control section](#)” on page 57, code the Recover and Restore functions as shown in the following format. For UCL examples, see “[Examples](#)” on page 471.

```

FUNCTION ( { RECOVER
            RESTORE } )

STATE ( { LOG - END
          LOG - BEGIN
          LAST - COMMIT } )

[ STANDARD - EXIT (exit-name) ]

OPEN - FILE ( [ INITIAL
               DYNAMIC ] )

[ STATISTIC ( [ ALL
               BASE
               NONE ] ) ]

FILE ( { ALL
        file-name } ) ...

[ RRN - RANGE ( { low-rrn
                  -high-rrn
                  low-rrn - high rrn } )

  [D' low-dec-key' ] [-D' hi-dec-key' ]
  KEY - RANGE ( [x' low-hex-key' ] [-X' hi-hex-key' ] )
  [C' low-chr-key' ] [-C' hi-chr-key' ] ]

```

FUNCTION ({ **RECOVER** }
 { **RESTORE** })

Description *Required.* Invokes the function.

STATE ({ **LOG - END** }
 { **LOG - BEGIN** })
 { **LAST - COMMIT** }

Description *Required.* Indicates the point to which you want to recover the database.

Options

LOG-END	Recover the database to the logical end of the System Log File.
LOG-BEGIN	Recover the database to the logical beginning of the System Log File.
LAST-COMMIT	Recover the database to the last commit point on the System Log File.

Considerations

- ◆ If you are not using Task Level Recovery and you code STATE (LAST-COMMIT), the function recovers database files to the last quiet point on the System Log File.
- ◆ If you are using Task Level Recovery and you code STATE (LAST-COMMIT), the function recovers each task to its own last commit point.

STANDARD-EXIT (*exit-name*)

Description *Optional.* Indicates the name of the exit program you want to invoke. For information on coding exit programs, see “[Writing exit programs](#)” on page 461.

Format 1–8 alphanumeric characters. The first character must be alphabetic.

Considerations

- ◆ If you code this statement, you must put it before the FILE statements.
 - ◆ Your exit program must reside in your execution library.
 - ◆ Only one exit at a time resides in memory. If you code a new exit name in a subsequent function, the function deletes the current exit before it loads the new one.
 - ◆ For information on the Cincom-supplied Log File I/O exit, refer to the [SUPRA Server PDM Logging and Recovery Guide \(OS/390 & VSE\)](#), P26-2223.
-

OPEN - FILE ($\left[\begin{array}{l} \text{INITIAL} \\ \text{DYNAMIC} \end{array} \right]$)

Description *Required.* Indicates when you want the function to open the files to be recovered.

Default INITIAL

Options

INITIAL	Opens the files during initialization of the image application phase.
DYNAMIC	During the image application phase, the function opens each file as it encounters the first image to be applied to the file.

Considerations

- ◆ If you code this statement, you must put it before the FILE statements.
- ◆ If you code OPEN-FILE (DYNAMIC) and the System Log File does not contain any images to be applied to a file, the function does not open the file and the file may still be locked at the end of the function. In that case, use the Unlock function to unlock the file.

```
STATISTIC ( [ ALL  
            [ BASE ]  
            [ NONE ] )
```

Description *Optional.* Indicates which statistics to report.

Default BASE

Options ALL Reports basic and separate statistics on each database file.

BASE Reports basic file information for the entire System Log File.

NONE Does not report statistics.

Consideration If you code this statement, you must put it before the FILE statements.

FILE ({ **ALL**
file-name }) ...

Description	<i>Required.</i> Identifies the file you want the function to recover.				
Format	4 alphanumeric characters. The first character must be alphabetic.				
Options	<table> <tr> <td>ALL</td> <td>Recovers all index, primary, related, and Directory files, but does not recover System Log, Task Log, or Statistics files.</td> </tr> <tr> <td><i>file-name</i></td> <td>Recovers the specified file.</td> </tr> </table>	ALL	Recovers all index, primary, related, and Directory files, but does not recover System Log, Task Log, or Statistics files.	<i>file-name</i>	Recovers the specified file.
ALL	Recovers all index, primary, related, and Directory files, but does not recover System Log, Task Log, or Statistics files.				
<i>file-name</i>	Recovers the specified file.				

Considerations

- ◆ The file name must be in the System Log File.
- ◆ The function does not automatically recover index files when it recovers the data file with which they are associated. To recover index files, you must code FILE (ALL) or code the index file in a FILE statement.
- ◆ When the function recovers an index file, it recovers the secondary keys it contains.
- ◆ You can code FILE (ALL) only one time.
- ◆ You cannot code both FILE (ALL) and FILE (*file-name*). You must use one or the other.
- ◆ You cannot code FILE (*file-name-list*).
- ◆ If you code FILE (), the function does not recover any files.
- ◆ When you code FILE (ALL) or several FILE (*file-name*) statements, the function prints the records in the order it reads them from the System Log File, not in the order you code the files.
- ◆ If you code LIST (ALL) in the control section, the function prints records for all files regardless of the FILE statements you code. However, you must code a FILE statement.
- ◆ You can code the FILE statement and its subordinate statements one or more times.

RRN - RANGE ($\left. \begin{array}{l} \textit{low-rrn} \\ \textit{-high-rrn} \\ \textit{low-rrn - high-rrn} \end{array} \right\}$)

- Restriction** Use this statement only following a FILE statement.
- Description** *Optional.* Applies images to records having relative record numbers within the specified range.
- Format** 1–9 numeric characters for each relative record number (rrn).
- Options**
- | | |
|-------------------------|--|
| <i>low-rrn</i> | Applies images to records having relative record numbers greater than or equal to the specified <i>low-rrn</i> . |
| <i>-high-rrn</i> | Applies images to records having relative record numbers less than or equal to the specified <i>high-rrn</i> . |
| <i>low-rrn-high-rrn</i> | Retrieves records having relative record numbers from <i>low-rrn</i> through <i>high-rrn</i> . |

Considerations

- ◆ During log analysis, the function prints before, after, or function images within this range if you code LIST (BEFORE), LIST (AFTER), or LIST (FUNCTION) in the control section. During image application, the function applies before-images within this range to the database files.
- ◆ If you code both *low-rrn* and *high-rrn*, *low-rrn* must be less than or equal to *high-rrn*.
- ◆ If you code FILE (ALL), the function ignores this statement.
- ◆ This statement is not valid for KSDS or index files.

	[D'low-dec-key'][-D'hi-dec-key']
KEY-RANGE ([X'low-hex-key'][-X'hi-hex-key'])
	[C'low-chr-key'][-C'hi-chr-key']
Restriction	Use this statement only following a FILE statement.
Description	<i>Optional.</i> Applies images to records having keys within the specified range.
Format	1–256 decimal digits for D'low-dec-key' or D'hi-dec-key' 1–512 hexadecimal digits for X'hi-hex-key' or X'low-hex-key' 1–256 character digits for C'hi-chr-key' or C'low-chr-key'
Options	D'low-dec-key' Applies images to records having decimal keys from the low-dec-key to the end of the file. -D'hi-dec-key' Applies images to records having decimal keys from the beginning of the file through the hi-dec-key. 'low-hex-key' Applies images to records having hexadecimal keys from the low-hex-key to the end of the file. -X'hi-hex-key' Applies images to records having hexadecimal keys from the beginning of the file through the hi-hex-key. C'low-chr-key' Applies images to records having character keys from low-chr-key to the end of the file. -C'hi-chr-key' Applies images to records having character keys from the beginning of the file through hi-chr-key.

Considerations

- ◆ During log analysis, the function prints before, after, or function images within this range if you code LIST (BEFORE), LIST (AFTER), or LIST (FUNCTION) in the control section. During image application, the function applies before-images within this range to the database files.
- ◆ If you code both low key and high key, low key must be less than or equal to high key.
- ◆ The key lengths must match the record lengths exactly. You cannot pad or make other length adjustments.
- ◆ If you code FILE (ALL), the function ignores this statement.
- ◆ This statement is valid only for KSDS files.

Coding the UCL for the Log-Print function

After you code the control section as shown in “Coding the control section” on page 57, code the Log-Print function as shown in the following format. For UCL examples, see “Examples” on page 471.

FUNCTION (LOG-PRINT)

[STANDARD-EXIT (*exit-name*)]

[STATISTIC ($\left[\begin{array}{c} \text{ALL} \\ \text{BASE} \\ \text{NONE} \end{array} \right]$)]

FILE ($\left\{ \begin{array}{c} \text{ALL} \\ \text{file-name} \end{array} \right\}$) ...

[RRN - RANGE ($\left\{ \begin{array}{c} \text{low-rrn} \\ \text{-high-rrn} \\ \text{low-rrn - high rrn} \end{array} \right\}$)]

[D' *low-dec-key*'] [-D' *hi-dec-key*']

KEY - RANGE ([x' *low-hex-key*'] [-X' *hi-hex-key*'])

[C' *low-chr-key*'] [-C' *hi-chr-key*']

FUNCTION (LOG-PRINT)

Description *Required.* Invokes the Log-Print function.

STANDARD-EXIT (*exit-name*)

Description *Optional.* Indicates the name of the exit program you want to invoke. For information on coding exit programs, see “[Writing exit programs](#)” on page 461.

Format 1–8 alphanumeric characters. The first character must be alphabetic.

Considerations

- ◆ If you code this statement, you must put it before the FILE statements.
 - ◆ Your exit program must reside in your execution library.
 - ◆ Only one exit at a time resides in memory. If you code a new exit name in a subsequent function, the function deletes the current exit before it loads the new one.
 - ◆ For information on the Cincom-supplied Log File I/O exit, refer to the [SUPRA Server PDM Logging and Recovery Guide \(OS/390 & VSE\)](#), P26-2223.
-

$$\left[\text{STATISTIC} \left(\begin{array}{c} \text{ALL} \\ \text{BASE} \\ \text{NONE} \end{array} \right) \right]$$

Description *Optional.* Indicates which statistics to report.

Default BASE

Options	ALL	Reports basic and separate statistics on each database file.
	BASE	Reports basic file information for the entire System Log File.
	NONE	Does not report statistics.

Consideration If you code this statement, you must put it before the FILE statements.

FILE ({ **ALL**
file-name }) ...

Description	<i>Required.</i> Identifies the file for which you want information printed.
Format	4 alphanumeric characters. The first character must be alphabetic.
Options	<p>ALL Prints information for all index, primary, related, and Directory files, but does not print information for System Log, Task Log, or Statistics files.</p> <p><i>file-name</i> Prints information for the specified file.</p>

Considerations

- ◆ The file name must be in the System Log File.
- ◆ Log-Print does not automatically print information for index files when it prints the data file with which they are associated. To print information for index files, code FILE (ALL) or code the index file in a FILE statement.
- ◆ When Log-Print prints information for an index file, it also prints information for all secondary keys within the file.
- ◆ You can code FILE (ALL) only one time.
- ◆ You cannot code both FILE (ALL) and FILE (*file-name*). You must use one or the other.
- ◆ You cannot code FILE (*file-name-list*).
- ◆ If you code FILE (), the function does not print any file image information.
- ◆ When you code FILE (ALL) or several FILE (*file-name*) statements, the function prints the records in the order it reads them from the System Log File, not in the order you code the files.
- ◆ If you code LIST (ALL) in the control section, the function prints records for all files regardless of the FILE statements you code. However, you must code a FILE statement.
- ◆ You can code the FILE statement and its subordinate statements one or more times.

RRN - RANGE ($\left. \begin{array}{l} \textit{low-rrn} \\ \textit{-high-rrn} \\ \textit{low-rrn - high-rrn} \end{array} \right\})$

Restriction Use this statement only following a FILE statement.

Description *Optional.* Prints records having relative record numbers within the specified range.

Format 1–9 numeric characters for each relative record number (rrn).

Options

<i>low-rrn</i>	Prints records having relative record numbers greater than or equal to the <i>low-rrn</i> .
<i>-high-rrn</i>	Prints records having relative record numbers less than or equal to the <i>high-rrn</i> .
<i>low-rrn-high-rrn</i>	Prints records having relative record numbers from <i>low-rrn</i> through <i>high-rrn</i> .

Considerations

- ◆ If you code LIST (BEFORE), LIST (AFTER), or LIST (FUNCTION) in the control section, the function prints before, after, or function images within this range.
- ◆ If you code both *low-rrn* and *high-rrn*, *low-rrn* must be less than or equal to *high-rrn*.
- ◆ If you code FILE (ALL), the function ignores this statement.
- ◆ This statement is not valid for KSDS or index files.

