

Cincom

SUPRA SERVER PDM

PDM DML Programming Guide
(OS/390 & VSE)

P26-4340-64



SUPRA[®] Server PDM DML Programming Guide (OS/390 & VSE)

Publication Number P26-4340-64

© 1989–1998, 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This document contains unpublished, confidential, and proprietary information of Cincom.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. Contact your Cincom representative to be certain the items are available to you.

Release information for this manual

The *SUPRA Server PDM DML Programming Guide (OS/390 & VSE)*, P26-4340-64, 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	ix
Using this document.....	ix
Document organization	x
Revisions to this manual	x
Conventions	xi
SUPRA Server documentation series	xiv
An overview of DML concepts	17
Opening and closing files	19
Opening and closing primary and related files	19
Opening and closing secondary key (index) files	21
Adding a primary record	22
Reading a primary record.....	23
Updating a primary record.....	25
Deleting a primary record.....	25
Adding a related record	26
Reading a related record.....	30
Updating a related record.....	32
Deleting a related record	33
Reading records via secondary keys	35
Diagnosing application DML errors	36
Maintaining database integrity in a task logging environment.....	40
Logical unit of work	40
Record holding	42
Program recovery.....	46
Using system control DML commands	47
Opening and closing files with PDM DML	48
Using system logging commands	49
Generating statistics.....	51
Monitoring resources.....	53
Terminating the PDM with PDM DML	58

CALL statements and data lists	59
CALL statements	60
Data list parameter keywords	62
Command syntax	67
ADD-M	68
ADDVA	72
ADDVB	77
ADDVC	82
ADDVR	86
CLOSX	93
COMIT	100
DEL-M	104
DELVD	107
ENDLG	112
ENDTO	116
FINDX	121
FINDX qualifier for BDAM or ESDS primary files	129
FINDX qualifier for KSDS primary files	130
FINDX qualifier for related files	133
FREEEX	137
MARKL	138
OPENX	140
QMARK	149
QUIET	151
RDNXT	152
RDNXT qualifier for BDAM or ESDS primary files	157
RDNXT qualifier for KSDS primary files	158
RDNXT qualifier for related files	161
READD	165
READM	169
READR	172
READV	179
READX	187
RESET	197
RQLOC	201
RSTAT	203
SHOWX	221
SHOWX for status returns	221
SHOWX for monitoring resources	224
SINOF	248
SINON	250
SINON (CICS compatibility)	256
SINON (TIS 1.x compatibility)	258
SINON (TOTAL compatibility)	260

WRITD	262
WRITM	264
WRITV	268
Programming examples	273
Example of IDENTIFICATION, ENVIRONMENT, and DATA divisions	274
Example of read-only environment	278
Example of update mode	283
Example of recoverable update mode	291
Example of primary serial processing	300
Example of related serial processing (physical).....	303
Example of related serial processing (logical).....	305
Index	307

About this book

Using this document

This manual incorporates applicable material from the *SUPRA Server PDM Systems Control Commands Supplement*, P26-2215, which has been discontinued. This manual now contains all PDM Data Manipulation Language (DML) commands.

This manual is intended for new users and existing users who are upgrading from SUPRA, TOTAL, or TIS. It contains information for database administration (DBA) and application programming personnel. It allows you to maintain your existing applications that issue PDM DML (Data Manipulation Language) commands. For compatibility differences, refer to the *SUPRA Server PDM Migration Guide (OS/390 & VSE)*, P26-0550.

This programming guide contains an overview of concepts of the PDM DML commands, instructions for coding a call and data-list, the complete syntax of DML commands, and programming examples.

The terms OS/390 and VSE in this manual refer to the following:

- ◆ OS/390 to OS/390/XA, OS/390/ESA, and OS/390/OS390
- ◆ VSE to VSE Advanced Functions, and VSE/ESA

The OS/390 term STEPLIB is used in this manual. The corresponding concept for VSE is the LIBDEF in effect during the particular step. Also, the OS/390 term address space is used; the corresponding term for VSE is partition.

Document organization

The information in this manual is organized as follows:

Chapter 1—An overview of DML concepts

Describes DML and how to access data in a SUPRA PDM database.

Chapter 2—CALL statements and data lists

Describes how to make requests to the database by coding CALL statements to DATBAS, the interface used for access to SUPRA Server databases. Also discusses the transfer of data between the database record and the program-supplied data area.

Chapter 3—Command syntax

Contains syntax descriptions for the PDM DML commands.

Chapter 4—Programming examples

Contains DML coding examples.

Index

Revisions to this manual

The following changes have been made for this release:

- ◆ Removed statements in descriptions of the *control-key* parameter that said the PDM returns an error message then “changes the *qualifier* to BEGN...” under “**RDNXT qualifier for BDAM or ESDS primary files**” on page 157 and “**RDNXT qualifier for KSDS primary files**” on page 158.
- ◆ The CNTRL command has been removed. This is no longer supported by SUPRA PDM.
- ◆ A consideration has been added to the *data-list* parameter under “**READX**” on page 187.

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.	BEGN b b b b SERIAL
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.	[WHERE <i>search-condition</i>]
	The example indicates that you can optionally enter a WHERE clause.	
	Stacked items enclosed by brackets represent optional alternatives, one of which can be selected.	<u>(WAIT)</u> (NOWAIT)
	The example indicates that you can optionally enter either WAIT or NOWAIT. (WAIT is underlined to signify that it is the default.)	

Convention	Description	Example
Braces { }	<p>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.</p> <p>The example indicates that you must enter ON or OFF when using the MONITOR statement.</p>	<p>MONITOR { ON } { OFF }</p>
<p><u>Underlining</u> (In syntax)</p>	<p>Indicates the default value supplied when you omit a parameter.</p> <p>The example indicates that if you do not choose a parameter, the system defaults to WAIT.</p> <p>Underlining also indicates an allowable abbreviation or the shortest truncation allowed.</p> <p>The example indicates that you can enter either STAT or STATISTICS.</p>	<p>[(WAIT)] [(NOWAIT)]</p> <p><u>STATISTICS</u></p>
Ellipsis points...	<p>Indicate that the preceding item can be repeated.</p> <p>The example indicates that you can enter multiple host variables and associated indicator variables.</p>	<p>INTO :host-variable [:ind-variable],...</p>

Convention	Description	Example
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. The example indicates that 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, password, db-name)</i> INFILE 'Cust.Memo' CONTROL LEN4
SMALL CAPS	Represent a required keystroke. Multiple keystrokes are hyphenated.	ALT-TAB
<div style="border: 1px solid black; padding: 2px; display: inline-block; margin-bottom: 5px;">OS/390</div> <div style="border: 1px solid black; padding: 2px; display: inline-block;">VSE</div>	Information specific to a certain operating system is flagged by a symbol in a shadowed box (<div style="border: 1px solid black; padding: 2px; display: inline-block;">OS/390</div>) indicating which operating system is being discussed. Skip any information that does not pertain to your environment.	<div style="border: 1px solid black; padding: 2px; display: inline-block; margin-bottom: 10px;">OS/390</div> See the SUPRA Server procedure library member TIS\$RDM for a list of RDM procedures. <div style="border: 1px solid black; padding: 2px; display: inline-block;">VSE</div> 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 PDM Digest (OS/390 & VSE)*, P26-9062.

Overview

- ◆ *SUPRA Server PDM 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

An overview of DML concepts

DML means Data Manipulation Language. To access data in a SUPRA physical manager data (PDM) database, you write an application program using one of two DML languages. The languages process through the optional Relational Data Manager (RDM) or the PDM (Physical Data Manager). With the RDM, you use Relational Data Manipulation Language (RDML). With the PDM, you use PDM Data Manipulation Language (PDM DML).

Do not mix RDML and PDM DML within an application because it can corrupt the database. However, RDML applications and PDM DML applications can coexist in SUPRA Server operation. Be aware that RDM relational integrity is not observed for PDM DML applications. PDM DML does not use the RDM component of SUPRA Server and there are no logical views of data. For information on RDML, refer to the *SUPRA Server PDM RDM COBOL Programmer's Guide (OS/390 & VSE)*, P26-8330, or to the *SUPRA Server PDM RDM PL/1 Programming Guide (OS/390 & VSE)*, P26-8331.

Your existing TOTAL, TIS, and SUPRA 1.x PDM DML applications can be executed under SUPRA 2.x. For compatibility considerations, refer to the *SUPRA Server PDM Migration Guide (OS/390 & VSE)*, P26-0550.

With PDM DML, you process by accessing one file at a time with one DML command. You code a CALL to DATBAS for each command (see “*CALL statements and data lists*” on page 59 for coding a CALL). You link DATBAS with your program after its compilation. Two modules are delivered with SUPRA, one for batch programs (DATBAS) and one for CICS programs (DATBASC).

The first PDM DML command a program must execute is a **SINON** to initialize the task with the PDM. Then you can perform whatever database activities your business function requires: access, search, update files, and so on. If you update files, do periodic **COMITs** or **RESETs**. The last command a program must execute (when your business function is complete) is a **SINOF**.

Other PDM DML commands are for file manipulation, according to which files you can access. Concerning this, consider that the PDM executes with a CSIPARM file which names a schema and environment description on the Directory. The schema and environment description grant or deny permission to a program to access or update certain files. Therefore, the schema and environment description affect which files your application can access with DML commands, and which files you can update.

The Database Administrator (DBA) places file descriptions in the Directory and assigns names to each physical field in a record. Your program uses those names in the *data-list* parameter of most DML commands. A data list can name some or all of the fields in a record, in any order. For information about special keywords in the data list, see “**CALL statements and data lists**” on page 59.

On a read, when the PDM returns control to the application (after servicing the command), the *data-area* parameter contains the contents of the data fields named in the data list, in the same order.

The PDM locates each database record by its relative record number (RRN) in its primary or related file. Many DML commands return an RRN (relative record number) as one of their outputs. Often, your program must pass this RRN to a subsequent DML command as an input parameter.

The following sections of this overview explain how to use PDM DML to perform functions that affect the database, such as open, add, delete, read, and write. The overview also discusses error handling and database integrity for your application. Then complete syntax descriptions of each DML command are presented in “**Command syntax**” on page 67. Simple example applications are presented and explained step-by-step in “**Programming examples**” on page 273.

Opening and closing files

For information about opening and closing primary and related files, see “[Opening and closing primary and related files](#)” on page 19. For information about the indirect opens and closes of secondary key files (index files), see “[Opening and closing secondary key \(index\) files](#)” on page 21.

Opening and closing primary and related files

Before any application can process a primary or related file, the file must be available in the user environment description, and must be open.

Open a primary or related file in one of the following ways:

- ◆ The setting of the environment description-to-file relationship for each file requests that the PDM open the file during PDM initialization.
- ◆ The CICS Connector command OPEN opens a defined group of files.
- ◆ A special PDM DML program written by your DBA opens all files with one or more **OPENX** commands.
- ◆ Each PDM DML application issues its own **OPENX** for the files it needs.

With the first method using the Directory open, the file can be opened for read only access or shared update access. With the methods using OPEN or **OPENX**, the file can be opened for read only, shared update, or exclusive update access. An **OPENX** can open one, multiple, or all files in the environment description. However, the environment description actually controls whether the PDM should process or ignore an **OPENX** command from a task or the CICS connector.

When a file is opened for read-only access, applications can issue a primary or related file read command. The PDM returns an error status code to any task attempting to update the records (add, change, delete). Your application can update a file only if it is opened for shared update (SUPD) or exclusive update (EUPD).

SUPD (shared update) is the recommended open mode for multitask PDM operating mode. SUPD mode allows multiple tasks to concurrently update the file. EUPD (exclusive update) mode allows only the opening task to update the file or to close it from EUPD mode. Use EUPD opens with caution, and always close and reopen for IUPD or SUPD in a multitask environment.

To close one or multiple primary or related files, use a CLOSX DML command. You can do this in individual applications or by a special DBA application that closes all files at an appropriate time. CLOSX closes a file if no other tasks are using that file. In a multitask environment, use CLOSX commands cautiously because they can cause problems for tasks that execute later and do not open their files.

CLOSX can close a file partially (PART) or completely (COMP). As with opens, the environment description actually controls whether the PDM should process or ignore a CLOSX command from a task or the CICS connector. At PDM termination, the PDM closes files completely and unlocks them if they are not locked by a failed task.

When task logging is active, you must finalize an OPENX or CLOSX with either a COMMIT or RESET. Do not use both OPENX and CLOSX for a file in the same logical unit of work. If a task changes a file's mode and does not issue a COMMIT or RESET, the PDM prevents any other task from changing that file's mode. The other task waits for the held file lock record (unless the wait creates a possible deadlock situation). If the first task does not issue a COMMIT or RESET, the waiting task receives an error status code.

Opening and closing secondary key (index) files

Index files contain groups of information called secondary keys. Secondary keys provide an alternate method to access primary or related file records via the **READX** (read indexed) command. The RDM uses this method for certain types of views. An index file can contain information about more than one secondary key. These multiple secondary keys for a file can all reside in the same index file or in different index files.

You do not open index files directly. The PDM opens index files when you (or your environment) open the first primary or related file having a secondary key defined and stored in the index file. The open mode for the index file is derived from the open mode of that first file, as follows:

Primary/related file open mode	Index file open mode
READ	READ
IUPD	IUPD
SUPD	SUPD
EUPD	SUPD

For an **OPENX** command, if an index file is already open in a lower mode than your specified mode, it escalates. If already open in a higher mode than your specified mode, it stays the same.

Index files also cannot be closed directly. When you use **COMP** (complete) mode to close the last primary or related file having secondary keys on a particular index file, the index file also closes **COMP**. When you close a primary or related file with **PART** (partial) mode, any associated index files that were locked remain locked. To unlock the index files, you must close all associated primary or related files with **COMP** mode.

When a task or RDM issues a **READX** command, the PDM determines (from the Directory) which index file to read according to which secondary key you specify. The PDM reads a record from that index file, and interprets the contents of that index record to determine which record to read from the base file. The PDM returns the base record to the task.

Adding a primary record

In order for your program to add a primary record, the file named in the *file* parameter must be open in EUPD or SUPD mode. You use the **ADD-M** command to add a record to a primary file. The PDM locates a space (RRN) for the new record by using a hashing algorithm against the value in the *control-key* parameter. If the result produces a synonym record, the PDM holds any affected synonym records and updates their pointers. The new record is constructed using the information you place in the *data-list* and *data-area* parameters. Data items not named in the data list are filled with blanks in the added record. The PDM adds this record to the file, and in addition, performs maintenance to all secondary keys that are populated for this file.

Reading a primary record

When accessing records on a primary file, you can navigate the file with three types of read commands: direct, secondary key serial, or serial.