	[D'low-dec-key'][-D'hi-dec-key']
KEY-RANGE ([X'low-hex-key'][-X'hi-hex-key'])
	[C'low-chr-key'][-C'hi-chr-key']
Description	<i>Optional.</i> Prints records having keys within the specified range.
Format	1–256 decimal digits for D' <i>low-dec-key</i> ' or D' <i>hi-dec-key</i> ' 1–512 hexadecimal digits for X' <i>hi-hex-key</i> ' or X' <i>low-hex-key</i> ' 1–256 character digits for C' <i>hi-chr-key</i> ' or C' <i>low-chr-key</i> '
Options	<p>D'<i>low-dec-key</i>' Prints records having decimal keys from the <i>low-dec-key</i> to the end of the file.</p> <p>-D'<i>hi-dec-key</i>' Prints records having decimal keys from the beginning of the file through the <i>hi-dec-key</i>.</p> <p>X'<i>low-hex-key</i>' Prints records having hexadecimal keys from the <i>low-hex-key</i> to the end of the file.</p> <p>-X'<i>hi-hex-key</i>' Prints records having hexadecimal keys from the beginning of the file through the <i>hi-hex-key</i>.</p> <p>C'<i>low-chr-key</i>' Prints records having character keys from <i>low-chr-key</i> to the end of the file.</p> <p>-C'<i>hi-chr-key</i>' Prints records having character keys from the beginning of the file through <i>hi-chr-key</i>.</p>

Considerations

- ◆ If you code LIST (BEFORE), LIST (AFTER), or LIST (FUNCTION) in the control section, the function prints before, after, or function images within this range.
- ◆ If you code both low key and high key, low key must be less than or equal to high key.
- ◆ The key lengths must match the record lengths exactly. You cannot pad or make other length adjustments.
- ◆ If you code FILE (ALL), the function ignores this statement.
- ◆ This statement is valid only for KSDS files.
- ◆ Recover, Restore, Log-Print Exits

Writing exit programs

You can use exit points from the Recover, Restore, and Log-Print functions to do the following additional tasks:

- ◆ Handle all I/O from the System Log File.
- ◆ Control the application of before images.
- ◆ Control the printing of System Log records.
- ◆ Collect data about the contents of the System Log File.
- ◆ Change the way buffer pools are set up for the PDM's use when the Recover function applies the images.

For information on how exit programs are loaded, how they operate, the languages you can use to write them, and the register conventions you must follow, see “[Inserting exit programs into functions](#)” on page 49. In register 1, for example, you must code the parameter list addresses. For a description of the parameter list addresses, see the following table.

Parameter	Data type	Contents before exit (passed to exit program)	Contents after exit (passed from exit program)
Function Name	16 bytes character	Name of Function	Must be unchanged
Exit Point	4 bytes integer	Exit point number	Must be unchanged
Action Indicator	8 bytes character	bbbbbbbb	bbbbbbbb or other valid values
Data	Variable	Data associated with exit point data	Same data or changed data if permitted

If your exit program changes anything it is not authorized to change, the results are unpredictable.

To use the exit points, see the following sections, which show the exit number, the data associated with an exit, and the valid actions.

For information on the Cincom-supplied Log File I/O exit, refer to the [SUPRA Server PDM Logging and Recovery Guide \(OS/390 & VSE\)](#), P26-2223.

Selecting exit points

To use an exit point, you must know when it occurs in the function, what data is passed, and what actions your program can take at that point. The following table shows when exit points occur. To determine the action you can take, use the tables in the following sections:

- ◆ “Initialization and termination exits” on page 464
- ◆ “Open and close log file exits” on page 465
- ◆ “Analysis phase exits” on page 466
- ◆ “Application phase exits” on page 467

These tables group the exit points by the phase of the function: initialization and termination phases, log open and close phases, analysis and image application phase. (The image application phase exit points do not apply to the Log-Print function.) Once you are familiar with the exit points, use the table in “Valid actions” on page 469 as a reference aid.

Exit	Phase	When exit occurs
1	Initialization	After the function has initialized.
2	Termination	After the function has completed all processing.
3	Analysis	After each read of a log block that did not result in a physical or logical end-of-file.
4	Analysis	After each read of a log block and before the printing of a log block. Note: This exit occurs only if you code LIST (ALL) or LIST (BLOCK) in the UCL control section.
5	Analysis	After each log record has been retrieved.
6	Analysis	After each log record has been retrieved and the print selection criteria have been applied, but before the record is printed. Note: This exit occurs only if the record meets the print criteria you code in the LIST statement in the UCL control section.
7	Image Application	After each read of a log block that did not result in a physical or logical end-of-file.
8	Image Application	After each log record is retrieved.
9	Image Application	After each log image record has been retrieved and the application criteria have been applied, but before the image is applied to the database.

Exit	Phase	When exit occurs
10	Image Application	After each log image record has been applied to the database, but before the image is printed. Note: This exit point occurs only if you code LIST (ALL) or LIST (APPLIED-IMAGES) in the UCL control section. This exit point occurs only if you code LIST (ALL) or LIST (APPLIED-IMAGES) in the UCL control section.
11	Image Application	Before any images are applied to the database. (This exit is inactive by default. Use exit 1 to activate this exit. The other exits are active by default.)
12	Log Open	Before the log is opened.
13	Analysis and Image Application	Before a log block is read.
14	Image Application	Before the log is reset.
15	Log Close	Before the log is closed.

Initialization and termination exits

To take these actions:	Do this:
Leave exit settings as they are	Use exit 1 and action indicator <code>bbbbbbbb</code> . Do not change the data parameter. The initial setting turns on all exits except exit 11.
Turn exits on and off	Use exit 1 and action indicator <code>SETbbbb</code> . Change the appropriate setting(s) in the data parameter. The data parameter is a 15-byte character string containing Y for yes (exit is taken) or N for no (exit is not taken). Note that the log I/O exits 12, 13, and 15 must be all on or all off. For example, you cannot open the file yourself and then let the function read it.
Monitor return code at termination	Use exit 2 and action indicator <code>bbbbbbbb</code> . Do not change the data parameter. The data parameter contains the four-byte return code: 0 - processing complete 4 - warning: problem, but processing complete 8 - error: processing not complete

Open and close log file exits

To take these actions:	Do this:
Allow the function	<p>Use exit 12 and action indicator <code>bbbbbbbb</code>. Do to open the log not change the data parameter.</p> <p>Warning: Exits 13, 14, and 15 must be active and must return <code>bbbbbbbb</code>, or the function abends.</p>
Open the log with your own program	<p>Use exit 12 and action indicator <code>SKIPbbbb</code>.</p> <p>The data parameter is a fullword integer containing the log block size you coded in the UCL (or 0 if not specified). If the block size is 0 or incorrect, you must code the correct size in the data parameter. You must also supply the log blocks (exit 13) in the analysis and image application phases, reset the log (exit 14) in the image application phase, and close the log (exit 15).</p>
Abort during open	<p>Use exit 12 and action indicator <code>ABORTbbb</code>.</p>
Allow the function to close the log	<p>Use exit 15 and action indicator <code>bbbbbbbb</code>. Do not change the data parameter. Take this action only if the function opened the log. That is, exit 12 is inactive, or exit 12 is active but did not open the log file (return action indicator was <code>bbbbbbbb</code>).</p> <p>Warning: If exit 12 is active, and the return action indicator was <code>SKIPbbb</code>, the function abends.</p>

Analysis phase exits

To take these actions:	Do this:
Close the log with your own program	Use exit 15 and action indicator SKIP bbbb . Do not change the data parameter. You must take this action if you opened the log with your own program. That is, exit 12 is active and the return action indicator was SKIP bbbb . Warning: If exit 12 did not open the log, the function abends.
Abort during close	Use exit 15 and action indicator ABORT bbbb .
Use the function to read the log blocks	Use exit 13 and action indicator bbbbbbbbbb . Do not change the data parameter. Take this action only if the function opened the log. That is, exit 12 was inactive or was active but did not open the log (return action indicator was bbbbbbbbbb). Warning: If exit 12 is active and the return action indicator was SKIP bbbb , the function abends.
Read the log blocks with your own program	Use exit 13 and action indicator SKIP bbbb . Take this action only if your program opened the log file. That is, exit 12 is active, and the return action indicator was SKIP bbbb . Warning: If exit 12 did not open the log, the function abends. The data parameter is the previous log block. Supply a log block that matches the block size specified in exit 12.
Indicate there are no more log blocks to be passed	Use exit 13 and action indicator EOF bbbbbb (force end-of-file). Do not change the data parameter. Exit 12 must be active, and the return action indicator must have been SKIP bbbb . Warning: If exit 12 did not open the log, the function abends.
Abort while the log blocks are being read	Use exit 13 and action indicator ABORT bbbb .
Monitor reading of log block	Use exit 3 and action indicator bbbbbbbbbb . Do not change the data parameter (log block).
Indicate a logical end-of-file condition to stop reading the log	Use exit 3 and action indicator EOF bbbbbb (force end-of-file). Do not change the data parameter (log block). The log block just read will not be included.
Print log block	Use exit 4 and action indicator bbbbbbbbbb . Do not change the data parameter (block).
Skip printing of log block	Use exit 4 and action indicator SKIP bbbb . Do not change the data parameter (block).

To take these actions:	Do this:
Monitor reading of log records	Use exit 5 and action indicator <code>BBBBBBBB</code> . Do not change the data parameter (record).
Monitor printing of log records	Use exit 6 and action indicator <code>BBBBBBBB</code> . Do not change the data parameter (record).
Skip printing of log records	Use exit 6 and action indicator <code>SKIPBBBB</code> . Do not change the data parameter (record).

Application phase exits

To take these actions:	Do this:
Use the function to read the log block	<p data-bbox="485 599 1206 716">Use exit 13 and action indicator <code>BBBBBBBB</code>. Use this exit only if the function opened the log. That is, exit 12 was inactive or was active but did not open the log (return action indicator was <code>BBBBBBBB</code>).</p> <p data-bbox="485 732 1206 786">Warning: If exit 12 is active and the return action indicator was <code>SKIPBBBB</code>, the function abends.</p>
Read the log block with your own program	<p data-bbox="485 802 1206 919">Use exit 13 and action indicator <code>SKIPBBBB</code>. The data parameter is the previous log block. Take this action only if your program opened the log. That is, exit 12 is active, and the return action indicator was <code>SKIPBBBB</code>.</p> <p data-bbox="485 935 1206 1127">Warning: If exit 12 did not open the log, the function abends. Supply a log block that matches the block size in exit 12. Supply the log blocks in reverse order, starting with the log block coded in the data parameter of exit 14. Note that the log block might not be the last block passed in the analysis phase.</p>
Abort while the log blocks are being read	Use exit 13 and action indicator <code>ABORTBBBB</code> .
Monitor reading of log blocks	Use exit 7 and action indicator <code>BBBBBBBB</code> . Do not change the data parameter (block).
Monitor reading of log records	Use exit 8 and action indicator <code>BBBBBBBB</code> . Do not change the data parameter (log record).
Leave buffer information as is	Use exit 11 and action indicator <code>BBBBBBBB</code>

To take these actions:	Do this:
Change buffer information	<p>Use exit 11 and SETbbbb to indicate that the data parameter has been changed. The data parameter is two fullwords containing numbers: the first number specifies the method (see below), and the second number specifies the number of buffers or the amount of memory required. The default is option 2: 4 buffers per pool.</p> <p><u>Method Options:</u></p> <p>1 - Buffers per file (code the number) 2 - Buffers for the whole buffer pool (code the number) 3 - Amount of memory (code the actual number of bytes)</p>
Apply before-image record to the database	<p>Use exit 9 and action indicator bbbbbbbb. Do not change the data parameter (before-image).</p>
Skip applying the before-image record to the data	<p>Use exit 9 and action indicator SKIPbbbb. Do not change the data parameter base(before-image).</p> <p>Caution: Failure to apply all images could corrupt the database.</p>
Print applied image	<p>Use exit 10 and action indicator bbbbbbbb Do not change the data parameter (applied image).</p>
Skip printing of applied image	<p>Use exit 10 and action indicator SKIPbbbb. Do not change the data parameter.</p>
Use the function to reset the log	<p>Use exit 14 and action indicator bbbbbbbb . Take this action only if the function opened the log. That is, exit 12 is inactive.</p> <p>Warning: If exit 12 is active and the return action indicator was SKIPbbbb, the function abends. Do not change the data parameter.</p>
Reset log with your own program	<p>Use exit 14 and action indicator SKIPbbbb. Take this action only if your program opened the log. That is, exit 12 is active, and the return action indicator was SKIPbbbb.</p> <p>Warning: If exit 12 did not open the log, the function abends. Do not change the data parameter. The data parameter is a fullword containing the relative block number (RBN) that the function reads first in the image application phase. RBN 0 is the first block of the first logical volume. The function increments or decrements the RBN value by one for each block it reads from that point. Use this number to determine which block to supply in exit 13.</p>
Abort during reset	<p>Use exit 14 and action indicator ABORTbbbb.</p>

Valid actions

Once you are familiar with the information in the preceding section, use the following table for quick reference.

Exit	Use	Data parameter	Action indicators				
			#####	EOF#####	SKIP#####	SET#####	ABORT#####
1	Set exit settings	15-byte 'Y' and 'N' string (may be changed)	Y	N	N	Y	N
2	Get return code	0 (complete) 4 (warning) 8 (error)	Y	N	N	N	N
3	Monitor block read (analysis)	Log file block	Y	Y	N	N	N
4	Print Block (analysis)	Log file block	Y	N	Y	N	N
5	Monitor record read (analysis)	Log record	Y	N	N	N	N
6	Print record (analysis)	Log record	Y	N	Y	N	N
7	Monitor block read (apply)	Log file block	Y	N	N	N	N
8	Monitor record read (apply)	Log record	Y	N	N	N	N
9	Apply before image (apply)	Log record	Y	N	Y	N	N
10	Print applied image (apply)	Applied image	Y	N	Y	N	N

Exit	Use	Data parameter	Action indicators				
			bbbbbbbb	EOFbbbbbb	SKIPbbbbbb	SETbbbbbb	ABORTbbbb
11	Apply buffer (apply)	Buffer technique (may be changed)	Y	N	N	Y	N
12	Open log file	fullword integer specifying block size (may be changed)	Y	N	Y	N	Y
13	Supply log block (analysis, apply)	Previous log block (may be changed)	Y ¹	Y ²	Y ³	N	Y
14	Reset log file (apply)	Fullword integer containing the relative block number	Y ¹	N	Y ³	N	Y
15	Close log file	Blanks	Y ¹	N	Y ³	N	Y

- bbbbbbbb Action continues
- EOFbbbbbb Force end-of-file
- SKIPbbbbbb Action is not taken or your own exit performs it.
- SETbbbbbb Data comments changed.
- ABORTbbbb Terminates recovery

¹ Exit 12 must either be inactive or have bbbbbbbb as the return action indicator.

² Valid only for analysis. Also see footnote 3.

³ Exit 12 must be active and have SKIPbbbb as the return action indicator.

Examples

This section shows output from sample runs of the Recover, Restore, and Log-Print functions.

Recover example

```
CSUL0101I : COMMENCING COMMAND VALIDATION.
 1 CONTROL(BEGIN)
 2 LIST(APPLIED-IMAGES)
 3 LINES(2)
 4 DATA-FORMAT(HEX CHAR)
 5 LOG-FILE(LOGFILE)
 6 FUNCTION(RECOVER)
 7 STATE(LOG-BEGIN)
 8 OPEN-FILE(DYNAMIC)
 9 FILE(ALL)
10 CONTROL(END)
CONTENTS OF SOURCE LINES OUTSIDE 1...72 MARGINS IGNORED.
 0 SYNTAX ERRORS DETECTED.
10 COMMAND LINES READ.
 1 CONTROL SECTIONS ANALYZED.
 1 FUNCTION COMMANDS ANALYZED.
CSUL0102I : COMMENCING COMMAND EXECUTION.
CSUL0310I : COMMENCING CONTROL SECTION.
CSUL0302I : COMMENCING RECOVER PROCESS.
                FUNCTION = RECOVER
CSUL2100I : START OF SYSTEM LOG FILE ANALYSIS.
CSUL2110I : END OF FILE DUE TO PDM TERMINATION RECORD :
                BLOCK =          29, RECORD =          127.
                FUNCTION = RECOVER
CSUL2161I : SYSTEM LOG FILE ANALYSIS STATISTICS (BASE).
```

LOG FILE DDNAME		LOGFILE
NUMBER OF LOG VOLUMES		1
LOG DEVICE TYPE		DISK
NUMBER OF LOG BLOCKS		29
NUMBER OF SPANNING BLOCKS		12
SPANNING BLOCKS AS A PERCENTAGE OF TOTAL BLOCKS	41.38%	
NUMBER OF LOG RECORDS		127
AVERAGE NUMBER OF LOG RECORDS PER BLOCK	4.38AVG.	
NUMBER OF SPANNING RECORDS		2
SPANNING RECORDS AS A PERCENTAGE OF TOTAL RECORDS	1.57%	
NUMBER OF RECORD BYTES		64,166
NUMBER OF UNUSED BYTES		23,530
UNUSED BYTES AS A PERCENTAGE OF TOTAL BYTES	26.83%	
NUMBER OF COMMAND RECORDS		8
COMMAND RECORDS AS A PERCENTAGE OF TOTAL RECORDS	6.30%	
NUMBER OF BEFORE IMAGES		5
BEFORE IMAGES AS A PERCENTAGE OF TOTAL RECORDS	3.94%	
NUMBER OF AFTER IMAGES		12
AFTER IMAGES AS A PERCENTAGE OF TOTAL RECORDS	9.45%	
NUMBER OF CONTROL RECORDS		102
CONTROL RECORDS AS A PERCENTAGE OF TOTAL RECORDS	80.31%	
TOTAL NUMBER OF TASKS ON THE LOG		6
NUMBER OF UPDATE TASKS ON THE LOG		1
NUMBER OF TASKS SIGNED ON AT THE END OF THE LOG		0
NUMBER OF FILES ON THE LOG		28
CSUL2163I : END OF SYSTEM LOG FILE ANALYSIS STATISTICS (BASE).		
CSUL2101I : END OF SYSTEM LOG FILE ANALYSIS.		

The output from the Recover function follows:

```

FUNCTION = RECOVER

CSUL2108I : START OF IMAGE APPLICATION PHASE.

CSUL2116I : BEFORE IMAGE RECORD:                BLOCK =          26,  RECORD =          97.
           EB24JADV EB24JADV EB24SADV                IMAG µb±_Dí<      E$MB
0008 CCFDCCCE CCFDCCCE CCFECCCE FFFFFFFFFFFFFFFFFF CDCC B872CC40 0000 C5DC FFFFFFFF
0006 52241145 52241145 52242145 FFFFFFFFFFFFFFFFFF 9417 02484FC0 0000 5B42 FFFFFFFF
BEFR                                ADDVC
CCCD 0000 0004 0000 0001 CCCEC FFFFFFFF
2569 0000 004C 0000 0005 14453 FFFFFFFF

CSUL2116I : BEFORE IMAGE RECORD:                BLOCK =          26,  RECORD =          93.
           EB24JADV EB24JADV EB24SADV                IMAG µb±_DÄ_      E$MB
0008 CCFDCCCE CCFDCCCE CCFECCCE FFFFFFFFFFFFFFFFFF CDCC B872CB60 0000 C5DC FFFFFFFF
0006 52241145 52241145 52242145 FFFFFFFFFFFFFFFFFF 9417 02484FD0 0000 5B42 FFFFFFFF
BEFR                                ADDVC
CCCD 0000 0004 0000 0001 CCCEC FFFFFFFF
2569 0000 004B 0000 0005 14453 FFFFFFFF

CSUL2116I : BEFORE IMAGE RECORD:                BLOCK =          26,  RECORD =          89.
           EB24JADV EB24JADV EB24SADV                IMAG µb±_D_—      E$MB
0008 CCFDCCCE CCFDCCCE CCFECCCE FFFFFFFFFFFFFFFFFF CDCC B872C390 0000 C5DC FFFFFFFF
0006 52241145 52241145 52242145 FFFFFFFFFFFFFFFFFF 9417 02484BA0 0000 5B42 FFFFFFFF
BEFR                                ADDVC
CCCD 0000 0004 0000 0001 CCCEC FFFFFFFF
2569 0000 004A 0000 0005 14453 FFFFFFFF

CSUL2116I : BEFORE IMAGE RECORD:                BLOCK =          26,  RECORD =          85.
           EB24JADV EB24JADV EB24SADV                IMAG µb±_D_«      E$MB
0008 CCFDCCCE CCFDCCCE CCFECCCE FFFFFFFFFFFFFFFFFF CDCC B872C240 0000 C5DC FFFFFFFF
0006 52241145 52241145 52242145 FFFFFFFFFFFFFFFFFF 9417 02484C80 0000 5B42 FFFFFFFF
BEFR                                ADDVC
CCCD 0000 0009 0000 0001 CCCEC FFFFFFFF
2569 0000 0048 0000 0005 14453 FFFFFFFF

CSUL2116I : BEFORE IMAGE RECORD:                BLOCK =          26,  RECORD =          82.
           EB24JADV EB24JADV EB24SADV                IMAG µb±_Cò»      E$BR
000D CCFDCCCE CCFDCCCE CCFECCCE FFFFFFFFFFFFFFFFFF CDCC B872CAB0 0000 C5CD FFFFFFFF
0004 52241145 52241145 52242145 FFFFFFFFFFFFFFFFFF 9417 02483E70 0000 5B29 FFFFFFFF
BEFR                                ADD-M
CCCD 0000 000E 0000 0006 CCC6D FFFFFFFF
2569 0000 000F 0008 0003 14404 FFFFFFFF

```

CSUL2164I : SYSTEM LOG FILE APPLICATION STATISTICS (BASE).

FUNCTION TYPE	RECOVER
TOTAL APPLIED IMAGES	5
SKIPPED IMAGES (FILE-ERROR)	0
SKIPPED IMAGES (USER-EXIT)	0
APPLIED IMAGES AS A PERCENTAGE OF TOTAL RECORDS	3.94%
NUMBER OF FILES TO WHICH IMAGES WERE APPLIED	2
NUMBER OF FILES WITHOUT PROCESSING ERRORS	24
NUMBER OF FILES WITH PROCESSING ERRORS	0

CSUL2166I : END OF SYSTEM LOG FILE APPLICATION STATISTICS (BASE).

CSUL2109I : END OF IMAGE APPLICATION PHASE.

CSUL2114I : OPERATION COMPLETED SUCCESSFULLY . 0 WARNINGS AND 0 ERRORS WERE ISSUED.

CSUL0303I : RECOVER PROCESS TERMINATING.

CSUL0305I : CONTROL SECTION TERMINATING.

CSUL0307I : ALL CONTROL SECTIONS PROCESSED.

CSUL0103I : DATA BASE UTILITIES SUCCESSFUL TERMINATION.

Restore example

```
CSUL0101I : COMMENCING COMMAND VALIDATION.
  1      CONTROL(BEGIN)
  2      LIST(APPLIED-IMAGES)
  3      LINES(2)
  4      DATA-FORMAT(HEX CHAR)
  5      LOG-FILE(LOGFILE)
  6      FUNCTION(RESTORE)
  7      STATE(LOG-END)
  8      FILE(ALL)
  9      CONTROL(END)
CONTENTS OF SOURCE LINES OUTSIDE 1...72 MARGINS IGNORED.
  0      SYNTAX ERRORS DETECTED.
  9      COMMAND LINES READ.
  1      CONTROL SECTIONS ANALYZED.
  1      FUNCTION COMMANDS ANALYZED.

CSUL0102I : COMMENCING COMMAND EXECUTION.
CSUL0310I : COMMENCING CONTROL SECTION.
CSUL0302I : COMMENCING RESTORE          PROCESS.
           FUNCTION = RESTORE

CSUL2100I : START OF SYSTEM LOG FILE ANALYSIS.

CSUL2110I : END OF FILE DUE TO PDM TERMINATION RECORD :
           BLOCK =          29,  RECORD =          127.
```

```

CSUL2161I : SYSTEM LOG FILE ANALYSIS STATISTICS (BASE).
LOG FILE DDNAME                                LOGFILE
NUMBER OF LOG VOLUMES                          1
LOG DEVICE TYPE                                DISK
NUMBER OF LOG BLOCKS                           29
NUMBER OF SPANNING BLOCKS                      12
SPANNING BLOCKS AS A PERCENTAGE OF TOTAL BLOCKS 41.38%
NUMBER OF LOG RECORDS                          127
AVERAGE NUMBER OF LOG RECORDS PER BLOCK      4.38AVG.
NUMBER OF SPANNING RECORDS                     2
SPANNING RECORDS AS A PERCENTAGE OF TOTAL RECORDS 1.57%
NUMBER OF RECORD BYTES                         64,166
NUMBER OF UNUSED BYTES                         23,530
UNUSED BYTES AS A PERCENTAGE OF TOTAL BYTES   26.83%
NUMBER OF COMMAND RECORDS                      8
COMMAND RECORDS AS A PERCENTAGE OF TOTAL RECORDS 6.30%
NUMBER OF BEFORE IMAGES                       5
BEFORE IMAGES AS A PERCENTAGE OF TOTAL RECORDS 3.94%
NUMBER OF AFTER IMAGES                        12
AFTER IMAGES AS A PERCENTAGE OF TOTAL RECORDS 9.45%
NUMBER OF CONTROL RECORDS                     102
CONTROL RECORDS AS A PERCENTAGE OF TOTAL RECORDS 80.31%
TOTAL NUMBER OF TASKS ON THE LOG              6
NUMBER OF UPDATE TASKS ON THE LOG             1
NUMBER OF TASKS SIGNED ON AT THE END OF THE LOG 0
NUMBER OF FILES ON THE LOG                    28
CSUL2163I : END OF SYSTEM LOG FILE ANALYSIS STATISTICS (BASE).
CSUL2101I : END OF SYSTEM LOG FILE ANALYSIS.