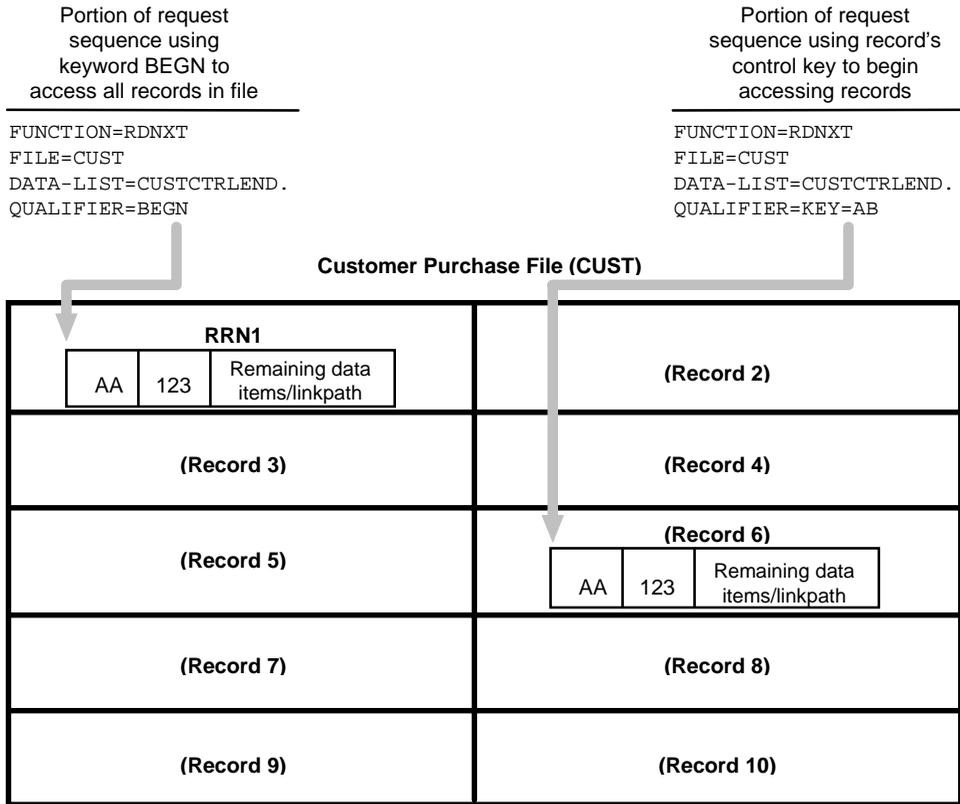
- ◆ A direct read reads one specific record, using a control key value. You can affect sequential processing by repeated reads, increasing or decreasing the specified key. Use the **READM** command for a direct read of a primary file.
- ◆ A secondary key serial read means repeated reads for sets of records associated with each secondary key value. Use a repeated **READX** command for secondary key serial reads. **READX** automatically uses the appropriate index file containing the secondary key. You can limit the retrievals to those records containing a specific secondary key value. You can also limit to only certain secondary key values by using masking information.

The first **READX** must use the keyword **BEGN** in the *qualifier* field. Because all secondary keys are in ascending sequence, records can be retrieved in either a forward or reverse direction on the secondary key values. To alter the direction of the read, specify the opposite direction keyword in *options*, and specify **REBD** in the *qualifier*; this starts reading in the opposite direction. You can start at the beginning or end of the secondary key values, or with a particular value.

- ◆ A serial read means repeated reads for records in the sequence they are physically stored on the file (in RRN order). For **ESDS** and **BDAM** primary files, the returned records are in random logical order since they are not stored physically in alphanumeric order by control key. For **KSDS** primary files, the returned records are in control key order. Use serial reads for large scale file access or when the records to be retrieved are not known uniquely by key. The **FINDX** and **RDNXT** commands are available for serial reads.

The **RDNXT** command serially retrieves all records. The **FINDX** command serially searches for a record which satisfies criteria defined in the *argument* parameter. The *argument* parameter specifies which data items are to be examined, how the test is to be made (a comparison operator), and what values comprise the criteria. For either **FINDX** or **RDNXT**, you can start at the beginning of the primary file and continue through it, or start at a specified record and then continue serially (physically). To start at the physical beginning, use the keyword **BEGN** in the *qualifier* parameter. To start processing with a specific record, use the control key (of the record where you want the reads to begin) in the *qualifier* parameter. Repeat the **FINDX** or **RDNXT** to continue serially to the end of the file. The PDM returns the data of each record and its RRN location.

The following figure illustrates a serial read using **RDNXT**. It shows which record the PDM retrieves from a file (CUST) if you code the **RDNXT** *qualifier* parameter as **BEGN** or if you use **KEY=control key**. If **FINDX** were used, the same record would be retrieved in either case if it met the criteria you specified in the *argument* parameter.



Data returned to your program when the qualifier is set to:

	BEGN	KEY=control-key
Qualifier contents after first RDNXT	1	6
Data area contents after first RDNXT	AA	AB

Serial retrieval of primary records using RDNXT

You may want to add (**ADD-M**) records to or delete (**DEL-M**) records from a primary file while serially processing it with a **RDNXT**, **FINDX**, or **READX** command. Use caution, because your application program might not have serial access to certain records after the PDM performs the delete and add logic.

For example, the current record (retrieved serially) could have a synonym that has not yet been read. If you or another task deletes the current record, the PDM automatically reorganizes the file and could physically move the synonym so it is unavailable for the next or a subsequent serial access.

To avoid this situation, you should issue all **ADD-Ms** and **DEL-Ms** after serial processing is complete. This technique ensures that all records are available for program analysis. You can execute the **WRITM** command while processing a primary file with the **RDNXT** command because **WRITM** does not reorganize synonym chains. **WRITM** simply updates in place.

Updating a primary record

Use the **WRITM** command to update a record in a primary file that is open in **SUPD** or **EUPD** mode. Before you can update a primary record with **WRITM**, you must issue a read command (**READM**) for it. In a multitask operating mode or when task logging is active, you must issue the read with explicit record holding (see “**Record holding**” on page 42). The PDM uses the specified control key to locate the record to be updated. The PDM then moves the data items (as specified in the data list) from the data area to the corresponding sections in the record you want to update.

Deleting a primary record

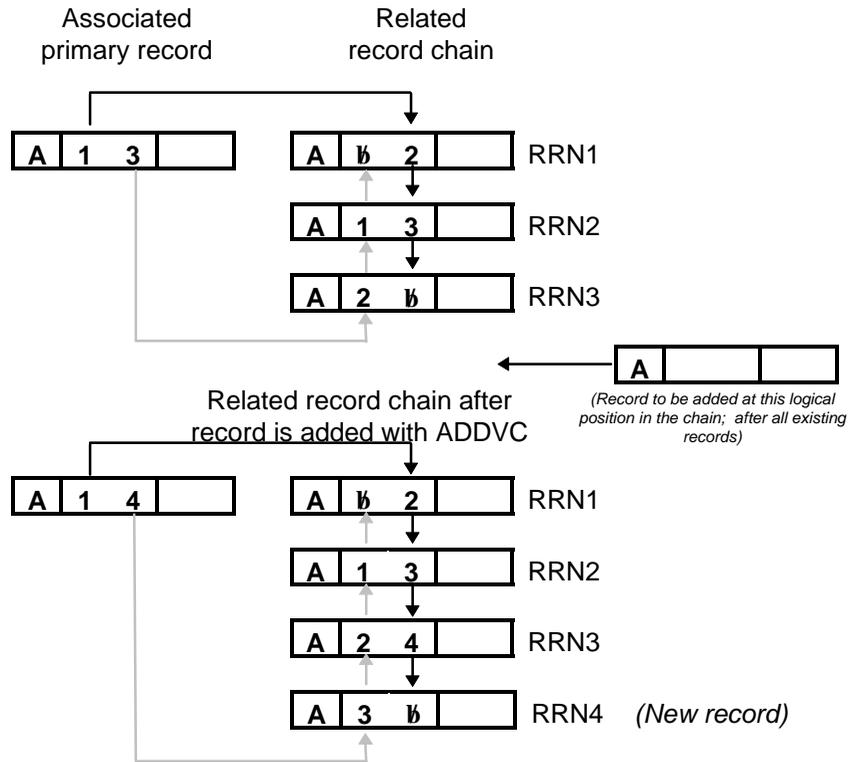
Use the **DEL-M** command to delete a record from a primary file that is open in **SUPD** or **EUPD** mode. The PDM deletes the record whose key is in the *control-key* parameter from the file identified by the *file* parameter. If you are running your program in a multitask environment or if task logging is active, you must read (**READM**) the record with explicit record holding before you can delete it. The PDM automatically holds any affected synonym records and updates their pointers. A primary record cannot be deleted if there are any related file records linked to it.

Adding a related record

Before your program can add a related record, the file named in the *file* parameter must be open in EUPD or SUPD mode. You use an **ADDVC**, **ADDVB**, or **ADDVA** command to add a record to a related file. The new record is constructed using the information you place in the *data-list* and *data-area* parameters. Data items not named in the data list are filled with blanks in the added record. The PDM adds this record to an available RRN (relative record number location) in the file, and performs structural maintenance to the linkpaths for the file. This causes automatic PDM holds on affected primary and related records. In addition, the PDM performs maintenance to all secondary keys that are populated for this file.

You can add new records to a chain of related records at various logical positions within the chain. You can add to the beginning, end, or at any logical position within the chain. When adding records to a related record chain, the PDM stores all records belonging to one chain as close together as possible. The following text and figures explain database navigation when you use an **ADDVC**, **ADDVB**, or **ADDVA** command.

Use the **ADDVC** command (add continue) to add a related record at the logical end of the chain on the controlling linkpath. The PDM also adds the new record to the end of all other linkpaths defined for this record. The following figure shows a primary record connected to a chain of three related records and what happens to the database's logical structure when you issue an **ADDVC** command. Follow the arrows in this figure to see how the PDM navigates through a single chain of related records.

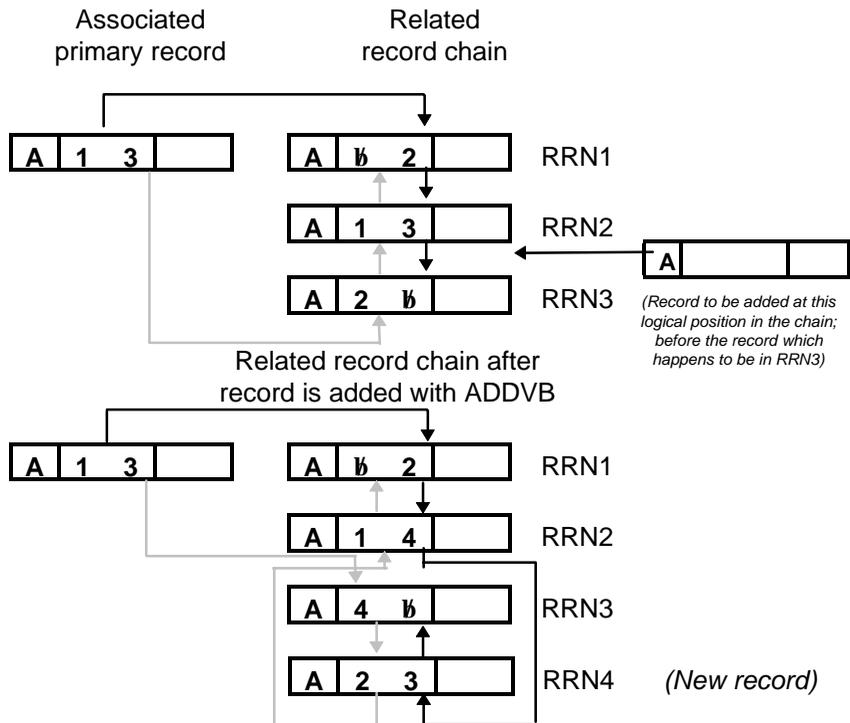


Legend:

- ↑ Chain read in forward direction
- ↑ Chain read in reverse direction

Related record chain before and after using the ADDVC command

Use the **ADDVB** command (add before) to add a related record in front of another related record in a record chain. Before making this addition, you must know the RRN of the record before which you want to place the new record; otherwise the PDM will not know where to place the new record. (Issue a read command to get the RRN of a record.) The PDM adds the new record logically in front of the record whose RRN you specify in the *reference* parameter. The new record is added to the end of all other linkpaths with which this record is associated. The following figure shows what happens to the database's logical structure when you issue an **ADDVB** command. This figure also shows how the PDM navigates (following linkpath pointers) through a single chain of related records.

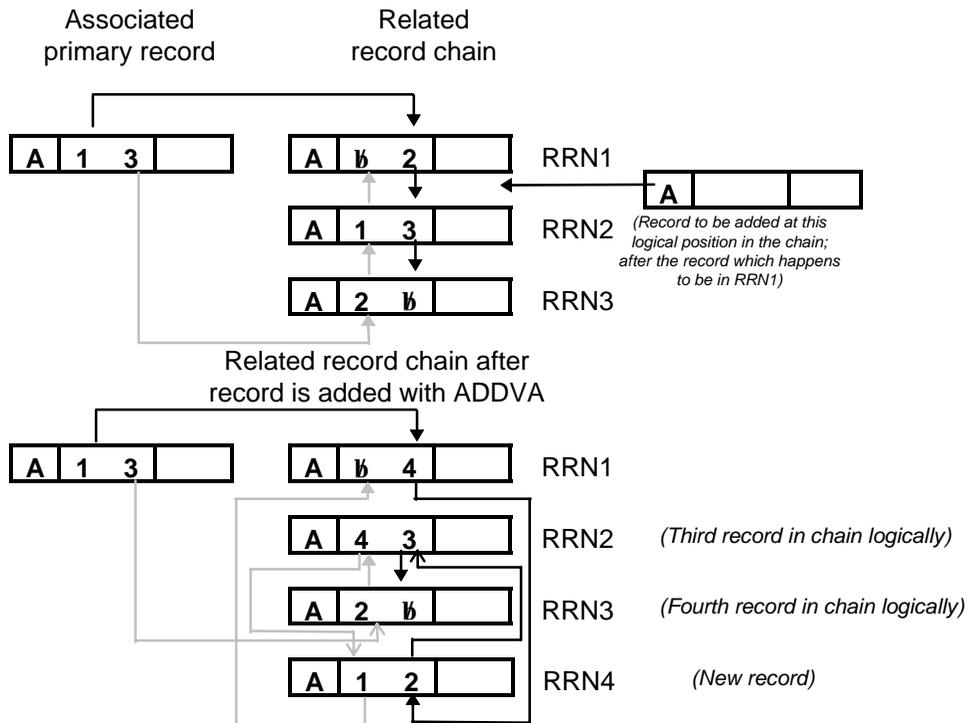


Legend:

- ↑ Chain read in forward direction
- ↑ Chain read in reverse direction

Related record chain before and after using the ADDVB command

Use the **ADDVA** command (add after) to add a related record logically after another related record in a record chain. Before making this addition, you must know the RRN of the record after which you want to place the new record; otherwise the PDM will not know where to place the new record. (Issue a read command to get the RRN of a record.) The PDM adds the new record logically after the record whose RRN you specify in the *reference* parameter. The new record is added to the end of all other associated linkpaths. The following figure shows what happens to the database's logical structure when you issue an ADDVA command. This figure also shows how the PDM navigates through a single chain of related records.



Legend:

- ↑ Chain read in forward direction
- ↑ Chain read in reverse direction

Related record chain before and after using the ADDVA command

Reading a related record

When accessing records on a related file, you can navigate the file with four types of read commands: direct, sequential, secondary key serial, or serial.

- ◆ A direct read reads one specific record. The PDM locates the record to be read by first using the *control-key* parameter to locate the associated primary record. This primary record's linkpath points to the appropriate chain of related records. Then, using the RRN specified in the *reference* parameter, the PDM goes directly to the required related record. Use the **READD** command for a direct read of a related file.
- ◆ A sequential read means repeated reads for records in a logical sequence along a linkpath chain. To perform strictly sequential reads, use a repeated **READV** or **READR** command. Use **READV** for a forward sequential read of a chain, from its logical beginning to its logical end. Use **READR** for a reverse sequential read of a chain, from its logical end to its logical beginning. When the end of a chain is reached, you can reinitialize with another control key on the same linkpath. Thus you can process an entire linkpath (usually the entire file).
- ◆ A secondary key serial read means repeated reads for sets of records associated with each secondary key value. Use a repeated **READX** command for secondary key serial reads. **READX** automatically uses the appropriate index file containing the secondary key. You can limit the retrievals to those records containing a specific secondary key value. You can also limit to only certain secondary key values by using masking information.