```

CSUL2108I : START OF IMAGE APPLICATION PHASE.

```

CSUL2116I : AFTER IMAGE RECORD:                BLOCK =          26,  RECORD =          83.
      EB24JADV EB24JADV EB24SADV                IMAG µb±_C¶_    E$BR
000D CCFDCE CCFDCE CCFECCE FFFFFFFFFFFFFFFFFF CDCC B872CB20 0000 C5CD FFFFFFFF
0004 52241145 52241145 52242145 FFFFFFFFFFFFFFFFFF 9417 02483130 0000 5B29 FFFFFFFF
AFTR                                ADD-M
CCED 0000 000E 0000 0006 CCC6D FFFFFFFF
1639 0000 000F 0008 0003 14404 FFFFFFFF
      HUGO D. SMYTHE      11111 ANYSTREET      ANYCITY      IN123450010212345678912345
44444444000044444444CECD4C44EDECCE44444444FFFFFFFF4CDEEEDCCE44444444CECCEE44444444CDFFFFFFFFFFFFFFFFFFFFFFFF
0000000000000000000000847604B024838500000011111015823955300000158393800000095123450010212345678912345
CSUL2116I : AFTER IMAGE RECORD:                BLOCK =          26,  RECORD =          84.
      EB24JADV EB24JADV EB24SADV                IMAG µb±_D_-    E$BR
000D CCFDCE CCFDCE CCFECCE FFFFFFFFFFFFFFFFFF CDCC B872C2D0 0000 C5CD FFFFFFFF
0004 52241145 52241145 52242145 FFFFFFFFFFFFFFFFFF 9417 02484AB0 0000 5B29 FFFFFFFF
AFTR                                ADDVC
CCED 0000 000E 0000 0006 CCCEC FFFFFFFF
1639 0000 000F 0008 0003 14453 FFFFFFFF
      _q _qHUGO D. SMYTHE      11111 ANYSTREET      ANYCITY      IN123450010212345678912345
4444444400000090009CECD4C44EDECCE44444444FFFFFFFF4CDEEEDCCE44444444CECCEE44444444CDFFFFFFFFFFFFFFFFFFFFFFFF
00000000000000480048847604B024838500000011111015823955300000158393800000095123450010212345678912345
CSUL2116I : AFTER IMAGE RECORD:                BLOCK =          26,  RECORD =          86.
      EB24JADV EB24JADV EB24SADV                IMAG µb±_D_V    E$MB
0008 CCFDCE CCFDCE CCFECCE FFFFFFFFFFFFFFFFFF CDCC B872C2E0 0000 C5DC FFFFFFFF
0006 52241145 52241145 52242145 FFFFFFFFFFFFFFFFFF 9417 02484C50 0000 5B42 FFFFFFFF
AFTR                                ADDVC
CCED 0000 0009 0000 0001 CCCEC FFFFFFFF
1639 0000 000F 0008 0005 14453 FFFFFFFF
      31441AAAA
00004444444444444444FFFCCEC
0000000000000314411111
CSUL2116I : AFTER IMAGE RECORD:                BLOCK =          26,  RECORD =          87.
      EB24JADV EB24JADV EB24SADV                IMAG µb±_D_ñ    E$BR
000D CCFDCE CCFDCE CCFECCE FFFFFFFFFFFFFFFFFF CDCC B872C390 0000 C5CD FFFFFFFF
0004 52241145 52241145 52242145 FFFFFFFFFFFFFFFFFF 9417 024849E0 0000 5B29 FFFFFFFF
AFTR                                ADDVC
CCED 0000 000E 0000 0006 CCCEC FFFFFFFF
1639 0000 000F 0008 0003 14453 FFFFFFFF
      _q _>HUGO D. SMYTHE      11111 ANYSTREET      ANYCITY      IN123450010212345678912345
4444444400000090004CECD4C44EDECCE44444444FFFFFFFF4CDEEEDCCE44444444CECCEE44444444CDFFFFFFFFFFFFFFFFFFFFFFFF
0000000000000048004A847604B024838500000011111015823955300000158393800000095123450010212345678912345

```

```

CSUL2116I : AFTER IMAGE RECORD:                BLOCK =          26,  RECORD =          88.
      EB24JADV EB24JADV EB24SADV                IMAG µb±_D_k      E$MB
0008 CCFDCE CCFDCE CCFECE FFFFFFFFFFFFFFFFFF CDCC B872C390 0000 C5DC FFFFFFFF
0006 52241145 52241145 52242145 FFFFFFFFFFFFFFFFFF 9417 02484A20 0000 5B42 FFFFFFFF
AFTR                ADDVC
CCED 0000 0009 FFFF 0001 CCCEC FFFFFFFF
1639 0000 0048 FFFF 0005 14453 FFFFFFFF
      _>31441AAAA
000044440004FFFFFFCCC
00000000004A314411111
CSUL2116I : AFTER IMAGE RECORD:                BLOCK =          26,  RECORD =          90.
      EB24JADV EB24JADV EB24SADV                IMAG µb±_D__     E$MB
0008 CCFDCE CCFDCE CCFECE FFFFFFFFFFFFFFFFFF CDCC B872C310 0000 C5DC FFFFFFFF
0006 52241145 52241145 52242145 FFFFFFFFFFFFFFFFFF 9417 02484C60 0000 5B42 FFFFFFFF
AFTR                ADDVC
CCED 0000 0004 0000 0001 CCCEC FFFFFFFF
1639 0000 004A 0000 0005 14453 FFFFFFFF
      _q      31441BBBB
000000094444FFFFFFCCC
000000480000314412222
CSUL2116I : AFTER IMAGE RECORD:                BLOCK =          26,  RECORD =          91.
      EB24JADV EB24JADV EB24SADV                IMAG µb±_D]ë     E$BR
000D CCFDCE CCFDCE CCFECE FFFFFFFFFFFFFFFFFF CDCC B872CB40 0000 C5DC FFFFFFFF
0004 52241145 52241145 52242145 FFFFFFFFFFFFFFFFFF 9417 02484D40 0000 5B29 FFFFFFFF
AFTR                ADDVC
CCED 0000 000E 0000 0006 CCCEC FFFFFFFF
1639 0000 000F 0008 0003 14453 FFFFFFFF
      _q  _ .HUGO D. SMYTHE      11111 ANYSTREET      ANYCITY      IN123450010212345678912345
4444444400000090004CECD4C44EDEECC444444FFFFF4CDEEDECCE444444CDEECBE444444CDFFFFFFFFFFFFFFFFFFFF
000000000000048004B847604B0248385000001111101582395530000015839380000095123450010212345678912345
CSUL2116I : AFTER IMAGE RECORD:                BLOCK =          26,  RECORD =          92.
      EB24JADV EB24JADV EB24SADV                IMAG µb±_D0_     E$MB
0008 CCFDCE CCFDCE CCFECE FFFFFFFFFFFFFFFFFF CDCC B872CB30 0000 C5DC FFFFFFFF
0006 52241145 52241145 52242145 FFFFFFFFFFFFFFFFFF 9417 02484E40 0000 5B42 FFFFFFFF
AFTR                ADDVC
CCED 0000 0004 FFFF 0001 CCCEC FFFFFFFF
1639 0000 004A FFFF 0005 14453 FFFFFFFF
      _q  _ .31441BBBB
000000090004FFFFFFCCC
000000480004B314412222

```


CSUL2164I : SYSTEM LOG FILE APPLICATION STATISTICS (BASE).

FUNCTION TYPE	RESTORE
TOTAL APPLIED IMAGES	12
SKIPPED IMAGES (FILE-ERROR)	0
SKIPPED IMAGES (USER-EXIT)	0
APPLIED IMAGES AS A PERCENTAGE OF TOTAL RECORDS	9.45%
NUMBER OF FILES TO WHICH IMAGES WERE APPLIED	2
NUMBER OF FILES WITHOUT PROCESSING ERRORS	24
NUMBER OF FILES WITH PROCESSING ERRORS	0

CSUL2166I : END OF SYSTEM LOG FILE APPLICATION STATISTICS (BASE).

CSUL2109I : END OF IMAGE APPLICATION PHASE.

CSUL2114I : OPERATION COMPLETED SUCCESSFULLY . 0 WARNINGS AND 0 ERRORS WERE ISSUED.

CSUL0303I : RESTORE PROCESS TERMINATING.

CSUL0305I : CONTROL SECTION TERMINATING.

CSUL0307I : ALL CONTROL SECTIONS PROCESSED.

CSUL0103I : DATA BASE UTILITIES SUCCESSFUL TERMINATION.

Log-Print example

The following example shows input and output for the Log-Print function. The function prints the records and before-images before the System Log Analysis Statistics.

```

CSUL0101I : COMMENCING COMMAND VALIDATION.
  1 CONTROL(BEGIN)
  2 LIST(BEFORE AFTER)
  3 LINES(2)
  4 DATA-FORMAT(HEX CHAR)
  5 DIAGNOSTICS(EXTENDED)
  6 LOG-FILE( )
  7 DEVICE(TAPE)
  8 SEQ-ERROR(ERROR)
  9 FUNCTION(LOG-PRINT)
 10 STATISTICS(ALL)
 11 FILE(ALL)
 12 CONTROL(END)
CONTENTS OF SOURCE LINES OUTSIDE 1...72 MARGINS IGNORED.
  0 SYNTAX ERRORS DETECTED.
 12 COMMAND LINES READ.
  1 CONTROL SECTIONS ANALYZED.
  1 FUNCTION COMMANDS ANALYZED.

CSUL0102I : COMMENCING COMMAND EXECUTION.
CSUL0310I : COMMENCING CONTROL SECTION.
CSUL0302I : COMMENCING LOG-PRINT PROCESS.

CSUL2100I : START OF SYSTEM LOG FILE ANALYSIS.
CSUL2116I : BEFORE IMAGE RECORD:                BLOCK =          26, RECORD =          82.
          EB24JADV EB24JADV EB24SADV                IMAG µc"~_ó-      E$BR
BEFR                                ADD-M
CSUL2116I : AFTER IMAGE RECORD:                 BLOCK =          26, RECORD =          83.
          EB24JADV EB24JADV EB24SADV                IMAG µc"~_Å¥      E$BR
AFTR                                ADD-M
          HUGO D. SMYTHE 11111 ANYSTREET ANYCITY IN123450010212345678912345
CSUL2116I : AFTER IMAGE RECORD:                 BLOCK =          26, RECORD =          84.
          EB24JADV EB24JADV EB24SADV                IMAG µc"~__      E$BR
AFTR                                ADDVC
          _q _qHUGO D. SMYTHE 11111 ANYSTREET ANYCITY IN123450010212345678912345
CSUL2116I : BEFORE IMAGE RECORD:                 BLOCK =          26, RECORD =          85.
          EB24JADV EB24JADV EB24SADV                IMAG µc"~__î      E$MB
BEFR                                ADDVC
CSUL2116I : AFTER IMAGE RECORD:                 BLOCK =          26, RECORD =          86.
          EB24JADV EB24JADV EB24SADV                IMAG µc"~__°      E$MB
AFTR                                ADDVC

```

```

CSUL2116I : AFTER IMAGE RECORD:          BLOCK =          26, RECORD =          87.
EB24JADV EB24JADV EB24SADV                IMAG µc"~___ E$BR
AFTR ADDVC
   _q _>HUGO D. SMYTHE          11111 ANYSTREET ANYCITY IN123450010212345678912345
CSUL2116I : AFTER IMAGE RECORD:          BLOCK =          26, RECORD =          88.
EB24JADV EB24JADV EB24SADV                IMAG µc"~___ E$MB
AFTR ADDVC
   _>31441AAAA
CSUL2116I : BEFORE IMAGE RECORD:         BLOCK =          26, RECORD =          89.
EB24JADV EB24JADV EB24SADV                IMAG µc"~___ E$MB
BEFR ADDVC
CSUL2116I : AFTER IMAGE RECORD:          BLOCK =          26, RECORD =          90.
EB24JADV EB24JADV EB24SADV                IMAG µc"~___] E$MB
AFTR ADDVC
   _q 31441BBBB
CSUL2116I : AFTER IMAGE RECORD:          BLOCK =          26, RECORD =          91.
EB24JADV EB24JADV EB24SADV                IMAG µc"~__>E E$BR
AFTR ADDVC
   _q _ .HUGO D. SMYTHE          11111 ANYSTREET ANYCITY IN123450010212345678912345
CSUL2116I : AFTER IMAGE RECORD:          BLOCK =          26, RECORD =          92.
EB24JADV EB24JADV EB24SADV                IMAG µc"~_ .~ E$MB
AFTR ADDVC
   _q _ .31441BBBB
CSUL2116I : BEFORE IMAGE RECORD:         BLOCK =          26, RECORD =          93.
EB24JADV EB24JADV EB24SADV                IMAG µc"~__<L E$MB
BEFR ADDVC
CSUL2116I : AFTER IMAGE RECORD:          BLOCK =          26, RECORD =          94.
EB24JADV EB24JADV EB24SADV                IMAG µc"~__( * E$MB
AFTR ADDVC
   _> 31441CCCC
CSUL2116I : AFTER IMAGE RECORD:          BLOCK =          26, RECORD =          95.
EB24JADV EB24JADV EB24SADV                IMAG µc"~__"\ E$BR
AFTR ADDVC
   _q _<HUGO D. SMYTHE          11111 ANYSTREET ANYCITY IN123450010212345678912345
CSUL2116I : AFTER IMAGE RECORD:          BLOCK =          26, RECORD =          96.
EB24JADV EB24JADV EB24SADV                IMAG µc"~_ x E$MB
AFTR ADDVC
   _> _<31441CCCC
CSUL2116I : BEFORE IMAGE RECORD:         BLOCK =          26, RECORD =          97.
EB24JADV EB24JADV EB24SADV                IMAG µc"~_!^ E$MB
BEFR ADDVC
CSUL2116I : AFTER IMAGE RECORD:          BLOCK =          26, RECORD =          98.
EB24JADV EB24JADV EB24SADV                IMAG µc"~_$_ E$MB
AFTR ADDVC
   _ . 31441DDDD
CSUL2110I : END OF FILE DUE TO PDM TERMINATION RECORD :
BLOCK =          29, RECORD =          127.