The first **READX** must use the keyword **BEGN** in the *qualifier* field. Because all secondary keys are in ascending sequence, the records can be retrieved using either a forward or reverse direction on the secondary key values. To alter the direction of the read, specify the opposite direction keyword in options, and specify **REBD** in the *qualifier*; this starts reading in the opposite direction. You can start at the beginning or end of the secondary key values, or with a particular value.

- ◆ A serial read means repeated reads for records in the sequence they are physically stored on the file (in RRN order). Use a **FINDX** or **RDNXT** command to perform a serial or serial-sequential read.

The **RDNXT** command retrieves all records. The **FINDX** command searches a file for a record which satisfies criteria defined in the *argument* parameter. The *argument* parameter specifies which data items are to be examined, how the test is to be made (a comparison operator), and what values comprise the criteria.

For either **FINDX** or **RDNXT**, you can start at the physical beginning of a file and continue through it by using the keywords **BEGN** and **SERIAL** in the *qualifier* parameter. Repeat the **FINDX** or **RDNXT** to continue to the end of file. The PDM returns the data of each record and its RRN location.

For a related file, **FINDX** and **RDNXT** can also perform a serial-sequential read. To make a **FINDX** or **RDNXT** serial-sequential, use a linkpath in the *qualifier* parameter with either **BEGN** or a control key value as the starting point. The serial part of the read involves finding and returning the first physical head-of-chain record. The sequential part is reading all other records in that chain. Then the next head-of-chain that physically follows the first one is retrieved, and so on to the end of the file.

Updating a related record

In order to update a related record, the file must be open in SUPD or EUPD mode. Before you can update a related record, you must issue a read command to obtain the RRN. In a multitask operating mode or when task logging is active, you must issue the read command with explicit record holding (see “Record holding” on page 42).

When updating a related record, determine if the file is coded or uncoded. If you are using a coded data list and if the file is coded, you must include the record code in the data list and data area. If these two codes do not match or if the code in the record does not match the code in the data area, an error status code is returned. You can use either the **WRITV** or **ADDVR** command to update a related record.

If the file is uncoded, use a **WRITV** command. Also use **WRITV** for a coded file if you are not changing a record code or control key. The PDM uses the RRN in the *reference* parameter to locate the related record. When the PDM processes a **WRITV**, like a **WRITM**, the data items in the data area are moved to the corresponding location in the record you want to update.

Use the **ADDVR** command to update a related record when you want to change a record code or control key; otherwise use **WRITV**. The **ADDVR** command can remove a related record from one chain and link it to the end of another chain.

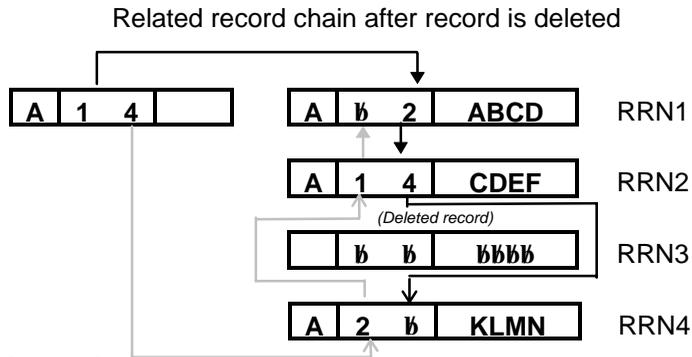
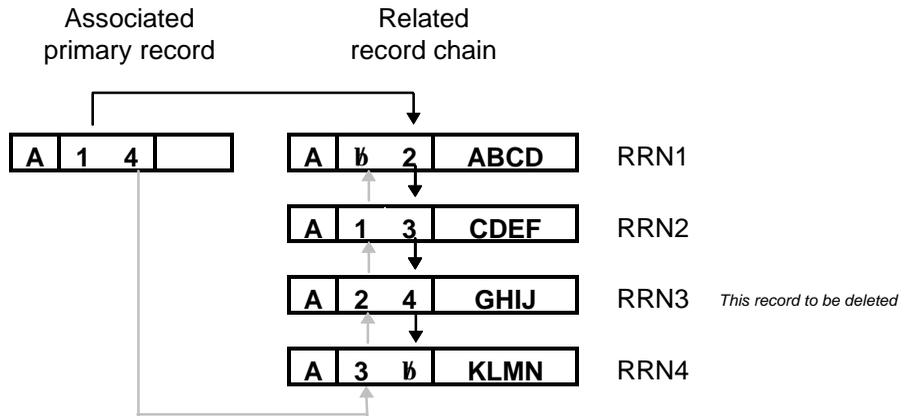


ADDVR is the only command which can be used to change a record code or a control key.

The **ADDVR** command performs like the **WRITV** except when the update requires the addition or deletion of a linkpath in the record. If the record exists in both the old and new linkpaths and the key changes, the PDM logically relinks an existing related record into different chains without physically moving the record. The PDM examines every control key defined for the record to be processed by comparing the new data area to the PDM's I/O area. (The PDM's I/O area contains the existing record which is to be updated.) If a control key changes, the PDM removes the record from the old chain and logically adds the record to the end of the new chain (the new linkpath is determined by the control key specified in the data area). The PDM updates the old linkpath and new linkpaths to reflect the new links. The PDM automatically holds the records affected by linkpath maintenance.

Deleting a related record

Use the **DELVD** command to delete a record from a related file that is open in SUPD or EUPD mode. If you are running your program in a multitask environment or if task logging is active, you must read the record with an explicit hold before you can delete it. The PDM removes the record whose RRN is in the *reference* parameter from the file identified in the *file* parameter. The PDM also removes the record from all associated linkpaths and fills the record with blanks so it is available for immediate reuse (see the following figure). After executing, the PDM updates the *reference* parameter with the RRN of the record immediately preceding the deleted record (in the record chain) identified by the *linkpath* parameter.



Legend:

- ↑ Chain read in forward direction
- ↑ Chain read in reverse direction

Related record chain before and after using the DELVD command

Reading records via secondary keys

Data can be retrieved by using a secondary key serial read. Refer to “[Reading a primary record](#)” on page 23 for information on secondary key serial reads.

If you are issuing a **READX** DML with an end parameter of RLSE and the data list fields you wish to retrieve are completely contained in the secondary key, I/O to the primary or related file becomes unnecessary.

There are additional criteria involved in satisfying this I/O performance feature that are dynamically decided at execution time.

If we have found a coded data element in the secondary key, then in order for this **READX** to qualify:

1. The record code must be part of the secondary key.
2. The record code must be at the same offset within the secondary key.
3. If the record code is not part of the secondary key, then the secondary key must be built on only one record code (excluding the base internal record).

Diagnosing application DML errors

Each PDM DML command returns a status code indicating either the successful completion of the operation or cancellation due to an error. You need to code your application to check the status code after each command and take appropriate action according to various codes that could be returned. If the status is not **** (success), then some sort of warning or error condition is indicated. The specific statuses for which to code differ from one command to another, and are presented with the *status* parameter entry for each DML command in “[Command syntax](#)” on page 67.

When coding your program, include two steps in the logic to help handle errors, warnings, and end-of-processing indicators. First, code logic to handle and correct certain expected situations. An expected status code could be a RSTR on a **SINON** to indicate you are reexecuting your program after a task or system failure. For the DBA, it could be an ACTV on an **ENDTO** to indicate that at least one task is still active.

In some cases, one of several status codes are expected, and the program’s subsequent processing depends on which status code the PDM returns. For example, when executing a **READM**, include logic to handle an MRNF status code (among others). MRNF tells you a record with the specified control key value does not exist on this primary file. When executing the **FINDX** commands on related files, ENDC and END. are expected.

Secondly, your program logic must handle all unexpected status codes. Ignoring these statuses could result in abnormal termination of your program, loss of database integrity (errors in chains, etc.), or failure of the database. Unexpected status codes could be a FNAV (file not available) or an IPAR (invalid parameter) during program testing. They could also be a notification that a command began executing but could not complete. The PDM cancels and backs out any processing caused by that command before it detected the error. For example, if the PDM detects an error in a record chain while processing a command, that command is backed out. In addition, all index changes made prior to discovery of the chain error are backed out. If some processing of the command was performed before the PDM detected the error (new linkages were constructed), the PDM attempts to reconstruct the file to its former configuration.

Usually, the appropriate action for all unexpected status codes is to produce a formatted report of the status and the DML parameters that caused the problem and stop the program. The explanation and user action for all status codes are documented in the *SUPRA Server PDM Messages and Codes Reference Manual (RDM/PDM Support for OS/390 & VSE)*, P26-0126.

You can also use the `TASKEEXT` option of the `SHOWX` command in this report. You issue the `SHOWX` command immediately following the failed DML command. This retrieves additional information describing the failure and is called the task extended status. It begins with a repeat of the status you received (e.g., `IPAR`) followed by a 4-digit identification code and text. These are also documented in the *SUPRA Server PDM Messages and Codes Reference Manual (RDM/PDM Support for OS/390 & VSE)*, P26-0126.



Do not code logic to handle specific 4-digit identification codes from `SHOWX`. These may change with releases of SUPRA Server.

The `SHOWX` task extended status information (`TASKEEXT`) is always 36 characters in length. Your application always places at least 36 in the *length-in* field of the *qualifier* parameter, and provides a *data-area* field of that length.

You need to issue two `SHOWX` commands: one to request the information and one to free the PDM context memory. The following examples show retrieval of task extended status information.

Before issuing the first `SHOWX`, your application initializes the parameters as follows:

```

command      : SHOWX
status       : output field
option-list  : TASK=*,END.
              Request for Task information
qualifier    : BEGN0036oooocccc
              BEGN = Means first SHOWX in a series
              0036 = Length of data-area (binary fullword)
              oooo = Blanks
              cccc = Blanks
data-list    : TASKEEXT,END.
data-area    : output field
endp        : END.
```

After executing the **SHOWX**, the PDM updates your *status*, *qualifier*, and *data-area* fields as follows:

```
command      : SHOWX
status       : ****
option-list  : TASK=*,END.
qualifier    : NEXT00360036cccc
              NEXT = PDM will try for more information if
                  the command is repeated with qualifier as-is
              0036 = Length of data-area (unchanged)
              0036 = Returned length of retrieved data in
                  data-area
              cccc = Returned PDM context pointer (do not
                  modify)
data-list    : TASKEXST,END.
data-area    : statnnnmodulenm-add'l info ... END.
endp        : END.
```

Your program can display or print the diagnostic information in the data area.

You can repeat the **SHOWX** to free the PDM context memory in either of the following ways:

- ◆ Repeat the command with *qualifier* containing NEXT, just as the PDM returned it to the application. The PDM finds no more information, frees the context, and returns END. in the qualifier.
- ◆ Change NEXT to ENDS in the qualifier and repeat the command. That tells the PDM you do not want any more information, so the PDM frees the context memory.

An example of using the second method to free the memory is as follows:

```

command      : SHOWX
status       : content ignored on input
option-list  : TASK=*,END.
qualifier    : ENDS00360036cccc
               ENDS = means no more information is requested
               0036 = contents ignored when input with ENDS
               0036 = contents ignored when input with ENDS
               cccc = PDM context pointer Must remain
                   unchanged from receding SHOWX.
data-list    : content ignored on input with ENDS
data-area    : content ignored on input with ENDS
endp         : END.

```

After this command completes, the *status* parameter contains **** to indicate successful execution. The PDM has released its internal context identified by *cccc* in the qualifier.

You can consult the table of **SHOWX** command keywords at the end of “**SHOWX for monitoring resources**,” beginning on page 224, to find the TASK group and the TASKEXST item in that group. There are also other items available about a task, but they usually have little to do with error diagnosis.

Maintaining database integrity in a task logging environment

When coding an application program, remember that your program is not the only program executing in the PDM. Therefore, you must do all you can to maintain the integrity of the database. In a task logging environment, concepts such as a logical unit of work, record holding, and recovery can help ensure database integrity. These concepts are discussed in the following sections.

Logical unit of work

SUPRA Server is designed to process logical units of work (transactions). A logical unit of work is a group of database requests from one caller that must all be completed together or not at all. A task may consist of one or several logical units of work. For example, a logical unit of work could be entering a purchase order to the system, posting an invoice, or adding a new customer. If an error occurs, the logical unit of work is incomplete. In that case, you must undo all the processing before the error.

In a task logging environment, you define the length of a logical unit of work by issuing **COMIT** commands. That is, the size of a logical unit of work is from commit point to commit point. The length of a logical unit of work determines how far back (in processing) you need to go in order to continue after an error. If task logging is not active, you might have to go back to the beginning of your program. If active, you go back just a few steps to the most recent COMIT. In a task logging environment, all updates and relevant information are written to a Task Log File on a task and logical unit of work basis. For more information on task logging and the Task Log File, refer to the *SUPRA Server PDM and Directory Administration Guide (OS/390 & VSE)*, P26-2250.

The following example illustrates a sample program structure which could be used to implement a logical unit of work using PDM DML:

```
INITIALIZATION (SINON)
    While not finished, do:
        screen input and validation
        Data Base Update Processing
        If error, then RESET
        else COMMIT
    end
TERMINATION (SINOF)
```

The initialization (**SINON**) process identifies the new task to the PDM and allocates a unique internal task identifier. Once you have signed on, you can begin processing.

You can process the database using either RDML or PDM DML. Do not use both languages in the same program. Your program can perform read-only database calls or can update database records. If your program updates database records, the records are held until committed, and cannot be accessed by other programs during that time.

If an error is detected while updating, you can remove the updates to the records by issuing a **RESET**. If no errors are detected, issue a **COMIT** to free any held records. Updates to the records are now permanent; that is, they can no longer be removed by issuing a **RESET**. Errors in a logical unit of work are handled differently based on where the error was detected:

- ◆ **Detected by the program.** Because the logical unit of work is incomplete when the error is detected, all the processing done before the error must be undone by issuing a **RESET**. Your program can either start a new logical unit of work or sign off, depending on the severity of the error.
- ◆ **Not detected by the program.** If you code your program to execute **COMIT**s at appropriate times, the PDM can assure the integrity of your database. The PDM ensures that each logical unit of work is always complete. If the PDM finds that a program has terminated normally (batch) or abnormally (batch and online) before signing off, the PDM resets to the most recent commit point. If the most recent commit point happens to be a **SINON**, then the PDM signs off the task; otherwise, the task remains signed-on, able to be restarted. This ability means that the PDM will return an RSTR status to the task when it issues a **SINON** the next time.

Record holding

Record holding is a facility within the PDM which protects records from being updated by two tasks at the same time. A task must reserve (hold) records before it can update them. Because only one task at a time can hold a record, record holding controls contention among tasks accessing the same record. The DML commands that can hold records while they execute are: **ADD-M**, **ADDVA**, **ADDVB**, **ADDVC**, **ADDVR**, **DEL-M**, **DELVD**, **FINDX**, **RDNXT**, **READD**, **READM**, **READR**, **READV** and **READX**. Records may be held differently depending on the processes being performed.

Because more than one task may be accessing the same file in a multitask environment, concurrent updating of a record by two or more tasks could occur. This could destroy database integrity. To prevent this, the multitask PDM uses record holding. The PDM also uses record holding in a single-task environment when task logging is active.

The PDM holds the records until the record is no longer required (for example, function completion, **COMIT** or **RESET** processing, record stealing). The DML commands perform two types of record holding (numbers 1 and 2 following); the PDM does one type of record holding (number 3 following):

1. **Automatic.** The PDM automatically holds records to prevent interference between DML commands. The PDM temporarily holds additional records that might be needed for completing a DML function (for example, a primary record on **ADDVC**, synonyms on **DEL-M**). The following commands cause automatic record holding: **ADD-M**, **ADDVA**, **ADDVB**, **ADDVC**, **ADDVR**, **DEL-M** and **DELVD**.

The PDM holds internally all the records affected by the add or delete (synonym chain of primary records), not only the record to be added or deleted. The linkpaths in the affected records may be changed as a result of the add or delete. The system determines which records could be affected and holds all the affected records for the duration of the DML command. These holds are in addition to the explicit hold (see #2) that you must obtain in a multitask or task logging environment. Automatic record holding is in operation in both task logging and non-task logging multitask environments:

- If task logging is active, and the held record(s) was updated, the automatically held record(s) becomes an uncommitted held record(s) (see #3) after the DML function completes. Otherwise, it is released for use by other tasks when the DML completes.
- If task logging is not active, automatically held records are released for use by other tasks when the DML completes.

2. **Explicit.** An explicit hold is required if you want to update or delete a record, or if you want to prevent someone else from doing so. Explicit record holding is accomplished with read commands having END. in the last parameter instead of RLSE. With END., any record you read successfully is not held or uncommitted by any other task; you now hold it. With RLSE, any record you read does not have an explicit hold on it, but could be an uncommitted record (see #3). You do not have a hold on it.

The read commands with which you can use END. are: **FINDX**, **RDNXT**, **READD**, **READM**, **READR**, **READV**, and **READX**. You can explicitly hold only one record per file per task. Your next DML command to that file can then update this record.

You can explicitly hold records in both task logging and non-task logging environments. The only time the PDM does not actually perform explicit record holding for END. is in a multitask environment without task logging when a file is open for read only access.

If task logging is not active, a task requesting a record already explicitly held by another task is not placed into a wait state, but immediately receives a HELD status. If the requesting task is an online task (CICS without task logging), a HELD status is returned to 999 requests. The task requesting access to that record on the 1,000th time is given ownership of that record (record stealing).

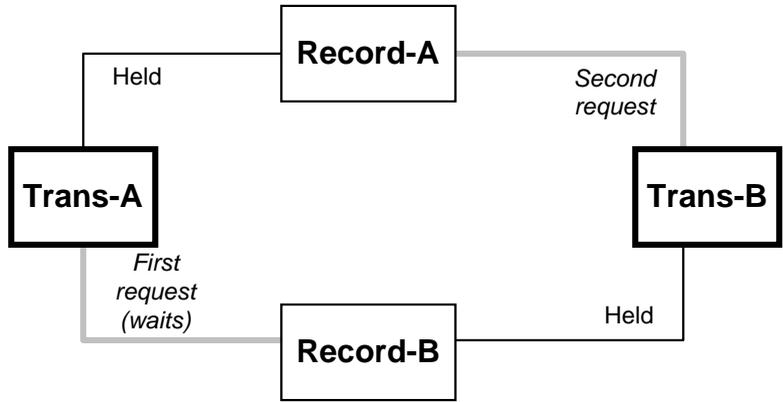
The PDM releases an explicit hold on a record if you:

- Issue a WRITE command for the record (**WRITM**, **WRITV**, **ADDVR**). Note that this also applies if the update command receives an error status (IMDL on a **DEL-M** command). Previous releases of SUPRA did not always release the explicitly held record after an error status. With SUPRA 2.4 and higher, a task will receive a NHLN status if it issues the update command (**DEL-M**) for the same record as it received the error status for, and it has not issued an intervening read with END. If task logging is active, and the task updates the record, the explicitly held record becomes an uncommitted record.
- Issue another read command to the same file.
- Issue a **FREEX** command for the file (allowed only in a non-task logging environment).
- Lose the record to another task successfully “stealing” it when task logging is not active.
- Issue a **COMIT** or **RESET** for the logical unit of work (this also releases uncommitted records (see #3) for update holds).

3. **Uncommitted.** If task logging is active, records previously held, either explicitly or automatically, become uncommitted records after an update to the record completes successfully. Uncommitted record holding by the PDM prevents interference between transactions that could jeopardize database integrity. Once a record has been updated by a task, it and any associated held records cannot be read with END. or updated by any other task, until the updating task has committed or reset. However, a read with RLSE is allowed for this uncommitted record.

Uncommitted record holding is not performed by specific DML commands. The PDM provides it to allow updates to be backed out or committed. Therefore, these records remain uncommitted (and unavailable for a read with END.) until you issue your next **COMIT** or **RESET** command. At that time, the records are released. The first task waiting for a hold on each record is given ownership of the specific record for which it is waiting.

In some cases, two logical units of work may request an automatic or explicit hold for the same database records simultaneously. In the following figure, Trans-A holds Record-A and starts processing. Meanwhile, Trans-B holds Record-B and starts processing. At some point in time, Trans-A tries to hold Record-B. Since it is already held, Trans-A waits for Record-B to be released. During Trans-B's processing, it tries to hold Record-A, which is held by Trans-A. Therefore, if Trans-B waits, neither Trans-A nor Trans-B will ever complete since they are waiting for each other.



This situation is called “deadlock,” “deadly embrace” or “fatal embrace.” The PDM prevents it by returning error status EMBR to TRANS-B (second request), indicating that a record could not be reserved for TRANS-B without creating the deadlock situation.

TRANS-B should issue a **RESET** and then retry its processing from the most recent commit point. This RESET command resolves the deadlock by releasing all held resources for the logical unit of work. Contention between logical units of work can cause poor performance.



Therefore, we recommend that you design your logical units of work so that different logical units of work in different tasks do not require the same database records

Program recovery

A failure can occur requiring you to recover your application program. These failures can be caused by:

- ◆ Operator cancels
- ◆ Application program failures
- ◆ Hardware failure
- ◆ Power failure

If the PDM terminates due to one of these failures when task logging is active, your DBA should warm start the PDM (using the same Task Log File) with the same CSIPARM file that was in use at the time of the failure. This provides the same PDM name and same user schema and environment description. When the warm start is complete, the PDM opens each file to its most recent committed open mode, and resets each task to its most recent commit point before the failure. This allows for all programs to continue from the failure point.

In a task logging environment, if your program fails in the middle of a logical unit of work, you can restart your program at the most recent commit point if you have been marking recovery points (commit points) with the **COMIT** command and if your **SINON** access was **RESTART=NORMAL**. If your program fails and you have not been issuing **COMIT** commands, you must start processing from the beginning.

In a task logging environment with **RESTART=NORMAL**, check the *status* parameter of the **SINON** command for **RSTR** when you restart your program. If your program has not issued a **COMIT** command, the **SINON** command returns a ******** status. If it has issued a **COMIT**, the PDM returns the **RSTR** status. This status code indicates that the database has been restarted from the Task Log File. You can begin processing from the most recent commit point taken before the failure. You can retrieve data you saved at the most recent **COMIT** by issuing a **RESET**. (The PDM identifies the commit point for you.) This data can include working storage for the program and information required to reposition input files. See “**Programming examples**” on page 273 for a coding example.

Commit points are especially useful for long jobs that update the database. If task logging is active and you have a long-running task that does not issue **COMIT**s, many other tasks may have to wait until your task signs off because records required by those other tasks are held by your task. Your DBA can specify how long a task waits for held records in the user environment description.

Using system control DML commands

The database administration personnel can use some of the DML commands to help control operations of the Physical Data Manager (PDM). These selected commands enable the DBA to write specialty programs or exit routines that perform 5 main functions:

1. Increase the PDM's efficiency and prevent locked files by writing one special application (or an exit) to open the database files for use by all other application programs. Another special application can close them at the end of the processing day. Applications can, of course, issue their own opens and closes but it is not recommended. Files can also be opened by using options of the environment description relationship on the Directory or the CICS Connector. See "Opening and closing files" on page 19 and "Opening and closing files with PDM DML" on page 48.
2. Aid in System Log control: volume switching, quieting the database for log synchronization, and writing your own information to the log. See "Using system logging commands" on page 49.
3. Monitor PDM performance and use of resources by periodically writing statistics to the Statistics file with **RSTAT**. See "Generating statistics" on page 51.
4. Monitor the **RSTAT** statistics plus Directory and PDM information by retrieving them with **SHOWX** into a special program for display or print. See "Monitoring resources" on page 53.
5. Terminate a central PDM not attached under a TP monitor with **ENDTO** (using this command is the same as using the PDM Termination utility). See "Terminating the PDM with PDM DML" on page 58.

All special programs you write for the PDM must contain the **SINON** and **SINOF** commands. The programs should contain a **COMIT** or **RESET** command to end each logical unit of work. This preserves database integrity and prevents locked records and files.

Opening and closing files with PDM DML

In a multitask operating mode, you can use the **OPENX** and **CLOSX** commands to open and close most files when you initialize and terminate the PDM. If you write a special application to open the files at the beginning of the day and close them at the end of the day, each individual application does not need to open and close them separately. However, in a single-task operating mode, individual applications can issue their opens and closes.

With the **OPENX** command, you can open your primary and related files, either one at a time or as a group. The PDM opens the files you specify in the *realm* parameter of the **OPENX**. **OPENX** cannot open index files, Directory files, log files, or the Statistics File, which are all handled by the PDM.

You also use the *realm* parameter to specify whether you want the files opened for shared update, exclusive update, or no update, that is, read-only:

- ◆ If you specify shared update (SUPD), all applications can update the file, not just the one that opened it.
- ◆ If you specify intent to update (IUPD) or READ, no applications can update the file; they can only read it. Specifying IUPD allows any task to reopen with an update mode later; READ does not.
- ◆ If you specify exclusive update (EUPD), only the application that opens the file can update it. However, other applications can read it. Therefore EUPD, if used by this special task, is in effect the same as opening for READ. This special task normally uses SUPD for most files, and READ for any static files.

With the **CLOSX** command, you can unlock and close primary or related files, either one at a time or as a group. The PDM closes the files you specify in the *realm* parameter.

See “**Opening and closing files**” on page 19 for more information about opening and closing files.

Using system logging commands

System logging, like task logging, is a method of tracking database updates.



While system logging is optional, we recommend you use it to help ensure database integrity.

If you use system logging, you can recover the entire system in the following situations:

- ◆ If you are not using the Task Log File or if it is unreadable
- ◆ If an updated database file is unreadable and must be rebuilt

Use the following commands in special applications to control some system logging activities: **ENDLG**, **MARKL**, **QUIET**, **QMARK**, and **WRITD**. However, you cannot use **QUIET**, **QMARK**, and **WRITD** if you are also using task logging.

- ◆ The **ENDLG** command terminates and initializes the logical System Log File(s). If you are not using task logging, **ENDLG** quiesces the database like the **QUIET** command. That is, the PDM waits for all commands to complete and keeps new commands from starting. Then it writes all updated buffers to the database. When it has written all the buffers, it resumes executing other commands. If you are using task logging, the PDM does not quiesce the database. It only writes the database buffers.
- ◆ The **MARKL** command writes any information you want to the System Log File(s). For example, if you have quiesced the system with the **QUIET** command (only in a non-task logging environment), you could use **MARKL** to write your own recovery point information. You can use this command even if task logging is active.
- ◆ The **QUIET** command synchronizes the system. **QUIET** quiesces the entire system, flushes the I/O and log buffers (physically writes all pending updates to the database), writes a **QUIET** record to the System Log File(s), and frees all held resources. At this point, the System Log File(s) and the database files are synchronized to a common point for recovery. You cannot use **QUIET** with task logging.
- ◆ The **QMARK** command is a combination of **QUIET** and **MARKL** commands. **QMARK** produces only one record on the System Log File(s) rather than the two that would be produced if you issued **QUIET** and **MARKL**. You cannot use **QMARK** with task logging.
- ◆ The **WRITD** command writes an entire logical record into a file that your program opened for exclusive update. **WRITD** is primarily for writing recovery programs. When you write the recovery program, you use the System Log File as input. From it, you obtain the address and the data area parameters. Be careful using **WRITD** because the PDM checks only to see if the file you are writing to exists. You cannot use **WRITD** with task logging.

For more information on system logging, refer to the *SUPRA Server PDM and Directory Administration Guide (OS/390 & VSE)*, P26-2250, and *SUPRA Server PDM Logging and Recovery (OS/390 & VSE)*, P26-2223.

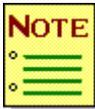
Generating statistics

The PDM generates statistics when the Statistics File is defined in the Directory and when statistics gathering has been turned on. These statistics describe some internal activities of the PDM.

The Statistics File opens when the PDM initializes and closes when the PDM terminates. The PDM automatically writes initial and final statistics to the Statistics File at initialization and termination if the file is not a dummy file.

You may also write the currently accumulated statistics to the Statistics File at any time during processing. You can clear the accumulators at any time. When you want statistics written to the Statistics File, you issue an **RSTAT** command from a special program you write for this purpose. You usually issue the command while running a lone application containing a series of reads or writes that you want to track, or at heavy database usage times.

You can use statistics during production for tuning to monitor performance and to use resources efficiently. Statistics are helpful for establishing optimum blocking factors, the maximum held records value, optimum buffering, and task timing.



As statistics features are enhanced, the **RSTAT** command may change in format or syntax. Therefore, use this command only in special routines that you can modify easily.

To report the Statistics File information, use the Execution Statistics utility. For information on this utility, refer to the *SUPRA Server PDM DBA Utilities User's Guide (OS/390 & VSE)*, P26-6260. To relate the statistics gathered with **RSTAT** to those printed on the Execution Statistics utility report, use the statistics identifiers listed in the tables at the end of "**RSTAT**" on page 203.

Instead of writing to the file with **RSTAT**, you can see accumulated statistics online in two ways:

- ◆ By writing a special application with **SHOWX** commands (see "**Monitoring resources**" on page 53).
- ◆ By selecting Interactive Services, which uses **SHOWX** commands.

With **RSTAT**, you can obtain accumulated statistics on the following internal PDM activities:

- ◆ **Task sign-ons and sign-offs.** The total number of tasks that signed on and the highest number of concurrent tasks signed-on since the accumulators were last initialized. (The task identifiers S2.01 and S2.02 in the tables at the end of “**RSTAT**” on page 203.)
- ◆ **I/O buffer allocations.** The physical and logical reads and writes performed during request processing for each file in the PDM. This measures the absolute I/O activity, as well as the efficiency of your buffer assignments and the buffer allocations algorithm in the PDM. (The statistics identifiers F1.01 to F1.05 and F3.01 to F3.03 in the tables at the end of “**RSTAT**” on page 203.)
- ◆ **Record contention.** The total number of held records (whether automatically held, explicitly reserved, or updated), the highest number of held records at any time, the total number of times a record desired by a task was held by other tasks, and the total number of times a held status is returned for the same record. (The statistics identifiers S3.01 to S3.04 in the tables at the end of “**RSTAT**” on page 203.)
- ◆ **I/O and waits for logging.** The total number of records written to the System Log File and Task Log File, the total number of times the PDM waited for log I/O to complete before writing the next log record, and the total number of times the PDM waited for log I/O to complete before writing a data record. The Execution Statistics utility calculates these statistics from separate statistics for each log file and data file. (The statistics identifiers F1.01 to F1.05 and F3.01 to F3.03 in the tables at the end of “**RSTAT**” on page 203.)
- ◆ **Function or command activity.** The total number of each type of command processed, the level of overlap or concurrency of the command processing by the PDM, the elapsed processing time for all commands, and the average processing time for a command. (The statistics identifiers S4.01 to S4.04, S6.01, S7.01, and S7.02 in the tables at the end of “**RSTAT**” on page 203.)
- ◆ **Interface activity.** The total number of commands issued to the PDM, the highest number of commands running concurrently within the PDM, the total elapsed time on all commands issued to the PDM and the maximum elapsed time to process a single command for all applications. (The statistics identifiers S5.01 to S5.07 in the tables at the end of “**RSTAT**” on page 203.)