```

```
CSUL2161I : SYSTEM LOG FILE ANALYSIS STATISTICS (BASE).  
LOG FILE DDNAME LOGFILE  
NUMBER OF LOG VOLUMES 1  
LOG DEVICE TYPE TAPE  
NUMBER OF LOG BLOCKS 29  
NUMBER OF SPANNING BLOCKS 12  
SPANNING BLOCKS AS A PERCENTAGE OF TOTAL BLOCKS 41.38%  
NUMBER OF LOG RECORDS 127  
AVERAGE NUMBER OF LOG RECORDS PER BLOCK 4.38AVG.  
NUMBER OF SPANNING RECORDS 2  
SPANNING RECORDS AS A PERCENTAGE OF TOTAL RECORDS 1.57%  
NUMBER OF RECORD BYTES 64,166  
NUMBER OF UNUSED BYTES 23,530  
UNUSED BYTES AS A PERCENTAGE OF TOTAL BYTES 26.83%  
NUMBER OF COMMAND RECORDS 8  
COMMAND RECORDS AS A PERCENTAGE OF TOTAL RECORDS 6.30%  
NUMBER OF BEFORE IMAGES 5  
BEFORE IMAGES AS A PERCENTAGE OF TOTAL RECORDS 3.94%  
NUMBER OF AFTER IMAGES 12  
AFTER IMAGES AS A PERCENTAGE OF TOTAL RECORDS 9.45%  
NUMBER OF CONTROL RECORDS 102  
CONTROL RECORDS AS A PERCENTAGE OF TOTAL RECORDS 80.31%  
TOTAL NUMBER OF TASKS ON THE LOG 6  
NUMBER OF UPDATE TASKS ON THE LOG 1  
NUMBER OF TASKS SIGNED ON AT THE END OF THE LOG 0  
NUMBER OF FILES ON THE LOG 28  
CSUL2163I : END OF SYSTEM LOG FILE ANALYSIS STATISTICS (BASE).
```


19

Coding the Review function

Coding the Review function

You use the Review function to determine whether database files are locked. The Review function prints a message indicating whether files are locked, but does not unlock the files.

When a file is unlocked after an abend, the file was not open for update at the time of the abend, and the abend should not have affected its contents.

When a file is locked after an abend, the file was open for update at the time of the abend. The file may have been updated since the last commit, or the abend may have interrupted an update in progress, damaging the contents of the file. You should recover the file using Task Level Recovery by warm starting the PDM, using the **Recover** function, or using the **Restore** function.

Coding the UCL for the Review function

After you code the control section as shown in “[Coding the control section](#)” on page 57, you can code the Review function as shown in the following format. For UCL examples, see “[Review example](#)” on page 487.

FUNCTION (REVIEW)

$$\text{FILE} \left(\begin{array}{l} \text{ALL} \\ \text{file - name - list} \end{array} \right) \dots$$

FUNCTION (REVIEW)

Description *Required.* Invokes the Review function.

$$\text{FILE} \left(\begin{array}{l} \text{ALL} \\ \text{file-name-list} \end{array} \right) \dots$$

Description *Required.* Names the files you want to review.

Format File names must be 4 alphanumeric characters. The first character must be alphabetic. Separate names with commas.

Options ALL Reviews all index, primary, and related files in the schema.

file-name-list Reviews the files you name.

Considerations

- ◆ If you code FILE (ALL), Review processes your user database files in alphabetical order, processing index files first, primary files second, and related files last. The Review function does not process Directory files.
- ◆ If you code FILE (*file-name-list*), Review processes the named files. You can name user database files and/or Directory files.
- ◆ If you code FILE (ALL), you cannot code any other FILE statements.
- ◆ If you code FILE (), the function does not process any files.
- ◆ If you specify primary or related files, the Review function does not automatically process the associated index files. You must explicitly name the index files you want to process.
- ◆ You can code the FILE statement one or more times.

Review example

Example 1. To print a message indicating whether the CUST, ACCR, PORD, and VEND files are locked, code the following:

```
CONTROL (BEGIN)
      ENV-DESC (MYDESC)
      SCHEMA (MYSHEMA)
FUNCTION (REVIEW)
      FILE (CUST)
      FILE (ACCR,PORD,VEND)
CONTROL (END)
```

Example 2. This example shows sample input and output.

```
CSUL0101I : COMMENCING COMMAND VALIDATION
1 CONTROL(BEGIN)
2 *****
3 * *
4 * REVIEW EXAMPLE #1 DESCRIPTION *
5 * *
6 * OBJECTIVE: DETERMINE WHETHER OR NOT THE *
7 * SPECIFIED FILES ARE LOCKED *
8 * *
9 * *
10 *****
11 ENV-DESC(UTEDOOS)
12 SCHEMA(UTILSCHM)
13 FUNCTION(REVIEW)
14 FILE(PANM,RANV,P001)
15 CONTROL(END)
CONTENTS OF SOURCE LINES OUTSIDE 1..72 MARGINS IGNORED.
0 SYNTAX ERRORS DETECTED.
15 COMMAND LINES READ.
1 CONTROL SECTIONS ANALYZED.
1 FUNCTION COMMANDS ANALYZED.
```

```
CSUL0102I : COMMENCING COMMAND EXECUTION
CSUL0301I : COMMENCING CONTROL SECTION USING ENVIRONMENT DESCRIPTION UTEDOOS AND SCHEMA UTILSCHM
CSUL0302I : COMMENCING REVIEW                PROCESS
CSUL0311I : COMMENCING REVIEW                AGAINST FILE PANM
CSUL1604I : FILE PANM IS LOCKED. PDMNAME = TESTPDM
CSUL0321I : REVIEW                          PROCESSING AGAINST FILE PANM TERMINATING NORMALLY
CSUL0311I : COMMENCING REVIEW                AGAINST FILE RANV
CSUL1604I : FILE RANV IS LOCKED. PDMNAME = TESTPDM
CSUL0321I : REVIEW                          PROCESSING AGAINST FILE RANV TERMINATING NORMALLY
CSUL0311I : COMMENCING REVIEW                AGAINST FILE P001
CSUL1603I : FILE P001 IS NOT LOCKED.
CSUL0321I : REVIEW                          PROCESSING AGAINST FILE P001 TERMINATING NORMALLY
CSUL0303I : REVIEW                          PROCESS TERMINATING.
CSUL0305I : CONTROL SECTION TERMINATING.
CSUL0307I : ALL CONTROL SECTIONS PROCESSED.
CSUL0103I : DATA BASE UTILITIES SUCCESSFUL TERMINATION.
```

20

Coding the Unlock function

Coding the Unlock function

The Unlock function examines each field in the lock record of the files you specify. If any fields are incorrect, the Unlock function prints an error message showing the incorrect field. If all fields are correct and indicate that the file is locked, the Unlock function updates the lock record to show that the file is unlocked.

When a file is locked after an abend, the file was open for update at the time of the abend. The file may have been updated since the last commit or the abend may have interrupted an update in progress, damaging the contents of the file. You should recover the file using Task Level Recovery by warm starting the PDM, using the [Recover](#) function, or using the [Restore](#) function.

If a file is locked after an abend and you believe the file had not been updated at the time of the abend, you may want to unlock the file with the Unlock function. If you are wrong, the damage to the database is unpredictable. Therefore, we strongly discourage the use of the Unlock function.

After you code the control section as shown in “[Coding the control section](#)” on page 57, you can code the Unlock function as shown in the following format. For UCL examples, see “[Coding the Unlock function](#)” on page 489.

FUNCTION (UNLOCK)

FILE ({ ALL
file-name-list }) ...

FUNCTION (UNLOCK)

Description *Required.* Invokes the Unlock function.

FILE ({ ALL
file-name-list }) ...

Description *Required.* Names the files you want to unlock.

Format File names must be 4 alphanumeric characters. The first character must be alphabetic. Separate names with commas.

Options ALL Unlocks all index, primary, and related files in the schema.

file-name-list Unlocks the files you name.

Considerations

- ◆ If you code FILE (ALL), Unlock processes your user database files in alphabetical order, processing index files first, primary files second, and related files last. The Unlock function does not process Directory files.
- ◆ If you code FILE (*file-name-list*), Unlock processes the named files. You can name user database files and/or Directory files.
- ◆ If you code FILE (ALL), you cannot code any other FILE statements.
- ◆ If you code FILE (), Unlock does not process any files.
- ◆ If you specify primary or related files, the Unlock function does not automatically process the associated index files. You must explicitly name the index files you want to process.
- ◆ You can code the FILE statement one or more times.

Example 1 To unlock the CUST, ACCR, PORD, and VEND files, code the following:

```
CONTROL (BEGIN)
      ENV-DESC (MYDESC)
      SCHEMA (MYSHEMA)
FUNCTION (UNLOCK)
      FILE (CUST)
      FILE (ACCR,PORD,VEND)
CONTROL (END)
```

Example 2 This example shows sample input and output.

```
CSUL0101I : COMMENCING COMMAND VALIDATION
1 CONTROL(BEGIN)
2 *****
3 *
4 * UNLOCK EXAMPLE #1 DESCRIPTION
5 *
6 * OBJECTIVE: TO RESET THE LOCK BYTE IN SPECIFIED
7 * DATA BASE FILES
8 *
9 * NOTES:
10 * *****
11 * *
12 * * 1. EXERCISE EXTREME CAUTION WHEN USING THE *
13 * * UNLOCK FUNCTION. DATABASE FILES MAY *
14 * * BE SEVERELY DAMAGED WHEN MISUSED. *
15 * *
16 * * 2. ALL FILES THAT ARE LOCKED AFTER A PDM *
17 * * OR SYSTEM ABEND SHOULD BE RECOVERED *
18 * * THROUGH THE USE OF THE RECOVER FUNCTION.*
19 * *
20 * *****
21 *
22 *****
23 ENV-DESC(UTEDOOUS)
24 SCHEMA(UTILSCHM)
25 FUNCTION(UNLOCK)
26 FILE(PANM,RANV,P001)
27 CONTROL(END)
```

CONTENTS OF SOURCE LINES OUTSIDE 1...72 MARGINS IGNORED.

0 SYNTAX ERRORS DETECTED.
27 COMMAND LINES READ.
1 CONTROL SECTIONS ANALYZED.
1 FUNCTION COMMANDS ANALYZED.

CSUL0102I : COMMENCING COMMAND EXECUTION
CSUL0301I : COMMENCING CONTROL SECTION USING ENVIRONMENT DESCRIPTION UTEDOOS AND SCHEMA UTILSCHM
CSUL0302I : COMMENCING UNLOCK PROCESS
CSUL0311I : COMMENCING UNLOCK AGAINST FILE PANM
CSUL1805I : UNLOCKING FILES WHICH ARE LOCKED DUE TO SYSTEM ABEND COULD RESULT IN STRUCTURAL DAMAGE
CSUL1804I : FILE PANM IS NOW UNLOCKED. PDMNAME WAS TESTPDM
CSUL0321I : UNLOCK PROCESSING AGAINST FILE PANM TERMINATING NORMALLY
CSUL0311I : COMMENCING UNLOCK AGAINST FILE RANV
CSUL1805I : UNLOCKING FILES WHICH ARE LOCKED DUE TO SYSTEM ABEND COULD RESULT IN STRUCTURAL DAMAGE
CSUL1804I : FILE RANV IS NOW UNLOCKED. PDMNAME WAS TESTPDM
CSUL0321I : UNLOCK PROCESSING AGAINST FILE RANV TERMINATING NORMALLY
CSUL0311I : COMMENCING UNLOCK AGAINST FILE P001
CSUL1805I : UNLOCKING FILES WHICH ARE LOCKED DUE TO SYSTEM ABEND COULD RESULT IN STRUCTURAL DAMAGE
CSUL1603I : FILE P001 IS NOT LOCKED.
CSUL0321I : UNLOCK PROCESSING AGAINST FILE P001 TERMINATING NORMALLY
CSUL0303I : UNLOCK PROCESS TERMINATING.
CSUL0305I : CONTROL SECTION TERMINATING.
CSUL0307I : ALL CONTROL SECTIONS PROCESSED.
CSUL0103I : DATA BASE UTILITIES SUCCESSFUL TERMINATION.

Index

*

*FILL parameter
 changing length of elements
 170, 179, 258
 equalizing length of elements
 156
 example 260
 exit programs 280, 286
 Unload function (Version 1) 165
 unloading files 354

+

+SIGNON statement for Inter-
 Directory Copy utility 435

A

abnormal termination 66
access linkpath
 in file header record 187
 specifying for
 Modify function 386
 Print function 368
 Unload function (Version 2)
 255
access mode
 direct 367, 384
 Modify function 384
 Print function 367
 sequential 367, 384
 serial 367, 384
access time 131
ACCESS-METHOD statement 79
arguments
 coding 27
 list of 27

B

backup copy 153, 333
BASE See Basic File Information
base statistics
 Depopulate function 117
 Reorganize function 129
 Sorted-Populate function 102
Basic File Information 138
batch Directory Maintenance See
 Directory Maintenance
BLANK-LINKS statement 254,
 263
BLKSIZE parameter
 LINKWKnn statement
 Insert Linkpath function 332
 Load function (Version 2) 309
 RECFORM statement for Load
 function (Version 2) 312
 Unload function (Version 2) 250
BLOCK-SIZE statement 76, 81
Boolean operators in CRITERIA
 statement 162, 373, 389
Burry's database
 internal schema 191, 335
 unloading and loading 200, 342

C

cataloged procedures
 TISDBTMC 40, 399
 TISDMCPY 445
 TISUTINS 325
 TISUTLOD 292
 TISUTNL 235
 TISUTUTL 20, 33
CHAIN See Chain Length
 Statistics Report
Chain Length Statistics report
 141
Chain Migration Statistics
 reports 143
 requesting 143
chains
 reorganizing 200, 342
 sequence retention when
 unloading 159, 256
CLEAR-LINKS statement
 Load function (Version 1) 173
 Unload function (Version 1) 160
CLEARLKS parameter 330