Monitoring resources

For monitoring (to obtain **RSTAT** statistics, Directory information, and internal PDM information), you can use the PDM Interactive Services or code **SHOWX** in special application programs. You code the **SHOWX** command with different options depending on the type of information you want.

The amount of information that **SHOWX** can retrieve will vary depending on your *option-list* parameter. Therefore, in a program, it is often not practical to allocate enough data area to receive all the information with one retrieval. You can provide a smaller data area, and reissue the same **SHOWX** several times, letting the PDM control retrievals through the *qualifier*. This is somewhat similar to **FINDX** or **RDNXT** control. However, the data area must be large enough for one full set as defined in the data list, or a complete multiple.

Each time the PDM returns information, the PDM indicates in the *qualifier* whether it has returned all or a part of the data. If you want the remainder, you repeat the **SHOWX** until the PDM indicates it has returned all of it and has released the context memory associated with retrieval. If you do not want it, you reset the *qualifier* to tell the PDM that you are finished so it can free the memory.

To code a **SHOWX**, you provide all the parameters in the order shown in the following example (complete syntax and options are in “**SHOWX**” on page 221). You initialize the following input parameters: *command*, *option-list*, *qualifier*, and *data-list*. The example shows the option list and data list you could use to retrieve some PDM information, including some DML command statistics:

```

command      : SHOWX
status       : output field
option-list  : DBM=*,END.
qualifier    : BEGN0064o0o0cccc
                  BEGN = First SHOWX in a series
                  0064 = data area length of 64 bytes
                      (binary fullword)
                  0000 = blanks
                  cccc = blanks
data-list    : DBMXNAME,DBMXTYPE,DBMXRSTZ,DBMXDMLR,DBMXDMLW,
                  DBMXDMLA,DBMXTHRC,END.
data-area    : character and binary output fields mapped by
                  data-list
endp        : END.

```

Option-list and data-list. The DBM in the example *option-list* parameter above means DBM-group. The DBMX... items in the *data-list* parameter mean DBM-group items. You request **SHOWX** information by using option list and data list keywords like these. You define a group with the option list (see the table of options listings under “**SHOWX for monitoring resources**,” beginning on page 224). You choose items for the data list from that group (see the table of SHOWX command keywords at the end of “**SHOWX for monitoring resources**,” beginning on page 224). In each SHOWX command, you can code data list items from only one group; the group defined by the option list.

The * in the DBM=* option signifies the executing PDM. The DBMXNAME item requests name, and DBMXTYPE requests whether single or multitask. DBMXRSTZ requests the time that accumulators were last cleared; DBMXDMLA requests the number of ADD and DEL commands; and so on.

Qualifier. The qualifier has four fields as shown. The BEGN in the first field of *qualifier* informs the PDM that this is the first iteration of this **SHOWX**. In the second field, you place the binary length of the data area you are assigning for output. You blank the third field, which will receive the actual length of returned output data. You also blank the fourth field, which will receive the PDM's pointer to a context area it creates for your SHOWX.

Data-area. The data list maps the data area. The program's definition of the data area fields must match in lengths, formats, and characteristics to the data items that are requested. Consult the table of SHOWX command keywords at the end of “**SHOWX for monitoring resources**,” beginning on page 224 for these specifications and to calculate the value you need in the *length-in* field of *qualifier*.

In this example, the data area provided is 64 bytes. Other **SHOWX** requests might need more or less. The minimum area needed for this example would be as follows:

DBMXNAME	8 bytes (according
DBMXTYPE	2 to table
DBMXRSTZ	8 of SHOWX
other 4 items, 4 bytes each	<u>16</u> commands)
	34
plus END.	<u>4</u>
	minimum 38 bytes length

In the next example, using the same data area length of 64, you do not receive all the data with one iteration of the **SHOWX**. This example illustrates the information you would provide to retrieve the logical and physical I/O statistics for each file in the active environment description.

```

command      : SHOWX
status       :
option-list  : FILE=ALL,END.
qualifier    : BEGN0064oooocccc
               BEGN = First SHOWX in a series
               0064 = data area length of 64 bytes (binary
                   fullword)
               oooo = blanks
               cccc = blanks
data-list    : FILENAME,FILEBPOL,FILELRED,FILELWRT,FILEIHIT,
               FILEMLTW,FILEPRED,FILEPWRT,FILEFWRT,END.
data-area    : character and binary output fields mapped by
               data-list
endp         : END.

```

Since the option list defines the FILE group, the data list can contain only those names that begin with FILE. There are nine data list items, each requiring 4 bytes, plus 4 bytes for END., so 40 bytes will be returned. Since the data area is 64 bytes, only one file's data can fit. In this situation, the PDM returns the statistics for only one file with each repeat of the **SHOWX**.

When the first iteration completes successfully, the PDM updates your *status*, *qualifier*, and *data-area* fields:

```

command      : SHOWX
status       : ****
option-list  : FILE=ALL,END.
qualifier    : NEXT00640040cccc
               NEXT = PDM will try for more information if
                   the command is repeated with entire
                   qualifier as-is
               0064 = data area length of 64 bytes
               0040 = returned output data length of 40
                   bytes
               cccc = returned PDM context identifier (do
                   not modify)
data-list    : FILENAME,FILEBPOL,FILELRED,FILELWRT,FILEIHIT,
               FILEMLTW,FILEPRED,FILEPWRT,FILEFWRT,END.
data-area    : |CCDF|CDDF|000F|0002|000C|000C|000F|0009|0006|CDC4|
               6931|2701|0014|001C|0008|0008|000A|0006|0004|554B|
               |FIL1|BP01|:::|:::|:::|:::|:::|:::|:::|END.
endp         : END.

```

The returned data area information from this second example is interpreted as follows:

```
For file FIL1, using BP01, the accumulated statistics since the
most recent clearing are:
```

```
500 logical reads
300 logical writes
200 reads which found the block already in memory
200 writes to an already updated block
250 physical reads
150 physical writes
100 forced writes in order to read
```

In this example, the PDM returned data for only the first file. The data area does not have enough room for more than one entry.

You can keep repeating the **SHOWX** without modifying any parameters, but checking the qualifier after each execution. Each time the PDM returns data to you, it updates the first field of the qualifier to NEXT. When the PDM has returned all your data, the PDM changes the qualifier's first field to END. That indicates the data area contains no more data, and that the PDM has released the context memory. Then you do not need to issue any more SHOWX commands.

If at any iteration you do not want the data for the rest of the files, you can change the first field of your qualifier to ENDS, leaving cccc undisturbed, and repeat the **SHOWX** one more time. This releases the context memory.

Never change the last field of the qualifier, the context identifier, except to begin a new **SHOWX** after freeing the context of a previous SHOWX.

Terminating the PDM with PDM DML

Termination of the PDM can be controlled or uncontrolled. Uncontrolled terminations result from an abend in the system, a hardware failure, and so on. Controlled terminations occur when you terminate the PDM intentionally.

If the PDM is attached under CICS, you terminate it with the DISCONNECT operator command. For more information, refer to the *SUPRA Server PDM CICS Connector Systems Programming Guide (OS/390 & VSE)*, P26-7452.



If the PDM is attached under CICS, the PDM terminates if the TP monitor terminates.

If the PDM is operating in central mode, you can write a special program to issue the **ENDTO** command or execute the PDM Termination utility. They both perform the same activities and offer the same options. For information on the utility, refer to the *SUPRA Server PDM DBA Utilities User's Guide (OS/390 & VSE)*, P26-6260.

A controlled termination can be forced or unforced, meaning how to handle still-active tasks. It depends on how you code the FORCE parameter of the **ENDTO** command or the PDM Termination utility.

2

CALL statements and data lists

Access to SUPRA Server databases, when using PDML (Physical Data Manipulation Language) instead of RDML, is through an interface named DATBAS. You link this interface module (DATBASC for CICS programs) with your compiled application program. You make requests to the database by coding CALL statements to DATBAS.

Data list processing is the transfer of data between the database record and the program-supplied data area, using the data list parameter as a “map” to the data area.

You will find information about these two subjects in this chapter.

CALL statements

CALL statements consist of DML commands and parameters. These parameters name variables defined in your program. The parameters are positional; that is, they must appear in the order indicated in each command description. The following examples show the format of a CALL statement in various languages:

◆ In COBOL:

```
CALL 'DATABAS' USING 'READM', STATUS, 'CUST', CONTROL-KEY,  
DATA-LIST, DATA-AREA, 'END.'
```

or

```
MOVE 'CUST' TO FILE  
MOVE 'READM' TO FUNCTION  
MOVE 'END.' TO ENDP  
CALL 'DATABAS' USING FUNCTION, STATUS, FILE, CONTROL-KEY,  
DATA-LIST, DATA-AREA, ENDP.
```

◆ In FORTRAN:

```
CALL DATABAS (FUNCTION, STATUS, FILE, CONTROL-KEY, DATA-LIST,  
DATA-AREA, ENDP)
```

◆ In BASIC:

```
CALL DATABAS (FUNCTION, STATUS, FILE, CONTROL-KEY, DATA-LIST,  
DATA-AREA, ENDP)
```

◆ In PASCAL (ensure that the procedure definition is declared as FORTRAN):

```
DATABAS (FUNCTION, STATUS, FILE, CONTROL-KEY, DATA-LIST,  
DATA-AREA, ENDP)
```

In the **READM** example, this is what the parameters mean:

DATBAS	The name of the interface between your program and the PDM. You link the DATBAS interface with each program.
READM	The 5-character name of the DML command you want to execute. You could also code the name of a field into which you move the DML command name before calling DATBAS.
STATUS	The name of a 4-character area in your program into which the PDM returns a completion status code. This code indicates the success of the operation or the cause of its failure.
FILE	The name of a 4-character field into which the program moves the required file name before calling DATBAS. You could also code the 4-character name of the file you want to process.
CONTROL-KEY	The name of a variable-length area in your program that contains the control key value of the primary record you want to read, or the controlling primary record of a related record you want to read.
DATA-LIST	The name of a variable-length area in your program that contains the names of the data items requested from each record. The data list serves as a map of the data area. You must always delimit a data list with the keyword END.
DATA-AREA	The name of a variable-length area in your program which will contain the data items named in the data list.
ENDP	The 4-character delimiter terminating the parameter list, whose value is END (or RLSE for some commands) to indicate the end of the parameter list.

Data list parameter keywords

The data list tells the PDM what data items to place in the data area (for reads) or what is already in the data area (for writes and adds). The data list and data area must match, or results are unpredictable.

Normally you code the data list as the Directory physical field names of the data items you want to process. However, you can use some keywords in your data list, or as your entire data list, to perform special functions. A data list must always end with "END." These keywords are: ****NONE****, ****DATA****, ****REST****, ****BIND****, ***FILL=nn**, ****CODE****, ***CODE=xx**, and ***COMMON***. The ***COMMON*** keyword retains compatibility for existing TOTAL or TIS applications.

****NONE****

Passes no data items from the record to the data area. Use ****NONE**** if you want to read and perform a count of records but receive no data. You must provide a data area parameter although it is not used. You can use this keyword for either primary or related files. To use ****NONE****, code your data list as follows:

```
**NONE**, END.
```

or

```
**NONE**END.
```

When processing add or write commands, if ****NONE**** appears ahead of the required data list, it is ignored.

****DATA****

Requests the PDM to use all data items in the record. For a read command, moves the entire data record to the data area. For update commands, replaces the entire data record (except the root and linkpaths) with the contents of the data area. For add commands, constructs the entire data record from the contents of the data area. (Note that before an add begins, the primary ROOT and primary and related linkpaths are automatically initialized.) You can use this keyword for either primary or related files. To use ****DATA****, code your data list as follows:

```
**DATA**, END.
```

or

```
**DATA**END.
```

****REST**** This is a special keyword mainly used for user-written recovery programs. For a read command, moves the entire data record to the data area. For update commands, replaces the entire data record, including the linkpaths and the root, with the contents of the data area. For add commands, constructs the entire data record, including the linkpaths and the root, from the contents of the data area. You can use this for either primary or related files. To use ****REST****, code your data list as follows:

```
**REST** ,END.
```

or

```
**REST**END.
```

Use such a list with care, as it can jeopardize database integrity.

****BIND**** Resolves the data list by constructing information in memory to eliminate a table lookup each time the data list is used. SUPRA Server automatically binds all data lists by using the entire data list as the key into the internal data list table. If used, ****BIND**** must be the first word in the data list. Use or omit commas consistently between items in the list as usual:

```
**BIND** ,dataitem1,dataitem2,...dataitemn,END.
```

or

```
**BIND**dataitem1dataitem2,...dataitemnEND.
```

On the first execution of the command with this list, the PDM changes ****BIND**** to ***BNDxxxx**, where **xxxx** is an internal binary identifier used as a key into the internal data list table.

You can use this keyword for either primary or related files. You can also use any of the following keywords along with ****BIND****.

***FILL=nn** Skips the specified number of bytes in the data area during data transfer. Multiple usage in the same data list is allowed. You can use this keyword for either primary or related files. You can use values of 00-99 for *nn*.

You initialize the data area to spaces or any values you want before using a data list with ***FILL=nn** keyword(s). You might use ***FILL=nn** to preformat a data area so that you can print directly from it after retrieving a record. For example, to generate a line of print that looks like:

```
CODE=02 KEY=00001 KEY=00002
```

you would first initialize the data area to:

```
CODE=      KEY=      KEY=
```

You would then code the data list as follows:

```
*FILL=05,rrrrCODE,*FILL=06,rrrrKY01,*FILL=06,rrrrKY02,END.
```

The PDM skips the first 5 bytes, which you have initialized to **CODE=**, returns the *rrrrCODE* value into the next 2 bytes, skips the next six bytes, and so on, resulting in the line you wanted.

When processing add or write commands, do not use ***FILL=nn** unless there are also data items in the data list.

****CODE**** This keyword is for coded related files only, to process only certain record codes. For a read command, it returns the record code value in the first 2 bytes of the data area. For an update or add command, it gets the record code from the first 2 bytes of the data area. This keyword, if used, must be the first element in the data list (possibly ****BIND**** is first). This keyword is a prerequisite for using the ***CODE=xx** keyword.

CODE=xx** This keyword is for coded related files only, to process only certain record codes. For each occurrence, it identifies which record code the following sub-data list refers to. This keyword must not be the first item in the data list; it must follow the *CODE**** keyword or another ***CODE=xx**. It must not be the last item in the list; at least one data item must follow it.

On most read commands, the PDM merely skips any records with record codes not identified in the data list. However, if the read command is a **READD**, the PDM returns an error status if the code is not identified. On a write or add command, the PDM performs the request only if the record in the data area has one of the record codes identified in the data list. Otherwise the PDM returns an error status. Use ***CODE=xx** as follows:

```
**CODE**,*CODE=xx,dataitem1,*CODE=xx,dataitem1,dataitem2,END.
```

The following examples further describe the action the PDM takes on the various types of read, and on adds or updates. All examples refer to the following data list:

```
**CODE** , *CODE=01 , RECSelml , *CODE=02 , RECSelml , RECSkey1
```

1. When issuing a **READD** command (read direct), the PDM uses the RRN in the READD's *reference* parameter to locate the record to be read. Then, the PDM fills the data area with different information depending on the record code.

According to the example data list, if the RRN points to a record whose record code is 02, the data area contains:

```
02xy00001
```

If the RRN points to a record whose record code is 01, the data area contains:

```
01xy
```

However, if the **READD** RRN points to a record whose record code is 03, the PDM returns an error status code.

2. When performing a series of reads using **FINDX**, **RDNXT**, **READR**, or **READV** for a coded file, using the above example data list, the PDM returns information as described below to the data area and status.

Notice that RRN 68 and 70 have a record code of 03, but record code 03 is not specified in the example data list. When processing a **FINDX**, **RDNXT**, **READV**, or **READR**, the PDM simply skips that record and processes the next record in the chain. You do not receive an error status as you would on a **READD** (see #1).

3. When performing an **ADDVA**, **ADDVB**, **ADDVC**, **ADDVR** or **WRITV** with the example coded data list, the PDM processes the request only if the data area record code is in the data list. The first 2 bytes of the data area determine which portion of the data list maps the remainder of data in the data area (*CODE=01 or *CODE=02). If another record code is in the data area, the PDM returns an error status.

COMMON This keyword is a compatibility form of ****CODE****, to make the data list a coded data list. Two formats are supported for existing TOTAL and TIS 1.x applications:

The following rules apply when coding a ***COMMON*** data list:

- ◆ Use the ***COMMON*** keyword for compatibility only. For new applications, use ****CODE**** (or use the RDML language for the application).
- ◆ Unlike ****CODE****, ***COMMON*** must be followed by at least one data item before a ***CODE=xx** item.
- ◆ A data list should follow each ***CODE=xx** entry. If you do have data items common to more than one record code, name them in the ***COMMON*** base data list or in each ***CODE=xx** sublist, as appropriate.
- ◆ Do not name the record code item in any coded data list unless you also name it as the first item in the base data list.
- ◆ When writing or adding a record (using the ***COMMON*** keyword), include the record code as the first data item in the base data list. This allows the PDM to determine which of the coded data lists maps the data area. If you omit the record code or include it somewhere other than at the beginning of the base data list, results are unpredictable.

3

Command syntax

This chapter contains syntax descriptions for the PDM DML commands. The commands are presented alphabetically for easy reference. DML command parameters are positional and must appear in the order indicated in the format box. The parameter descriptions explain the required and optional parts of the format, default values, options, and considerations. The General considerations explain any special considerations for using the command.

Each DML command returns a status code indicating the successful completion of the operation or the nature of the failure. Check the status code after each DML command, and take corrective action.

ADD-M

The ADD-M (Add Primary) command adds a record to a primary file.

ADD-M, status, file, control-key, data-list, data-area, end

status

Description *Required.* Points to a field into which the PDM places a status code indicating the result of the command.

Format 4-byte field

Considerations

- ◆ If the command fails or if the status code indicates some special condition other than failure, your program should include logic to handle and possibly correct the situation. For a list of all status codes, refer to the *SUPRA Server PDM Messages and Codes Reference Manual (RDM/PDM Support for OS/390 & VSE)*, P26-0126.
- ◆ Code your program to handle the following status codes:
 - FULL If the file is a KSDS file, you can get a FULL status when trying to add one record and a **** status on the next add. This depends on where in the file you are trying to add the record and how full the file is. Thus, FULL on a KSDS file may mean only that you cannot add a particular record.
 - DUPM Returned when the file contains a record whose key equals the control key of the record you are trying to add.
 - HELD When accessing a database record currently being held by another task, the PDM waits until a user-defined time-out counter expires before it reexamines the status of the lock request. If the record is still HELD, the PDM returns the HELD status to the application. The application may elect to terminate the task or reissue the command. If you choose recycle logic for HELD statuses, you should implement an upper limit to the mechanism in the program logic.
 - EMBR As with the HELD status, the PDM may return EMBR when another task is using the database record. Distinguishing the EMBR from the HELD status is the occurrence of a deadly embrace. Deadly embrace occurs when two tasks, already owners of a resource (database record), attempt to gather each other's resources. To handle an EMBR status, restart the logical unit of work. Alternatively, you can introduce a retry mechanism with an upper limit retry count in your program logic.

file

- Description** *Required.* Identifies the primary file to be acted upon. You can define a field containing the name or you can code the actual file name as a literal in the CALL statement.
- Format** 4 alphanumeric characters; first character must be alphabetic
- Consideration** The file must be a primary file, or the PDM returns a fatal status code (FNTE).

control-key

- Description** *Required.* Points to a field containing the key of the primary record to be added. The PDM uses this key to determine whether a record with the same control key already exists.
- Format** Variable length as defined on the Directory
- Consideration** During the command processing, if the *control-key* parameter does not match the corresponding field in your data area, a status code informs you of the failure. To avoid this, you should use the *control-key* field name in the data area for this parameter, rather than define a separate field.

data-list

Description *Required.* Points to a field containing a list of data items. This list acts as a map of the layout of the data area. Compose this list using data names (physical fields) defined on the Directory.

Format *dataitem1,dataitem2,...dataitemn,END.*

Considerations

- ◆ The commas between the entries are optional and only serve as separators; be consistent whether you use them or not.
- ◆ The data list can include the names of data items and control keys. Do not name linkpaths or the root field. Do not list any name twice. If a name is not accepted, the PDM returns an error status code.
- ◆ You can list the data names in any order. They are processed in the order listed, not in the order defined on the Directory.
- ◆ **VSE** When using a cross-address space central PDM in VSE/AF with XPCC=YES, the data area mapped by this *data-list* parameter is limited by the CSIPARM MAXIO value. This governs DML having no *length* parameter. Refer to the *SUPRA Server PDM and Directory Administration Guide (OS/390 & VSE)*, P26-2250, for CSIPARM information.
- ◆ Data items omitted from the data list and data area are filled with blanks in the added record.

data-area

Description *Required.* Points to a field containing the data that ADD-M writes on the primary file.

Format The structure and characteristics of the data area must conform exactly to the Directory definition of the data items (physical fields) named in the data list.

Considerations

- ◆ The data area must be large enough to hold values for all data items named in the data list.
- ◆ The data area and the data list have corresponding fields. The data list holds names, and the data area holds a value for each of those names. You must include the control key name in the data list and the control key value in the data area.

end

Description *Required.* Points to a field that delimits the parameter list.

Format END.

General considerations

- ◆ If the value of the *control-key* parameter does not match the value of the control key in the data area, the PDM returns an error status code.
- ◆ Do not execute ADD-M commands between consecutive serial functions, as this can result in records being skipped or read multiple times in the serial scan.
- ◆ When you add this new record, the PDM maintains all populated secondary keys in the specified data file accordingly.

ADDVA

The ADDVA (Add Related After) command inserts the new record after the record whose RRN is in the *reference* parameter. This placement occurs only on the linkpath specified by the *linkpath* parameter. On all other linkpaths defined for this record, the PDM adds the record to the end of the respective chains.

ADDVA, status, file, reference, linkpath, control-key, data-list, data-area, end

status

Description *Required.* Points to a field into which the PDM places a status code indicating the result of the command.

Format 4-byte field

Considerations

- ◆ If the command fails or if the status code indicates some special condition other than failure, your program should include logic to handle and possibly correct the situation. For a list of all status codes, refer to the *SUPRA Server PDM Messages and Codes Reference Manual (RDM/PDM Support for OS/390 & VSE)*, P26-0126.
- ◆ Code your program to handle the following status codes:
 - FULL The PDM returns this status code when no more records can be added to a file.
 - MRNF The specified *control-key* parameter value or a *linkpath* key value does not exist in the respective primary file.
 - HELD When accessing a database record currently being held by another task, the PDM waits until a user-defined time-out counter expires before it reexamines the status of the lock request. If the record is still HELD, the PDM returns the HELD status to the application. The application may elect to terminate the task or reissue the command. If you choose recycle logic for HELD statuses, you should implement an upper limit to the mechanism in the program logic.
 - EMBR As with the HELD status, the PDM may return EMBR when another task is using the database record. Distinguishing the EMBR from the HELD status is the occurrence of a deadly embrace. Deadly embrace occurs when two tasks, already owners of a resource (database record), attempt to gather each other's resources. To handle an EMBR status, restart the logical unit of work. Alternatively, you can introduce a retry mechanism with an upper limit retry count in your program logic.

file

- Description** *Required.* Identifies the related file to be acted upon. You can define a field containing the name or you can use the actual file name as a literal in the CALL statement.
- Format** 4 alphanumeric characters; first character must be alphabetic
- Consideration** The file must be a related file, or the PDM returns a fatal status code (FNTF).

reference

- Description** *Required.* Points to a field identifying the RRN of the record after which to add the new record in this chain.
- Format** 4 alphanumeric characters or a binary fullword
- Options**
- | | |
|------|--|
| LKxx | Identifies the last 4 characters of the linkpath named by the <i>linkpath</i> parameter. Substitute the actual characters for xx. The PDM adds the record to the end of the linkpath instead of after a particular record. |
| rrrr | Identifies the RRN of the record after which the new record is to be added to the linkpath. |
- Consideration** After successful execution, this parameter contains the RRN of the record just added.

linkpath

Description *Required.* Points to a field containing the name of the linkpath as defined on the Directory. This parameter indicates which related record chain the PDM is to process.

Format *ppppLKxx* where *pppp* identifies the name of an associated primary file, LK must appear as shown, and *xx* represents the last 2 characters of the linkpath name as defined on the Directory.

Considerations

- ◆ If you specify an invalid linkpath, the PDM returns an error status code.
- ◆ A related file record can contain more than one linkpath, but the *linkpath* parameter names only the controlling linkpath. Each linkpath must be maintained. ADDVA holds all affected primary and related records and updates the linkpath pointers.

control-key

Description *Required.* Points to a field containing the key of the record in the primary file named in the *linkpath* parameter.

Format Variable length as defined on the Directory

Consideration During processing, if the *control-key* field does not match the corresponding field in your data area, a status code informs you of the failure. To avoid this, you should use the *control-key* field name in the data area for this parameter, rather than define a separate field.

data-list

Description *Required.* Points to a variable-length field containing a list of data items. This list acts as a map of the layout of the data area. Compose this list using data names (physical fields) defined on the Directory.

Format *dataitem1,dataitem2,...dataitemn,END.*

Considerations

- ◆ The commas between the entries are optional and only serve as separators; be consistent whether you use them or not.
- ◆ The data list can include the names of data items, control keys, and record codes. Do not name linkpaths. Do not list any name twice. If a name is not accepted, the PDM returns an error status code.
- ◆ You can list the data names in any order. They are processed in the order listed, not in the order defined on the Directory. However, for coded records, the record code must be first in the list. If not, unpredictable results occur. See “[Data list parameter keywords](#)” on page 62 for coded data list construction rules.
- ◆ **VSE** When using a cross-address space central PDM in VSE/AF with XPCC=YES, the data area mapped by this *data-list* parameter is limited by the CSIPARM MAXIO value. This governs DML having no *length* parameter. Refer to the [SUPRA Server PDM and Directory Administration Guide \(OS/390 & VSE\)](#), P26-2250, for CSIPARM information.
- ◆ Data items omitted from the data list and data area are filled with blanks in the added record.

data-area

Description *Required.* Points to a field containing the data that ADDVA writes on the related file.

Format The structure and characteristics of the data area must conform exactly to the Directory definition of the data items (physical fields) in the data list.

Considerations

- ◆ The data area must be large enough to hold values for all data items in the data list.
- ◆ The data area and the data list have corresponding fields. The data list holds names, and the data area holds a value for each of those names. You must include the control key name in the data list and the control key value in the data area.

end

Description *Required.* Points to a field that delimits the parameter list.

Format END.

General considerations

- ◆ Adding a record requires the PDM to retrieve a primary record for each control key defined for that record or record code. Therefore, these control keys must be present in the data list and data area. They must be valid (they must represent records which actually exist in the respective primary files). For coded records, the data area must contain the data for only one record code.
- ◆ If the value of the *control-key* parameter does not match the value of the control key in the data area, the PDM returns an error status code.
- ◆ When you add this new record, the PDM maintains all populated secondary keys in the specified data file accordingly.

ADDVB

The ADDVB (Add Related Before) command adds the new record before the record whose RRN is in the *reference* field. This placement occurs only on the linkpath specified by the *linkpath* parameter. On all other linkpaths defined for this record, the PDM adds the record to the end of the respective chains.

ADDVB, status, file, reference, linkpath, control-key, data-list, data-area, end

status

Description *Required.* Points to a field into which the PDM places a status code indicating the result of the command.

Format 4-byte field

Considerations

- ◆ If the command fails or if the status code indicates some special condition other than failure, your program should include logic to handle and possibly correct the situation. For a list of all status codes, refer to the *SUPRA Server PDM Messages and Codes Reference Manual (RDM/PDM Support for OS/390 & VSE)*, P26-0126.
- ◆ Code your program to handle the following status codes:
 - FULL The PDM returns this status code when no more records can be added to a file.
 - MRNF The specified *control-key* parameter value or a *linkpath* key value does not exist in the respective primary file.
 - HELD When accessing a database record currently being held by another task, the PDM waits until a user-defined time-out counter expires before it reexamines the status of the lock request. If the record is still HELD, the PDM returns the HELD status to the application. The application may elect to terminate the task or reissue the command. If you choose recycle logic for HELD statuses, you should implement an upper limit to the mechanism in the program logic.
 - EMBR As with the HELD status, the PDM may return EMBR when another task is using the database record. Distinguishing the EMBR from the HELD status is the occurrence of a deadly embrace. Deadly embrace occurs when two tasks, already owners of a resource (database record), attempt to gather each other's resources. To handle an EMBR status, restart the logical unit of work. Alternatively, you can introduce a retry mechanism with an upper limit retry count in your program logic.

file

- Description** *Required.* Identifies the related file to be acted upon. You can define a field containing the name or you can code the actual file name as a literal in the CALL statement.
- Format** 4 alphanumeric characters; first character must be alphabetic
- Consideration** The file must be a related file.
-

reference

- Description** *Required.* Points to a field identifying the RRN of the record before which to add the new record in this chain.
- Format** 4 alphanumeric characters or a binary fullword
- Options**
- | | |
|------|--|
| LKxx | Identifies the last 4 characters of the linkpath named by the <i>linkpath</i> parameter. Substitute the actual characters for xx. The PDM adds the record to the very beginning of the linkpath instead of before a particular record. |
| rrrr | Identifies the RRN of the record before which the new record is to be added to the linkpath. |
- Consideration** After successful execution, this parameter contains the RRN of the record just added.

linkpath

Description *Required.* Points to a field containing the name of the linkpath as defined on the Directory. This parameter indicates which related record chain is to be processed.

Format *ppppLKxx* where *pppp* identifies the name of an associated primary file, LK must appear as shown, and *xx* represents the last 2 characters of the linkpath name as defined on the Directory.