- CLOSE statement
 - File Statistics function 134
 - Modify function 384
 - Print function 366
- coded records
 - loading 319
 - unloading 163, 256
- coded related file
 - access linkpath 386
 - loading 177
 - modifying 385, 390
 - statistics 135, 146
- coding arguments 26, 27
- comments in UCL 25
- conceptual schema input
 - statement 435, 443
- conceptual schemas 443
- CONSOLE statement
 - control section of UCL 72
 - PDM Termination utility 401
- control section of UCL
 - ACCESS-METHOD statement 79
 - BLOCK-SIZE statement 76, 81
 - coding 58
 - CONSOLE statement 72
 - CONTROL statement 62
 - control statements for functions 86
 - DATA-FILE statement 73
 - DATA-FORMAT statement 71
 - DEVICE statement 77, 80
 - DEVICE-ADDRESS statement 80
 - DIAGNOSTICS statement 66
 - ENV-DESC statement 63
 - EXTENSION statement 69
 - FORMAT statement 65
 - FUNCTION statement 84
 - HEADER statement 69
 - LABEL statement 74
 - LINES statement 70
 - LIST statement 67
 - LOG-FILE statement 78
 - LOG-ID-ERROR statement 83
 - MEMORY statement 72
 - NOTIFY statement 73
 - PDM-ID-ERROR statement 82
 - RECORD-FORMAT statement 74
 - RECORD-SIZE statement 75
 - REPLY statement 73
 - SCHEMA statement 64
 - SEQ-ERROR statement 81
 - SORT statement 71
 - SUMMARY-DATA statement 77
 - SUPPRESS statement 70
- CONTROL statement 62
- control statements 328
- converted format files
 - Series 80 21, 22, 240
 - SUPRA 30, 87
 - unloading 225
- copy table input statement 435, 439
- CRITERIA statement
 - Modify function 389
 - Print function 373
 - Unload function (Version 1) 162
- CSI#REC file 236
- CSI#WK0n file 34
- CSI#WK1 file 36
- CSI#WKnn sort work files 236, 295
- CSIPARM file
 - coordinating with run control statements 230
 - Expand function 149
 - Insert Linkpath function 325
 - Load function (Version 2) 295
 - OS/390 34, 40
 - Unload function (Version 2) 236
 - unloading and loading
 - Directory files 231, 264, 322
 - PDM files 230
 - VSE 36, 41
- CSU#REC file
 - defining 238
 - Load function (Version 2) 295
- CSUAUX file
 - defining 238
 - files in converted and Series 80 format 238
 - Insert Linkpath function 325
 - Load function (Version 2) 295
 - Unload function (Version 2) 236, 238
- CSUINSRT 226, 322
- CSULOADR 226, 292
- CSUNLOAD 226, 235, 292

CSUOUTIL load module 20
 CSUPUNLD 264
 CSUSLOAD 322
 CSUSUNLD 264
 CSUULKUN 265
 CSUWORK file
 OS/390 34
 VSE 36
 CSUXSTAT 403, 417
 Current File Size information
 report 139
 CYLL parameter 240

D

damaged linkpath repair 19, 200,
 342
 data elements
 Modify function 391
 Print function 375
 data file format 184
 data members
 CSUPUNLD 264
 CSUSLOAD 322
 CSUSUNLD 264
 data parameters for exit
 programs 99, 114, 126
 data records
 exit programs
 adding with 183
 deleting with 182
 modifying with 182
 formatting 188
 order on data file 184
 DATA statement 392
 database files
 Burry's 191, 334
 formatting 87, 89
 loading 169
 statistics 131
 unloading 155, 225
 DATA-FILE statement 73
 DATA-FORMAT statement 71
 DATA-TYPE statement 176
 DBMNAME statement 401
 DBMOD parameter 330
 debug facility 31
 DEBUG function 32
 DEBUG parameter 32
 DEPOPULATE command 154,
 226

Depopulate function
 coding 105, 106
 examples 201, 214, 343, 353
 exit programs 112
 FILE statement 109
 FUNCTION statement 107
 general information 19, 21
 PURGE statement 111
 SECONDARY-KEY statement
 110
 STANDARD-EXIT statement
 108
 statistics 117
 STATISTICS statement 107
 DEVADDR parameter
 LINKWKnn statement
 Insert Linkpath function 332
 Load function (Version 2) 310
 RECFORM statement for Load
 function (Version 2) 313
 Unload function (Version 2) 251
 DEVICE parameter
 LINKWKnn statement
 Insert Linkpath function 331
 Load function (Version 2) 309
 RECFORM statement for Load
 function (Version 2) 312
 Unload function (Version 2) 250
 DEVICE statement 77, 80
 DEVICE-ADDRESS statement 80
 DIAGNOSTICS statement 66
 direct access mode 367, 384
 DIRECTION statement 175
 Directory copying 433
 Directory files
 coding CSIPARM file and run
 control statements 231
 expanding 147
 formatting 87, 89
 inserting linkpath data 322
 loading 171, 292, 322
 OS/390 34
 printing 363
 unloading 157, 225, 264
 VSE 36
 Directory Maintenance
 DEPOPULATE command 154,
 226
 POPULATE command 91, 154,
 226
 setting record size for INPUT
 file 404, 418

- DIRECTORY parameter 89
- domains
 - copying 444
 - input statement 435, 444
- DUMP parameter 44, 45
- DUMP statement 243, 244

- E**
- edit mask copying 439
- edit mask input statement 435, 439
- Element List statement 254, 257, 318
- ELEMENT statement
 - Load function (Version 1) 178
 - Modify function 391
 - Print function 375
 - Unload function (Version 1) 164
- elements
 - adding 165, 259
 - deleting 165, 259, 355
 - loading 318
 - modifying 381
 - selecting
 - for Load function (Version 1) 178
 - for Unload function (Version 1) 164
 - size
 - decreasing 166, 259
 - increasing 166, 259
 - unloading 257
- END parameter 330
- entry-sequenced data set (ESDS) 131
- ENV-DESC statement 63
- error
 - flag 28
 - in sort programs 56
 - number 28
 - PDM Termination Utility 399
 - pointer 28
- ESDS See entry-sequenced data set
- EUPD open mode 94, 109, 366, 384
- EXEC statement 43

- Execution Statistics utility
 - coding
 - for release 2.1.6 403
 - for release 2.4 417
 - file definitions 38
 - general information 19, 22
- exit point 10
 - sample programs 271
 - Unload function (Version 2) 265, 267
- exit point 20
 - sample programs 281
 - Unload function (Version 2) 265, 279
- exit point 30
 - sample programs 290
 - Unload function (Version 2) 265, 286
- exit point parameter lists
 - Depopulate function 112
 - Load function (Version 1) 50
 - Log-Print function 461
 - Modify function 50, 393
 - Print function 50, 376
 - Recover function 461
 - Reorganize function 124
 - Restore function 461
 - Sorted-Populate function 97, 99
 - Unload function (Version 1) 50
 - Version 1 Unload and Load functions 181
- exit programs
 - adding data records 183
 - conventions 50
 - data parameters 126
 - deleting
 - data records 182
 - elements 355
 - Depopulate function 108, 112
 - exit point 10 267
 - exit point 20 279
 - exit point 30 286
 - inserting into functions 49
 - Log-Print function 461
 - Modify function 393
 - modifying data records 182
 - Print function 376
 - Recover function 461
 - Reorganize function 124
 - Restore function 461
 - Sorted-Populate function 97

- exit programs (*cont.*)
 - Unload function (Version 2) 265
 - valid actions 100, 115, 127
 - Version 1 Load and Unload function 181
- Expand function
 - coding 147
 - examples 151
 - FILE statement 150
 - FUNCTION statement 149
 - general information 19, 21
- extended statistics
 - Depopulate function 117
 - Reorganize function 129
 - Sorted-Populate function 102
- EXTENSION statement 69

- F**
- FILABL parameter
 - LINKWKnn statement
 - Insert Linkpath function 332
 - Load function (Version 2) 309
 - RECFORM statement for Load function (Version 2) 312
 - Unload function (Version 2) 250
- file control statements
 - CSUPUNLD 264
 - Unload function (Version 2) 254, 317
- file definitions
 - in OS/390 33, 34, 38, 40
 - in VSE 36, 39, 41
- file header record
 - formatting 187
 - order on data file 184
- FILE parameter 239
- file performance optimization 131
- file pre-header record 184
- FILE statement
 - Depopulate function 109
 - Expand function 150
 - File Statistics function 133
 - Format function 88
 - Load function (Version 1) 171
 - Log-Print function 458
 - Modify function 383
 - Print function 365
 - Recover function 453
 - Reorganize function 122
 - Restore function 453
 - Review function 486
 - Sorted-Populate function 94
 - Unload function (Version 1) 157
 - Unlock function 490
- file statistics
 - group totals 413, 429
- File Statistics
 - reports 137
- File Statistics function
 - CLOSE statement 134
 - coding 131, 132
 - examples 136
 - FILE statement 133
 - FUNCTION statement 133
 - general information 19, 21
 - LINKPATH statement 134
 - reports 137
 - sort work space 55
 - STATISTICS statement 135
- file statistics page in Execution Statistics 411, 426
- file trailer records 184, 190
- files
 - building control record 87
 - closing
 - for Modify function 384
 - for Print function 366
 - coded related
 - access linkpath 386
 - loading 177
 - modifying 385, 390
 - statistics 135, 146
 - creating backup copies 153
 - defining
 - in OS/390 33
 - in OS/390 JCL 233, 293, 323
 - in VSE JCL 234, 294, 324
 - examples of unloading and loading 191
 - Execution Statistics utility 38, 404, 418
 - formatting
 - with Format function 87
 - with FORMAT statement 65
 - growth 131
 - inserting linkpath data 322
 - Load function (Version 2) 295
 - loading 292
 - opening
 - for Modify function 384
 - for Print function 366
 - PDM Termination utility 40

- files (*cont.*)
 - statistics 131, 411, 426
 - Unload function (Version 2) 235
 - unloading 155, 235
 - VSAM 292
 - FILES parameter 329, 330
 - FORCE statement 401
 - format
 - compatibility 87
 - converted 21, 22, 30, 87
 - data file 184
 - data records 188
 - records in SYSUT1 and INPUT file 320
 - Series 80 21, 22, 30, 240
 - SUPRA converted 21, 22, 30
 - SUPRA native 87, 131
 - UCL 23, 25
 - Format function
 - coding 87
 - FILE statement 88
 - FUNCTION statement 88
 - general information 19, 21
 - FORMAT statement 65
 - FUNCTION statement
 - control section of UCL 84
 - Depopulate function 107
 - Expand function 149
 - File Statistics function 133
 - Format function 88
 - Load function (Version 1) 170
 - Log-Print function 457
 - Modify function 383
 - Print function 365
 - Recover function 450
 - Reorganize function 120
 - Restore function 450
 - Review function 486
 - Sorted-Populate function 92
 - Unload function (Version 1) 156
 - Unlock function 490
 - functions
 - DEBUG 32
 - Depopulate
 - coding 105
 - examples 343, 353
 - general information 19, 21
 - executing 33
 - Expand 19, 21, 147
 - File Statistics 19, 21, 131
 - Format 19, 21, 87
 - Insert Linkpath 20, 22, 322
 - inserting exit programs in 49
 - Load 19
 - Load (Version 1) 20, 21, 169
 - Load (Version 2) 20, 22, 292
 - Log-Print 21, 447, 456
 - Modify 20, 21, 381
 - Print 20, 21, 363
 - Recover 21, 447
 - Reorganize 19, 21, 119
 - Restore 21, 447
 - review 485
 - Review 21
 - Sorted-Populate 19, 21, 91
 - UCL not required 22, 29
 - UCL required 23
 - Unload (Version 1) 20, 21, 155
 - Unload (Version 2) 20, 22
 - Unlock 21, 489
 - XTRACE 32
- G**
- group identification page 406, 420
- H**
- HEADER statement 69
 - heap area 46
 - hierarchical structure of UCL 24
- I**
- IDCAMS utility 292
 - IEBCOPY utility 434
 - ILBDSET0 routine 50
 - ILBOSTP0 routine 50
 - index files
 - density of blocks 96, 123
 - loading 171
 - reclaiming blocks 105, 111
 - unloading 154, 157
 - input file
 - defining in VSE 39
 - INPUT file
 - Execution Statistics utility 404, 418
 - Load function (Version 2) 296
 - OS/390 34, 38, 40
 - PDM Termination utility 400

- input files
 - defining
 - in OS/390 34, 38, 40
 - in VSE 36, 41
 - PDM Termination utility 40
 - input statements 435
 - Insert Linkpath function
 - BLKSIZE parameter 332
 - CLEARLKS parameter 330
 - coding 225, 322
 - control statements 328
 - CSIPARM file 325
 - CSUAUX file 325
 - DBMOD parameter 330
 - DEVADDR parameter 332
 - DEVICE parameter 331
 - END parameter 330
 - examples 333
 - FILABL parameter 332
 - FILES parameter 329
 - general information 20, 22
 - INSERT statement 328
 - LINKWK01/LNKWRK1 file 325, 326
 - LINKWK02/LNKWRK2 file 325, 327
 - LINKWKnn statement 328
 - LINKWKnn statements 331
 - RECSIZE parameter 332
 - SYSIN file 325
 - SYSIPT file 325
 - SYSLST file 325
 - SYSPRINT file 325
 - SYSUDUMP file 325
 - use 226
 - INSERT statement 328
 - Inter-Directory Copy utility
 - coding 433
 - input statements 435
 - conceptual schema input statement 443
 - copy table input statement 439
 - domain input statement 444
 - edit mask input statement 439
 - example 446
 - executing
 - in OS/390 445
 - in VSE 445
 - general information 22
 - schema input statement 442
 - security group input statement 441
 - signon input statement 438
 - user input statement 440
 - IOBUF parameter 44
 - IUPD open mode 366
- J**
- JCL 232
 - JOB statement 43
- K**
- KEY statement
 - Modify function 387
 - Print function 369
 - KEY-RANGE statement
 - Log-Print function 460
 - Recover function 455
 - Restore function 455
 - keys
 - secondary
 - deleting 87, 105
 - depopulating 226
 - index files 19
 - tree structure 119
 - selecting to access files 387
 - key-sequenced data set (KSDS)
 - basic information 138
 - current size 139
 - formatting 89
 - statistics 131, 140
 - KSDS See key-sequenced data set
- L**
- LABEL statement 74
 - LIBDEF file
 - VSE 36, 39, 41
 - library files
 - Execution Statistics utility 38, 39
 - OS/390 34
 - PDM Termination utility 40, 41
 - LINES statement 70
 - LINK See Linkpath Statistics
 - linkage verification 131

- linkage work files
 - allocating space
 - Insert Linkpath function 326
 - LINKWK01/LNKWRK1 297
 - Load function (Version 2) 297
 - defining characteristics 331
 - format 326
 - LINKWK02/LNKWRK2 300
- linkdeck CSUULKUN 265
- LINKPATH statement
 - after Element List statement 319
 - File Statistics function 134
 - Load function (Version 1) 172
 - Load function (Version 2) 321
 - Modify function 386
 - Print function 368
 - Unload function (Version 1) 158
 - Unload function (Version 2) 254, 255
- linkpaths
 - blanking in primary files 160, 263
 - blanking with CLEARLKS parameter 330
 - chains 200, 342
 - clearing
 - with BLANK-LINKS statement 263
 - with Modify function 157, 171, 381, 394
 - data 22, 226, 322, 358
 - modifying 381
 - printing 363
 - repairing by unloading and loading files 200, 342
 - selecting primary to load file 321
 - statistics 140
 - unloading and loading files 227, 228
 - updating 20, 292, 381
 - with Modify function 217
- LINKWK01/LNKWRK1 file
 - coding 297
 - defining 326
 - Insert Linkpath function 325
 - Load function (Version 2) 296
- LINKWK02/LNKWRK2 file
 - coding 300
 - defining 327
- Insert Linkpath function 325
- Load function (Version 2) 296, 300
- LINKWKnn statement
 - Insert Linkpath function 331
 - Load function (Load function (Version 2) 303
 - Load function (Version 2) 308
- LINKWKnn statements
 - Insert Linkpath function 328
- list of arguments 27
- LIST statement 67
- Load function (Version 1)
 - CLEAR-LINKS statement 173
 - coding 169
 - DATA-TYPE statement 176
 - DIRECTION statement 175
 - ELEMENT statement 178
 - examples 191
 - exit programs 181
 - FILE statement 171
 - FUNCTION statement 170
 - general information 19, 20, 21
 - LINKPATH statement 172
 - RECORD statement 177
 - retaining data file format 184
 - SEQUENCE statement 174
 - sort work space 54
 - sorting unloaded record 153
 - STANDARD-EXIT statement 170
- Load function (Version 2)
 - BLKSIZE parameter 309, 312
 - coding 225, 292
 - CSI#WKnn sort work files 295
 - CSIPARM file 295
 - CSU#REC file 295
 - CSUAUX file 295
 - DEVADDR parameter 310, 313
 - DEVICE parameter 309, 312
 - Element List statement 318
 - examples 333
 - FILABL parameter 309, 312
 - file control statements 317
 - general information 19, 20, 22
 - INPUT file 296
 - LINKPATH statement 321
 - LINKWK01/LNKWRK1 file 296, 297
 - LINKWK02/LNKWRK2 file 296, 300

Load function (Version 2) (*cont.*)

- LINKWKnn statements 303, 308
- LRECL parameter 299
- MAXKEY statement 303, 307
- PRIMARY statement 303, 306
- RECFORM statement 303, 311
- RECSIZE parameter 310, 313
- RELATED statement 303, 305
- run control statements 302
- SCHEMA statement 303, 304
- S-E statement 303, 307
- SORTCORE statement 303, 314
- SORTLIB file 296
- SORTNAME statement 303, 315
- SORTWKnn file 296
- SYSIN file 296
- SYSIPT file 296
- SYSLST file 296
- SYSOUT file 296
- SYSPRINT file 296
- SYSUDUMP file 296
- SYSUT1 file 296
- V-E statement 303, 306
- WORK statement 303, 316
- load modules
 - CSUOUTIL 20
- LOAD parameter 240
- LOAD-DENSITY statement
 - Reorganize function 123
 - Sorted-Populate function 96
- LOG-FILE statement 78
- LOG-ID-ERROR statement 83
- Log-Print function
 - coding 447, 456
 - examples 481
 - exit programs 461
 - FILE statement 458
 - FUNCTION statement 457
 - general information 21
 - KEY-RANGE statement 460
 - RRN-RANGE statement 459
 - STANDARD-EXIT statement 457
 - STATISTIC statement 457
 - use 448
- LRECEL parameter 299

M

- maximum number
 - modified records 388
 - printed records 372
- MAXIMUM statement
 - Modify function 388
 - Print function 372
- MAXKEY statement 303, 307
- memory for sort program 53, 72
- MEMORY statement 53, 72
- Modify function
 - clearing linkpaths 157
 - CLOSE statement 384
 - CRITERIA statement 389
 - DATA statement 392
 - ELEMENT statement 391
 - examples 191
 - exit programs 393
 - FILE statement 383
 - FUNCTION statement 383
 - general information 20, 21
 - KEY statement 387
 - LINKPATH statement 386
 - MAXIMUM statement 388
 - OPEN-MODE statement 384
 - QUALIFIER statement 384
 - RECORD statement 390
 - RRN statement 386
 - STANDARD-EXIT statement 383
 - using to clear linkpaths 171
- Modify function examples 394
- Modify Schema utility 149

N

- native format fields
 - converting
 - Series 80 files 21, 22
 - SUPRA converted files 21, 22
 - unloading and loading 153, 225
- native format file expansion 147
- NEW-SCHEMA/NEW-ENVDESC statement 243, 245
- NODUMP parameter 44, 45
- NONE open mode 230, 295
- non-UCL utilities 22
- NOSPIE parameter 44, 45
- NOTIFY statement 73
- null arguments 26

O

open mode
 EUPD 94, 109, 366, 384
 IUPD 366
 NONE 230, 295
 READ 366
 SUPD 366, 384
 OPEN-FILE statement
 Recover function 451
 Restore function 451
 OPEN-MODE statement
 Modify function 384
 Print function 366
 optional input statements 437
 OUTFILE
 defining 241
 Unload function (Version 2) 257
 OUTFILE file 236
 out-of-block synonym records
 minimizing by unloading and
 loading files 200, 300, 342
 number 300, 327
 OUTPUT file
 defining in VSE 41
 Execution Statistics Utility 38,
 39
 OS/390 34, 38, 40
 PDM Termination Utility 40
 Unload function (Version 2)
 241, 249
 VSE 36, 39

P

parameter lists for exit points
 Depopulate function 112
 Load function (Version 1) 50
 Log-Print function 461
 Modify function 50, 393
 Print function 50, 376
 Recover function 461
 Reorganize function 124
 Restore function 461
 Sorted-Populate function 97
 Unload function (Version 1) 50
 Version 1 Unload and Load
 functions 181

parameters

*FILL
 changing length of elements
 170, 179, 258
 equalizing length of elements
 156
 example 260
 exit programs 280, 286
 Unload function (Version 1)
 165
 unloading files 354
 BLKSIZE 250, 309, 312, 332
 CLEARLKS 330
 CYLL 240
 DBMOD 330
 DEBUG 32
 DEVADDR 251, 310, 313, 332
 DEVICE 250, 309, 312, 331
 DIRECTORY 89
 DUMP 44, 45
 END 330
 FILABL 250, 309, 312, 332
 FILE 239
 FILES 329, 330
 IOBUF 44
 LOAD 240
 LRECL 299
 NODUMP 44, 45
 NOSPIE 44, 45
 PASSWORD 438
 PRESERVE 256
 RC 256
 RCYL 239
 REALM 89
 RECSIZE 251, 310, 313, 332
 REGION 43
 SIZE 43
 SPIE 44, 45
 STACK 44, 46
 XTRACE 32
 PARM file 237
 PASSWORD parameter 438
 PASSWORD statement 400
 PDM See Physical Data Manager
 PDM files
 coding CSIPARM file and run
 control statements 230
 inserting linkpath data 322
 loading 292
 OS/390 34
 unloading 235
 VSE 36

- PDM termination utility
 - file definitions 40
 - PDM Termination utility
 - CONSOLE statement 401
 - DBMNAME statement 401
 - example 402
 - FORCE statement 401
 - general information 22
 - PASSWORD statement 400
 - PDM Termination Utility
 - coding 399, 400
 - PDM-ID-ERROR statement 82
 - Physical Data Manager (PDM)
 - identification page in Execution Statistics 406, 420
 - shutting down
 - active tasks 401
 - with PDM Termination Utility 399
 - POPULATE command 91, 154, 226
 - pre-header record 186
 - PRESERVE parameter 256
 - PRESERVE statement 159
 - primary files
 - basic information on 138
 - chain length statistics 141
 - Chain Migration Statistics 143
 - expanding 147
 - inserting linkpath data 325, 328
 - loading 306
 - modifying 381
 - printing 363
 - synonym statistics 145
 - unloading 158, 248
 - primary linkpath 322
 - PRIMARY statement 243, 248, 303, 306
 - Print function
 - CLOSE statement 366
 - coding UCL for 363
 - CRITERIA statement 373
 - ELEMENT statement 375
 - examples 377
 - exit programs 376
 - FILE statement 365
 - FUNCTION statement 365
 - general information 20, 21
 - KEY statement 369
 - LINKPATH statement 368
 - MAXIMUM statement 372
 - OPEN-MODE statement 366
 - QUALIFIER statement 367
 - RECORD statement 374
 - RRN statement 370
 - RRN-RANGE 371
 - STANDARD-EXIT statement 365
 - processing environment 57
 - PURGE statement 111
- Q**
- QUALIFIER statement
 - Modify function 384
 - Print function 367
- R**
- RC parameter 256
 - RCYL parameter 239
 - READ open mode 366
 - REALM parameter 89, 149
 - RECFORM statement
 - Load function (Version 2) 303, 311
 - Unload function (Version 2) 243, 249
 - Record Code Statistics
 - reports 146
 - requesting 146
 - record format of SYSUT1 and INPUT file 320
 - RECORD statement
 - Load function (Version 1) 177
 - Modify function 390
 - Print function 374
 - Unload function (Version 1) 163
 - RECORD-FORMAT statement 74
 - records
 - file control 87
 - format 320
 - maximum number modified 388
 - maximum number printed 372
 - modifying 381
 - printing 363
 - selecting
 - for Load function (Version 1) 177
 - for Modify function 389
 - for Print function 373
 - for Unload function (Version 1) 163

- RECORD-SIZE statement 75
- Recover function
 - coding 447, 449
 - examples 471
 - exit programs 461
 - FILE statement 453
 - FUNCTION statement 450
 - general information 21
 - KEY-RANGE statement 455
 - OPEN-FILE statement 451
 - RRN-RANGE statement 454
 - STANDARD-EXIT statement 451
 - STATE statement 450
 - STATISTIC statement 452
- RECSIZE parameter
 - LINKWKnn statement
 - Insert Linkpath function 332
 - Load function (Version 2) 310
 - RECFORM statement for Load function (Version 2) 313
 - Unload function (Version 2) 251
- REGION parameter 43
- registers
 - exit point 10 269
 - exit point 20 280
 - exit point 30 288
- related files
 - basic information on 138
 - chain length statistics 142
 - Chain Migration Statistics 144
 - expanding 147
 - formatting 87
 - loading 171, 305
 - modifying 381
 - printing 363, 370
 - RRN selection 386
 - unloading 158, 251, 258
- RELATED statement 243, 247, 303, 305
- relative record numbers (RRN)
 - selecting
 - in related file 370, 386
 - in Unload function (Version 1) 161
 - specifying range to retrieve 371
- Reorganize function
 - coding 119
 - exit programs 124
 - FILE statement 122
 - FUNCTION statement 120
 - general information 19, 21
 - LOAD-DENSITY statement 123
 - requesting statistics 129
 - SECONDARY-KEY statement 123
 - STANDARD-EXIT statement 121
 - STATISTICS statement 120
- REPLY statement 73
- Restore function
 - coding 447, 449
 - examples 475
 - exit programs 461
 - FILE statement 453
 - FUNCTION statement 450
 - general information 21
 - KEY-RANGE statement 455
 - OPEN-FILE statement 451
 - RRN-RANGE statement 454
 - STANDARD-EXIT statement 451
 - STATE statement 450
 - STATISTIC statement 452
- Review function
 - coding 485, 486
 - example 487
 - FILE statement 486
 - FUNCTION statement 486
 - general information 21
- RQLOC value 189
- RRN See relative record number
- RRN statement
 - Modify function 386
 - Print function 370
- RRN-RANGE statement
 - Log-Print function 459
 - Print function 371
 - Recover function 454
 - Restore function 454
 - Unload function (Version 1) 161
- RSTAT command 403, 417
- run control record 184, 185
- run control statements
 - coding Load function (Version 2) 302
 - coordinating with CSIPARM file 230
 - CSUSUNLD 264
 - Unload function (Version 2) 242, 243
 - unloading and loading PDM files 230
 - validating 253

run-time interface parameters 44
 run-time options
 choosing 42
 examples 42

S

sample JCL members
 TXJDRCPY 445
 TXJPSTAT 404
 TXJSHUTP 399
 schema copying 442
 schema input statement 435, 442
 SCHEMA statement
 control section of UCL 64
 Load function (Version 2) 304
 Load function (Version 2) 303
 S-E statement
 Load function (Version 2) 303,
 307
 Unload function (Version 2)
 243, 249
 secondary keys
 creating 91
 deleting 21, 87, 105
 depopulating 105, 153, 226,
 292
 tree structure 21, 91, 119
 SECONDARY-KEY statement
 Depopulate function 110
 Reorganize function 123
 Sorted-Populate function 95
 security group input statement
 435, 441
 SEQ-ERROR statement 81
 SEQUENCE statement 174
 sequential access mode 367,
 384
 serial access mode 367, 384
 Series 80 format files
 executing utilities 30
 inserting linkpath data 22
 loading 21, 22
 modifying 21
 Unload function (Version 2) 240
 unloading 21, 22, 225
 signon input statement 438
 SIZE *See* Current File Size
 SIZE parameter 43
 sort fields 174
 sort memory 53, 72