Considerations

- ◆ If you specify an invalid linkpath, the PDM returns an error status code.
- ◆ A related file record can contain more than one linkpath, but the *linkpath* parameter names only the controlling linkpath. Each linkpath must be maintained. ADDVB holds all affected primary and related records and updates the linkpath pointers.

control-key

Description *Required.* Points to a field containing the key of the record in the primary file named in the *linkpath* parameter.

Format Variable length as defined on the Directory

Consideration During the command processing, if the *control-key* parameter does not match the corresponding field in your data area, a status code informs you of the failure. To avoid this, you should use the *control-key* field name in the data area for this parameter, rather than define a separate field.

data-list

Description *Required.* Points to a variable-length field containing a list of data items. This list acts as a map of the layout of the data area. Compose this list using data names (physical fields) defined on the Directory.

Format *dataitem1,dataitem2,...dataitemn,END.*

Considerations

- ◆ The commas between the entries are optional and only serve as separators; be consistent whether you use them or not.
- ◆ The data list can include the names of data items, control keys, and record codes. Do not name linkpaths. Do not list any name twice. If a name is not accepted, the PDM returns an error status code.
- ◆ You can list the data names in any order. They are processed in the order listed, not in the order defined on the Directory. However, for coded records, the record code must be first in the list. If not, unpredictable results occur. See “[Data list parameter keywords](#)” on page 62 for coded data list construction rules.
- ◆ **VSE** When using a cross-address space central PDM in VSE/AF with XPCC=YES, the data area mapped by this *data-list* parameter is limited by the CSIPARM MAXIO value. This governs DML having no *length* parameter. Refer to the [SUPRA Server PDM and Directory Administration Guide \(OS/390 & VSE\)](#), P26-2250, for CSIPARM information.
- ◆ Data items omitted from the data list and data area are filled with blanks in the added record.

data-area

Description *Required.* Points to a field containing the data that ADDVB writes on the related file.

Format The structure and characteristics of the data area must conform exactly to the Directory definition of the data items (physical fields) named in the data list.

Considerations

- ◆ The data area must be large enough to hold values for all data items named in the data list.
- ◆ The data area and the data list have corresponding fields. The data list holds names, and the data area holds a value for each of those names. You must include the control key name in the data list and the control key value in the data area.

end

Description *Required.* Points to a field that delimits the parameter list.

Format END.

General considerations

- ◆ Adding a record requires the PDM to retrieve a primary record for each control key defined for that record or record code. Therefore, these control keys must be present in the data list and data area. They must be valid (they must represent records which actually exist in the respective primary files). For coded records, the data area must contain the data for only one record code.
- ◆ If the value of the *control-key* parameter does not match the value of the control key in the data area, the PDM returns an error status code.
- ◆ When you add this new record, the PDM maintains all populated secondary keys in the specified data file accordingly.

ADDVC

The ADDVC (Add Related Continue) command adds the new record to the end of the chain on the controlling linkpath and on all other linkpaths defined for the record.

ADDVC, status, file, reference, linkpath, control-key, data-list, data-area, end

status

Description *Required.* Points to a field into which the PDM places a status code indicating the result of the command.

Format 4-byte field

Considerations

- ◆ If the command fails or if the status code indicates some special condition other than failure, your program should include logic to handle and possibly correct the situation. For a list of all status codes, refer to the *SUPRA Server PDM Messages and Codes Reference Manual (RDM/PDM Support for OS/390 & VSE)*, P26-0126.
- ◆ Code your program to handle the following status codes:
 - FULL The PDM returns this status code when no more records can be added to a file.
 - MRNF The specified *control-key* parameter value or a *linkpath* key value does not exist in the respective primary file.
 - HELD When accessing a database record currently being held by another task, the PDM waits until a user-defined time-out counter expires before it reexamines the status of the lock request. If the record is still HELD, the PDM returns the HELD status to the application. The application may elect to terminate the task or reissue the command. If you choose recycle logic for HELD statuses, you should implement an upper limit to the mechanism in the program logic.
 - EMBR As with the HELD status, the PDM may return EMBR when another task is using the database record. Distinguishing the EMBR from the HELD status is the occurrence of a deadly embrace. Deadly embrace occurs when two tasks, already owners of a resource (database record), attempt to gather each other's resources. To handle an EMBR status, restart the logical unit of work. Alternatively, you can introduce a retry mechanism with an upper limit retry count in your program logic.

file

- Description** *Required.* Identifies the related file to be acted upon. You can define a field containing the name or you can code the actual file name as a literal in the CALL statement.
- Format** 4 alphanumeric characters; first character must be alphabetic
- Consideration** The file must be a related file.

reference

- Description** *Required.* Points to a field into which the PDM places the RRN of the added record.
- Format** 4 alphanumeric characters or a binary fullword
- Consideration** The content of the *reference* field is ignored on input with the ADDVC function. After successful execution, the *reference* field contains the RRN of the record just added.

linkpath

- Description** *Required.* Points to a field containing the name of the linkpath as defined on the Directory. This parameter indicates which related record chain is to be processed.
- Format** *ppppLKxx* where *pppp* identifies the name of an associated primary file, LK must appear as shown, and *xx* represents the last 2 characters of the linkpath name as defined on the Directory.

Considerations

- ◆ If you specify an invalid linkpath, the PDM returns an error status code.
- ◆ A related file record can contain more than one linkpath, but the *linkpath* parameter names only the controlling linkpath. Each linkpath must be maintained. ADDVC holds all affected primary and related records and updates the linkpath pointers.

control-key

- Description** *Required.* Points to a field containing the key of the record in the primary file named in the *linkpath* parameter.
- Format** Variable length as defined on the Directory
- Consideration** During the command processing, if the *control-key* parameter does not match the corresponding field in your data area, a status code informs you of the failure. To avoid this, you should use the *control-key* field name in the data area for this parameter, rather than define a separate field.
-

data-list

- Description** *Required.* Points to a variable-length field containing a list of data items. This list acts as a map of the layout of the data area. Compose this list using data names (physical fields) defined on the Directory.
- Format** *dataitem1,dataitem2,...dataitemn,END.*

Considerations

- ◆ The commas between the entries are optional and only serve as separators; be consistent whether you use them or not.
- ◆ The data list can include the names of data items, control keys, and record codes. Do not name linkpaths. Do not list any name twice. If a name is not accepted, the PDM returns an error status code.
- ◆ You can list the data names in any order. They are processed in the order listed, not in the order defined on the Directory. However, for coded records, the record code must be first in the list. If not, unpredictable results occur. See “[Data list parameter keywords](#)” on page 62 for coded data list construction rules.
- ◆ **VSE** When using a cross-address space central PDM in VSE/AF with XPCC=YES, the data area mapped by this *data-list* parameter is limited by the CSIPARM MAXIO value. This governs DML having no *length* parameter. Refer to the [SUPRA Server PDM and Directory Administration Guide \(OS/390 & VSE\)](#), P26-2250, for CSIPARM information.
- ◆ Data items omitted from the data list and data area are filled with blanks in the added record.

data-area

Description *Required.* Points to a field containing the data that ADDVC writes on the related file.

Format The structure and characteristics of the data area must conform exactly to the Directory definition of the data items (physical fields) named in the data list.

Considerations

- ◆ The data area must be large enough to hold values for all data items named in the data list.
- ◆ The data area and the data list have corresponding fields. The data list holds names, and the data area holds a value for each of those names. You must include the control key name in the data list and the control key value in the data area.

end

Description *Required.* Points to a field that delimits the parameter list.

Format END.

General considerations

- ◆ Adding a record requires the PDM to retrieve a primary record for each control key defined for that record or record code. Therefore, these control keys must be present in the data list and data area. They must be valid (they must represent records which actually exist in the respective primary files). For coded records, the data area must contain the data for only one record code.
- ◆ If the value of the *control-key* parameter does not match the value of the control key in the data area, the PDM returns an error status code.
- ◆ When you add this new record, the PDM maintains all populated secondary keys in the specified data file accordingly.

ADDVR

The ADDVR (Add Related Replace) command is actually an update function which allows one or more control keys, or the record code if coded file, to be changed on an existing record. This functions similarly to a [DELVD](#) and [ADDVC](#) without physically moving the record (the RRN remains the same). If a control key value in the data area is different from the value of the existing record at that RRN, ADDVR unlinks and relinks the existing related record into the new chain(s).

ADDVR, status, file, reference, linkpath, control-key, data-list, data-area, end

status

Description *Required.* Points to a field into which the PDM places a status code indicating the result of the command.

Format 4-byte field

Considerations

- ◆ If the command fails or if the status code indicates some special condition other than failure, your program should include logic to handle and possibly correct the situation. For a list of all status codes, refer to the [SUPRA Server PDM Messages and Codes Reference Manual \(RDM/PDM Support for OS/390 & VSE\)](#), P26-0126.
- ◆ Code your program to handle the following status codes:
 - MRNF You have specified a new control key value for a linkpath, and that value does not exist in the associated primary file.
 - HELD When accessing a database record currently being held by another task, the PDM waits until a user-defined time-out counter expires before it reexamines the status of the lock request. If the record is still HELD, the PDM returns the HELD status to the application. The application may elect to terminate the task or reissue the command. If you choose recycle logic for HELD statuses, you should implement an upper limit to the mechanism in the program logic.
 - EMBR As with the HELD status, the PDM may return EMBR when another task is using the database record. Distinguishing the EMBR from the HELD status is the occurrence of a deadly embrace. Deadly embrace occurs when two tasks, already owners of a resource (database record), attempt to gather each other's resources. To handle an EMBR status, restart the logical unit of work. Alternatively, you can introduce a retry mechanism with an upper limit retry count in your program logic.

file

Description *Required.* Identifies the related file to be acted upon. You can define a field containing the name or you can code the actual file name as a literal in the CALL statement.

Format 4 alphanumeric characters; first character must be alphabetic

Consideration The file must be a related file.

reference

Description *Required.* Points to a field identifying the RRN of the existing record to be compared to your data area.

Format 4 alphanumeric characters or a binary fullword

Consideration After successful completion, the RRN remains the same.

linkpath

- Description** *Required.* Points to a field containing the name of the linkpath as defined on the Directory. This parameter indicates which related record chain is to be processed.
- Format** *ppppLKxx* where *pppp* identifies the name of an associated primary file, *LK* must appear as shown, and *xx* represents the last 2 characters of the linkpath name as defined on the Directory.

Considerations

- ◆ If you are changing the record's code, this field must contain a linkpath for the new coded record.
- ◆ If you specify an invalid linkpath, the PDM returns an error status code.
- ◆ If the linkpath is in the old record and the new record, and the control key does not change, the PDM does not disturb that chain.
- ◆ If the linkpath is in the old record and the new record, and the control key changes, the PDM removes the record's RRN from the old chain and adds it to the end of the new chain.
- ◆ If the linkpath is in the old record but not in the new record, the PDM removes the record's RRN from the old chain and adds it to the end of the new chain.
- ◆ If a linkpath other than this one is affected (a different or new control key in the data area for other linkpaths), similar relinking occurs.
- ◆ ADDVR holds all affected primary and related records and updates the linkpath pointers.

control-key

Description *Required.* Points to a field containing the key of the record in the primary file named by the *linkpath* parameter.

Format Variable length as defined on the Directory

Considerations

- ◆ During processing, if the *control-key* field does not match the corresponding field in your data area, a status code informs you of the failure. To avoid this, you should use the *control-key* field name in the data area for this parameter, rather than define a separate field.
- ◆ If you are changing the key value associated with the *linkpath* parameter, this field must contain the new key value.

data-list

Description *Required.* Points to a variable-length field containing a list of data items. This list acts as a map of the layout of the data area. Compose this list using data names (physical fields) defined on the Directory.

Format dataitem1,dataitem2,...dataitemn,END.

Considerations

- ◆ The commas between the entries are optional and only serve as separators; be consistent whether you use them or not.
- ◆ The data list can include the names of data items, control keys, and record codes. Do not name linkpaths. Do not list any name twice. If a name is not accepted, the PDM returns an error status code.
- ◆ You can list the data names in any order. They are processed in the order listed, not in the order defined on the Directory. However, for coded records, the record code must be first in the list. If not, unpredictable results occur. See “[Data list parameter keywords](#)” on page 62 for coded data list construction rules.
- ◆ **VSE** When using a cross-address space central PDM in VSE/AF with XPCC=YES, the data area mapped by this *data-list* parameter is limited by the CSIPARM MAXIO value. This governs DML having no *length* parameter. Refer to the [SUPRA Server PDM and Directory Administration Guide \(OS/390 & VSE\)](#), P26-2250, for CSIPARM information.
- ◆ Whether or not you are changing the record code, data items omitted from the data list and data area are filled with blanks in the revised record.
- ◆ If you are changing the record's code, this list must name data items that are valid for the new code.
- ◆ You can update the value of any of the data items in addition to updating control keys or record code.

data-area

Description *Required.* Points to a field containing the data that ADDVR writes on the related file.

Format The structure and characteristics of the data area must conform exactly to the Directory definition of the data items named in the data list.

Considerations

- ◆ The data area must be large enough to hold values for all data items named in the data list.
- ◆ The data area and the data list have corresponding fields. The data list holds names and the data area holds a value for each of those names. You must include the control key name in the data list and the control key value in the data area.

end

Description *Required.* Points to a field that delimits the parameter list.

Format END.

General considerations

- ◆ This is the only command you can use to change a record code or a *control-key* field.
- ◆ If you change a record code, it could add or delete a linkpath in the record. The PDM examines the linkpaths defined in the Directory for the old and new record codes. Any linkpath not common to both codes is maintained as required. Additions are made to the logical end of the chain(s) controlled by the control key(s) in your *data-area* parameter.
- ◆ If a related record is controlled by records in several primary files, you must include all control keys in the data list and data area. If it is a coded record, you include only those control keys that apply to the record code.
- ◆ You can obtain the RRN for a record by issuing a read command (e.g., **READV**). The *reference* field returned by the READV contains the RRN of the record just read. In a multitasking environment or if task logging is active, you must issue the read command with END. specified as the end parameter before performing the ADDVR.
- ◆ If the *control-key* parameter value does not match the corresponding value in the data area, the PDM returns an error status code.
- ◆ When you revise this record, the PDM maintains all populated secondary keys in the specified data file accordingly.

CLOSX

CLOSX (close file) logically, and if necessary physically, closes primary and related files. Logically closing includes checking and updating each file's control record. Physically closing involves issuing the CLOSE command of the host operating system.

In your environment description, there is an OPENX-OPTION. Its setting determines the outcome of your CLOSX as follows:

- ◆ If OPENX-OPTION=IGNORE, the PDM does not carry out the request. You always receive a status indicating success.
- ◆ If OPENX-OPTION=PROCESS, CLOSX can change the file mode, unlock the file, and issue the operating system CLOSE command. These effects depend on the current status lock condition and mode of the file, and on the type of CLOSX you issue.
- ◆ If OPENX-OPTION=CHECK, the PDM does not carry out the request, but the CLOSX can return an error status.
- ◆ If OPENX-OPTION=CHECK or OPENX-OPTION=IGNORE, you must terminate the PDM to close any file. Your DBA is most likely to use the CHECK or IGNORE options if you use a central operating mode and the file/environment description relationship defines automatic opening.

You can find which setting is in effect by using a **SHOWX** command with the ENV DOPNX data item. The tables at the end of this section present the various situations and results when the option is PROCESS or CHECK.

CLOSX, status, realm, end

status

Description *Required.* Points to a field into which the PDM places a status code indicating the overall result of the command.