SORT program
 exit point 20 279
 specifying for Load function
 (Version 2) 314
 virtual storage 314
 sort programs
 allocating space 153
 errors 56
 using 52
 sort sequence 174
 SORT statement
 control section of UCL 71
 retaining values in 270
 sort programs 52
 Unload function (Version 2) 267
 sort work files
 CSI#WKnn 236, 295
 sort work space
 allocating 54
 File Statistics function 55
 Load function (Version 1) 54
 Sorted-Populate function 54
 SORTCORE statement 303, 314
 Sorted-Populate function
 coding 91, 92
 examples 206, 219, 348, 359
 exit programs for 97
 FILE statement 94
 FUNCTION statement 92
 general information 19, 21
 LOAD-DENSITY statement 96
 requesting statistics 102
 SECONDARY-KEY statement
 95
 sort work space 54
 STANDARD-EXIT statement 93
 STATISTICS statement 93
 SORTLIB file
 Load function (Version 2) 296
 OS/390 34
 Unload function (Version 2) 237
 SORTNAME statement
 Load function (Version 2) 303,
 315
 Unload function (Version 2)
 243, 252
 SORTWKnn file
 Load function (Version 2) 296
 Unload function (Version 2) 237
 SPIE parameter 44, 45

- STACK parameter
 - estimating size 47
 - use 44, 46
- stack/heap area 46
- STANDARD-EXIT statement
 - Depopulate function 108
 - exit programs 50
 - Load function (Version 1) 170
 - Log-Print function 457
 - Modify function 383
 - Print function 365
 - Recover function 451
 - Reorganize function 121
 - Restore function 451
 - Sorted-Populate function 93
 - Unload function (Version 1) 156
- STATE statement
 - Recover function 450
 - Restore function 450
- statements
 - +SIGNON 435
 - ACCESS-METHOD 79
 - BLANK-LINKS 254, 263
 - BLOCK-SIZE 76, 81
 - CLEAR-LINKS 160, 173
 - CLOSE 134, 366, 384
 - conceptual schema input 435, 443
 - CONSOLE 72, 401
 - CONTROL 62
 - copy table input 435, 439
 - CRITERIA 162, 373, 389
 - DATA 392
 - DATA FORMAT 71
 - DATA-FILE 73
 - DATA-TYPE 176
 - DBMNAME 401
 - DEVICE 77, 80
 - DEVICE-ADDRESS 80
 - DIAGNOSTICS 66
 - DIRECTION 175
 - domain input 435
 - DUMP 243, 244
 - edit mask input 435, 439
 - ELEMENT 164, 178, 375, 391
 - Element List 254, 257, 318
 - ENV-DESC 63
 - EXEC 43
 - EXTENSION 69
- FILE
 - Depopulate function 109
 - Expand function 150
 - File Statistics function 133
 - Format function 88
 - Load function (Version 1) 171
 - Log-Print function 458
 - Modify function 383
 - Print function 365
 - Recover function 453
 - Reorganize function 122
 - Restore function 453
 - Review function 486
 - Sorted-Populate function 94
 - Unload function (Version 1) 157
 - Unlock function 490
- file control 254, 317
- FORCE 401
- FORMAT 65
- FUNCTION 84
 - Depopulate function 107
 - Expand function 149
 - File Statistics function 133
 - Format function 88
 - Load function (Version 1) 170
 - Log-Print function 457
 - Modify function 383
 - Print function 365
 - Recover function 450
 - Reorganize function 120
 - Restore function 450
 - Review function 486
 - Sorted-Populate function 92
 - Unload function (Version 1) 156
 - Unlock function 490
- HEADER 69
- INSERT 328
- JOB 43
- KEY 369, 387
- KEY-RANGE 455, 460
- LABEL 74
- LINES 70

statements (*cont.*)

LINKPATH

- File Statistics function 134
- Load function (Version 1) 172
- Load function (Version 2) 321
- Modify function 386
- Print function 368
- Unload function (Version 1) 158
- Unload function (Version 2) 254, 255

LINKWKnn 303, 308, 328, 331

LIST 67

LOAD-DENSITY 96, 123

LOG-FILE 78

LOG-ID-ERROR 83

MAXIMUM 372, 388

MAXKEY 303, 307

MEMORY 53, 72

NEW-SCHEMA/NEW-ENVDESC 243, 245

NOTIFY 73

OPEN-FILE 451

OPEN-MODE 366, 384

optional input 437

PASSWORD 400

PDM-ID-ERROR 82

PRESERVE 159

PRIMARY: 243, 248, 303, 306

PURGE 111

QUALIFIER 367, 384

RECFORM 243, 249, 303, 311

RECORD 163, 177, 374, 390

RECORD-FORMAT 74

RECORD-SIZE 75

RELATED: 243, 247, 303, 305

REPLY 73

RRN 370, 386

RRN-RANGE 161, 371, 454, 459

run control 243, 302

SCHEMA 64, 303, 304

schema input 435, 442

S-E 243, 249, 303, 307

SECONDARY-KEY 95, 110, 123

security group 435

security group input 441

SEQ-ERROR 81

SEQUENCE 174

signon input 438

SORT 52, 71, 267

SORTCORE 303, 314

SORTNAME 243, 252, 303, 315

STANDARD-EXIT

- Depopulate function 108
- exit programs 50
- Load function (Version 1) 170
- Log-Print function 457
- Modify function 383
- Print function 365
- Recover function 451
- Reorganize function 121
- Restore function 451
- Sorted-Populate function 93
- Unload function (Version 1) 156

STATE 450

STATISTIC 452, 457

STATISTICS 93, 107, 120, 135

subordinate 24

SUMMARY-DATA 77

superordinate 24

SUPPRESS 70

TEST 243, 253

user input 435, 440

V-E 243, 248, 303, 306

WORK 243, 253, 303, 316

STATISTIC statement

Log-Print function 457

Recover function 452

Restore function 452

statistics

base 102, 117, 129

Basic File Information 138

chain 141

collecting with exit program

376, 393

Current File Size 139

Depopulate function 117

extended 102, 117, 129

file 131, 411, 426

generating 403, 417

record codes 146

Reorganize function 129

requesting 102, 117

Sorted-Populate function 102

synonym 145

system 407, 421

- statistics reports
 - content 137
 - format 137
 - generating with Execution
 - Statistics utility 403, 417
- STATISTICS statement
 - Depopulate function 107
 - File Statistics function 135
 - Reorganize function 120
 - Sorted-Populate function 93
- STATS file 418
 - Execution Statistics utility 404
 - OS/390 38
 - VSE 39
- STEPLIB file
 - OS/390 34, 38, 40
- storage 43
- subordinate statements 24
- SUMMARY-DATA statement 77
- SUPD open mode 366, 384
- superordinate statement 24
- SUPPRESS statement 70
- SUPRA native format fields
 - converting Series 80 files 21
 - converting SUPRA converted files 21
 - unloading and loading 153, 225
- synonym
 - chain reorganization 200
 - records 300
 - statistics
 - reports 145
 - requesting 145
- SYSIN file
 - Insert Linkpath function 325
 - Load function (Version 2) 296
 - Unload function (Version 2) 237
- SYSIPT file
 - Insert Linkpath function 325
 - Load function (Version 2) 296
 - Unload function (Version 2) 237
 - VSE 36, 39, 41
- SYSLST file
 - Insert Linkpath function 325
 - Load function (Version 2) 296
 - Unload function (Version 2) 237
 - VSE 36, 39, 41
- SYSOUT file
 - Load function (Version 2) 296
 - OS/390 34
 - Unload function (Version 2) 237

- SYSPRINT file
 - Insert Linkpath function 325
 - Load function (Version 2) 296
 - Unload function (Version 2) 237
- System Log File 447
- System Log file format 89
- System Log File format 87
- system statistics page 407, 421
- SYSUDUMP file
 - Insert Linkpath function 325
 - Load function (Version 2) 296
 - OS/390 34, 38, 40
 - Unload function (Version 2) 237
- SYSUT1 file 296

T

- tables 439
- Task Level Recovery 450, 485
- Task Log File format 87, 89
- termination 66
- termination page in Execution
 - Statistics 416, 432
- TEST statement 243, 253
- TISDBTMC cataloged procedure 40, 399
- TISDMCPY cataloged procedure 445
- TISUTINS cataloged procedure 325
- TISUTLOD cataloged procedure 292
- TISUTUNL cataloged procedure 235
- TISUTUTL cataloged procedure 20, 33
- trace facility 31
- TRACE facility 32
- tree structure of secondary keys 91, 119
- TXJDRCPY sample JCL member 445
- TXJPSTAT sample JCL member 38, 404, 418
- TXJSHUTP sample JCL member 399

U

- UCL See Utility Command Language
- Unload function
 - general information 20
- Unload function (Version 1)
 - CLEAR-LINKS statement 160
 - coding 153, 155
 - CRITERIA statement 162
 - ELEMENT statement 164
 - examples 191
 - exit programs 181
 - FILE statement 157
 - FUNCTION statement 156
 - general information 20, 21
 - LINKPATH statement 158
 - PRESERVE statement 159
 - RECORD statement 163
 - retaining data file format 184
 - RRN-RANGE statement 161
 - STANDARD-EXIT statement 156
- Unload function (Version 2)
 - BLANK-LINKS statement 254, 263
 - BLKSIZE parameter 250
 - coding 225, 235
 - CSI#WKnn sort work files 236
 - CSIPARM file 236
 - CSU#REC file 236, 238
 - CSUAUX file 236, 238
 - CYLL parameter 240
 - DEVADDR parameter 251
 - DEVICE parameter 250
 - DUMP statement 243, 244
 - Element List statement 257
 - ELEMENT list statement 254
 - examples 333
 - exit points 265
 - FILABL parameter 250
 - file control statements 254
 - FILE parameter 239
 - general information 20, 22
 - LINKPATH statement 254, 255
 - LOAD parameter 240
 - NEW-SCHEMA/NEW-ENVDDESC statement 243, 245
 - OUTFILE file 236, 241
 - OUTPUT file 249
 - PARM file 237
 - PRESERVE parameter 256
 - PRIMARY statement 243, 248
 - RC parameter 256
 - RCYL parameter 239
 - RECFORM statement 243, 249
 - RECSIZE parameter 251
 - RELATED statement 243, 247
 - run control statements 242
 - S-E statement 243, 249
 - sorting in 265, 267
 - SORTLIB file 237
 - SORTNAME statement 243, 252
 - SORTWKn file 237
 - SYSIN file 237
 - SYSIPT file 237
 - SYSLST file 237
 - SYSOUT file 237
 - SYSPRINT file 237
 - SYSUDUMP file 237
 - TEST statement 243, 253
 - V-E statement 243, 248
 - WORK statement 243, 253
- Unlock function
 - coding 489
 - examples 491
 - FILE statement 490
 - FUNCTION statement 490
 - general information 21
 - user input statement 435, 440
- utilities
 - Execution Statistics 19, 22, 403, 417
 - functions 21
 - Inter-Directory Copy 22, 433
 - PDM termination 399
 - PDM Termination 22

Utility Command Language
(UCL)

- coding
 - control section 58
 - Depopulate function 106
 - File Statistics function 132
 - Load function (Version 1) 169
 - Log-Print function 456
 - Modify function 282
 - Print function 363
 - Recover function 449
 - Reorganize function 119
 - Restore function 449
 - Review function 486
 - Sorted-Populate function 92
 - Unload function (Version 1)
155
- coding control section 57
- control statements for functions
86
- executing functions that require
23
- executing utilities that do not
require 29
- formatting 25
- general information 20
- hierarchical structure 24
- programs
 - comments in 25
 - formatting 25
 - hierarchical structure 24
- sample program 57
- submitting programs 20

V

- validating programs 28
- values in SORT statement
 - retaining 270
- V-E statement 243, 248, 303,
306
- Version 1 Load function *See*
Load function (Version 1)
- Version 1 Unload function *See*
Unload function (Version 1)
- Version 2 Load function *See*
Load function (Version 2)
- Version 2 Unload function *See*
Unload function (Version 2)
- virtual storage for SORT program
314
- VSAM file 292

W

- work files 34, 36
- WORK statement
 - Load function (Version 2) 303,
316
 - Unload function (Version 2)
243, 253

X

- XTRACE function 32
- XTRACE parameter 32