Format 4-byte field

Considerations

- ◆ If the command fails or if the status code indicates some special condition other than failure, your program should include logic to handle and possibly correct the situation. For a list of all status codes, refer to the *SUPRA Server PDM Messages and Codes Reference Manual (RDM/PDM Support for OS/390 & VSE)*, P26-0126.
- ◆ Code your program to handle the following status codes:
 - CERR Close error. The tables at the end of this section present the various file conditions possible, and whether a CLOSX results in a CERR status.
 - FNAV File is not available for you to close. Issue a **SHOWX** to obtain extended status codes.
- ◆ If you use **ALL.** in *realm*, the stat field in that parameter is not set. If you use a file-list, the stat field for the file causing the error is set to the same value as this *status* parameter unless the error is FNTF.

realm

Description *Required.* Points to a list of the REALM= keyword and one or more 12-character entries followed by END. This determines which files to close and in what mode.

Format $REALM = \left\{ \begin{array}{l} ALL.modestat \\ filemodstat1, filemodestat2, \dots \end{array} \right\}, END.$

where:

ALL Closes all user files related to this environment description. This does not include Directory files.

file Identifies the 4-character name of a primary or related file you want to close.

mode Specifies the type of close you want. The options are:

PART Performs a logical close. This completes pending output to the file, and unlocks the file if currently locked. Leaves mode unchanged if currently READ or IUPD, or changes mode to IUPD if currently SUPD or EUPD. (This mode is not valid for KSDS.)

COMP Performs a physical close. This completes pending output to the file, unlocks the file if currently locked, changes the mode to null, and issues the operating system CLOSE command.

stat Provides an area to receive the PDM status code indicating the result of closing an individual file. You initialize these fields to blank. If you use ALL., this field is not set upon completion but the command's status field is set.

END. Specifies the delimiter for the *realm* parameter.

Considerations

- ◆ The commas after each “*filemodestat*” entry are optional and only serve as separators; be consistent whether you use them or not. Do not use commas within an entry.
- ◆ CLOSX returns a CERR status if another task previously opened the file for exclusive update (OPENX EUPD mode). Conversely, if your task opens a file for EUPD, it is the only task allowed to close it. In most cases, you should reopen it for sharing.
- ◆ KSDS cannot be partially closed. Therefore, you should not use a “*filePARTstat*” entry for KSDS in a file list. You also should not use “ALL.PART*stat*” if the active environment description includes KSDS. If you use PART, the PDM returns an error status. You are allowed to issue a CLOSX with COMP for KSDS.
- ◆ Index files are not permitted in the CLOSX file list. See “[Diagnosing application DML errors](#)” on page 36 about index files.
- ◆ If you code a file list, and a file name is not found, the PDM does not set that entry’s stat field. Instead, it sets the command’s *status* field to FNTF, and the function is immediately terminated. (This measure protects memory from being over-written if the list delimiter (END.) is missing.) To determine which file caused the error, your program can issue a [SHOWX](#) for extended status.
- ◆ If the PDM processes some of the files successfully, then encounters an error, the PDM does the following:
 - If you used a file list, sets the error code in the stat field for the file in error (unless FNTF).
 - Sets the error code in the command *status* field.
 - Reprocesses all involved files to their original state.
 - Does not process the remaining files.

end

Description	<i>Required.</i> Points to a field that delimits the parameter list.
Format	END.

General considerations

- ◆ If any fatal errors occur on any file in the realm, the PDM restores all files to their states that existed before executing this CLOX. In a task logging environment, you should issue a **RESET** after a CLOX that receives a fatal error. Issue a **COMIT** after a successful CLOX execution.
- ◆ Do not issue both an **OPENX** and a CLOX within the same logical unit of work.
- ◆ If you use task logging and your task changes a file's mode (even if backed out because of errors), the PDM keeps other tasks from changing the file's mode until your task issues a **COMIT** or **RESET**. Any other task that tries to change the file's mode must wait. The wait is according to the environment description delay time for held records (the file lock record is being held). If your task does not issue a **COMIT** or **RESET** to free the file during this time, the other task receives an error status code.
- ◆ If you use task logging, a **SINOF** is not sufficient to commit or reset an **OPENX** or CLOX.
- ◆ Index files cannot be closed directly. When you use **COMP** mode to close the last primary or related file having secondary keys on a particular index file, the index file also closes **COMP**. When you close a primary or related file with **PART** mode, any associated index files that were locked remain locked. To unlock the index files, you must close all associated primary or related files with **COMP** mode.

- ◆ CLOSX processing results depend upon the following factors:
 - The current open or closed state of the file.
 - The current value of the lock record.
 - The current file mode of the file.
 - The mode you request in the *realm* parameter.
 - The environment description PDM access mode (RONLY or UPDATE).
 - The OPENX-OPTION in the active user environment description. If the option is IGNORE, you receive a status of ****, but no close processing occurs. If the option is PROCESS and any records are reserved by another task, an error status code is returned. If the option is PROCESS or CHECK, the CLOSX is processed as shown in the following tables:

Current file condition				Resultant file condition			
File state	File lock record	File mode	Command attempted	File state	File lock record	File mode	Status code returned
Closed	*	*	CLOSX (PART)	-	-	-	****
Open	*	READ		-	-	-	****
Open	*	IUPD		-	-	-	****
Open	*	SUPD		-	Unlocked	IUPD	****
Open	*	EUPD (S)		-	Unlocked	IUPD	****
Open	*	EUPD (D)		-	-	-	CERR
Closed	*	*	CLOSX (COMP)	-	-	null	****
Open	*	READ		Closed	-	null	****
Open	*	IUPD		Closed	-	null	****
Open	*	SUPD		Closed	Unlocked	null	****
Open	*	EUPD (S)		Closed	Unlocked	null	****
Open	*	EUPD (D)		-	-	-	CERR
Closed	*	*	CLOSX (PART)	-	-	-	CERR
Open	*	READ		-	-	-	****
Open	*	SUPD		-	-	-	CERR
Closed	*	*	CLOSX (COMP)	-	-	-	****
Open	*	READ		-	-	-	CERR
Open	*	SUPD	-	-	-	CERR	

- * PDM does not check (S) Same task attempts the close
- PDM does not change (D) Different task attempts the close

Concerning the preceding table, when the OPENX-OPTION=CHECK, the file can be only closed or open for READ or SUPD. This is according to the file/environment description relationship, where EUPD is not an option.

COMIT

The COMIT command establishes a Task Level Recovery point (commit point) for this task on the Task Log File. COMIT ends a logical unit of work, makes permanent all update and file mode changes, releases all records that are held or locked by the issuing task, and writes a COMIT record. You can also pass a data area to the Task Log File with the COMIT. If the PDM recovers the task to this COMIT, the PDM returns this data area to your task if you issue a **RESET** after **SINON**.

COMIT, *status, comit-id, length, data-area, end*

status

Description *Required.* Points to a field into which the PDM places a status code indicating the result of the command.

Format 4-byte field

Consideration If the command fails or if the status code indicates some special condition other than failure, your program should include logic to handle and possibly correct the situation. For a list of all status codes, refer to the *SUPRA Server PDM Messages and Codes Reference Manual (RDM/PDM Support for OS/390 & VSE)*, P26-0126.

comit-id

Description	<i>Required.</i> Points to an area containing the literal ASGN or a commit number to be the identification of this commit point.				
Format	4-byte field				
Options	<table><tr><td>ASGN</td><td>The PDM assigns a commit number that is one higher than the previous commit of this task, if any.</td></tr><tr><td><i>comit-number</i></td><td>The PDM uses this 4-byte binary number. It must be at least one higher than the previous commit ID.</td></tr></table>	ASGN	The PDM assigns a commit number that is one higher than the previous commit of this task, if any.	<i>comit-number</i>	The PDM uses this 4-byte binary number. It must be at least one higher than the previous commit ID.
ASGN	The PDM assigns a commit number that is one higher than the previous commit of this task, if any.				
<i>comit-number</i>	The PDM uses this 4-byte binary number. It must be at least one higher than the previous commit ID.				

Considerations

- ◆ Issue the first COMIT after **SINON** with ASGN as commit identifier. Subsequent COMITs can continue to use ASGN, or can update this field to a higher number before issuing the commit. If the number is not higher, the PDM returns an error status. Using ASGN is recommended.
- ◆ After successful execution, the PDM updates this field to contain the number that was used for this commit record on the Task Log File.

length

Description	<i>Required.</i> Points to an area containing the length of the data in the <i>data-area</i> parameter, or a zero.
Format	Binary fullword
Options	<i>zero</i> No user data is written with this commit record. <i>length value</i> This amount of the data area is added to this commit record in the Task Log File.

Considerations

- ◆ *Length* must be greater than or equal to zero.
- ◆ Since records are spread physically across Task Log blocks, there is no theoretical limit to the size of your data area. However, you must consider your program's available memory and your Task Log File capacity.
- ◆ **VSE** When using a central PDM in a cross-address space VSE/AF SP2.1 (XPCC=YES), the maximum length is limited by the CSIPARM MAXPACKET value. Refer to the *SUPRA Server PDM and Directory Administration Guide (OS/390 & VSE)*, P26-2250, for CSIPARM information.

data-area

Description	<i>Required.</i> Points to a field used as the area for any text or memory fields you want to save and store on the Task Log File. The PDM can pass this data back to your task if a task recovery resets to this commit point. To obtain it, issue a RESET after getting an RSTR upon SINON .
Format	Variable length

Considerations

- ◆ The data area should be at least as large as the length specified in the *length* parameter.
- ◆ If the *length* parameter is zero, the PDM does not examine the data nor write it to the Task Log File. However, you must code the *data-area* parameter to complete this parameter list.

end

Description *Required.* Points to a field that delimits the parameter list.

Format END.

General considerations

- ◆ If task logging is not active when you issue the COMIT command, the PDM returns an error status code.
- ◆ Always use a COMIT after any **OPENX** or **CLOX** to avoid file lock problems. A **SINOF** is not sufficient for closing and unlocking in the event that a warm PDM start is needed.

DEL-M

The DEL-M (Delete Primary) command deletes the record identified by the *control-key* parameter. In a multitask operating mode or if task logging is active, you must hold the record to be deleted before you can delete it. The primary record cannot be deleted if there are any related file records linked to it.

DEL-M, status, file, control-key, data-list, data-area, end

status

Description *Required.* Points to a field into which the PDM places a status code indicating the result of the command.

Format 4-byte field

Considerations

- ◆ If the command fails or if the status code indicates some special condition other than failure, your program should include logic to handle and possibly correct the situation. For a list of all status codes, refer to the *SUPRA Server PDM Messages and Codes Reference Manual (RDM/PDM Support for OS/390 & VSE)*, P26-0126.
- ◆ Code your program to handle the following status codes:
 - IMDL Returned if there are any related records in any linkpath associated with the record you are trying to delete. (You need to issue **DELVD** for each related record associated with this primary record before you can issue a DEL-M successfully.)
 - MRNF There is no record in the primary file with the specified control key.
 - HELD When accessing a database record currently being held by another task, the PDM waits until a user-defined time-out counter expires before it reexamines the status of the lock request. If the record is still HELD, the PDM returns the HELD status to the application. The application may elect to terminate the task or reissue the command. If you choose recycle logic for HELD statuses, you should implement an upper limit to the mechanism in the program logic.
 - EMBR As with the HELD status, the PDM may return EMBR when another task is using the database record. Distinguishing the EMBR from the HELD status is the occurrence of a deadly embrace. Deadly embrace occurs when two tasks, already owners of a resource (database record), attempt to gather each other's resources. To handle an EMBR status, restart the logical unit of work. Alternatively, you can introduce a retry mechanism with an upper limit retry count in your program logic.

file

- Description** *Required.* Identifies the primary file to be acted upon. You can define a field containing the name or you can code the actual file name as a literal in the CALL statements.
- Format** 4 alphanumeric characters; first character must be alphabetic
- Consideration** The file must be a primary file.
-

control-key

- Description** *Required.* Points to a field containing the key of the primary record to be processed. The PDM uses this parameter to locate the primary record.
- Format** Variable length as defined on the Directory
-

data-list

- Description** *Required.* This field is not examined; however, it must point to a valid address.
- Consideration** **VSE** When using a cross-address space central PDM in VSE/AF with XPCC=YES, the data area mapped by this *data-list* parameter is limited by the CSIPARM MAXIO value. This governs DML having no *length* parameter. Refer to the *SUPRA Server PDM and Directory Administration Guide (OS/390 & VSE)*, P26-2250, for CSIPARM information.
-

data-area

- Description** *Required.* This field is not examined; however, it must point to a valid address.
-

end

- Description** *Required.* Points to a field that delimits the parameter list.
- Format** END.

General considerations

- ◆ Do not execute the DEL-M command while performing serial reads with the **RDNXT** or **FINDX** commands as this could result in some records being missed and others being read multiple times.
- ◆ When you delete this record, the PDM maintains all populated secondary keys in the specified data file accordingly.

DELVD

The DELVD (Delete Related Direct) command deletes the related record whose RRN is in the *reference* parameter. This removes it from all linkpaths, clears it of data, and indicates that it can be reused. In a multitask operating mode or if task logging is active, you must hold the record to be deleted before you can delete it.

DELVD, *status, file, reference, linkpath, control-key, data-list, data-area, end*

status

Description *Required.* Points to a field into which the PDM places a status code indicating the result of the command.

Format 4-byte field

Considerations

- ◆ If the command fails or if the status code indicates some special condition other than failure, your program should include logic to handle and possibly correct the situation. For a list of all status codes, refer to the *SUPRA Server PDM Messages and Codes Reference Manual (RDM/PDM Support for OS/390 & VSE)*, P26-0126.
- ◆ Code your program to handle the following status codes:
 - MRNF The specified *control-key* parameter value does not exist in the respective primary file.
 - UCTL The specified *control-key* parameter value does not match the control key in the record at RRN.
 - HELD When accessing a database record currently being held by another task, the PDM waits until a user-defined time-out counter expires before it reexamines the status of the lock request. If the record is still HELD, the PDM returns the HELD status to the application. The application may elect to terminate the task or reissue the command. If you choose recycle logic for HELD statuses, you should implement an upper limit to the mechanism in the program logic.
 - EMBR As with the HELD status, the PDM may return EMBR when another task is using the database record. Distinguishing the EMBR from the HELD status is the occurrence of a deadly embrace. Deadly embrace occurs when two tasks, already owners of a resource (database record), attempt to gather each other's resources. To handle an EMBR status, restart the logical unit of work. Alternatively, you can introduce a retry mechanism with an upper limit retry count in your program logic.

file

- Description** *Required.* Identifies the related file to be acted upon. You can define a field containing the name or you can code the actual file name as a literal in the CALL statement.
- Format** 4 alphanumeric characters; first character must be alphabetic
- Consideration** The file must be a related file.

reference

- Description** *Required.* Points to a field identifying the RRN of the related record to be deleted.
- Format** 4 alphanumeric characters or a binary fullword.
- Consideration** When the DELVD command completes, the PDM updates the *reference* field to the RRN of the previous logical record in the chain. However, if the deleted record was logically the first record on the chain, the *reference* field is updated to LKxx instead, where xx are the actual characters from the linkpath name.

linkpath

- Description** *Required.* Points to a field containing the name of the linkpath as defined on the Directory. This parameter indicates which related record chain is being processed.
- Format** *ppppLKxx* where *pppp* identifies the name of an associated primary file, LK is a literal, and xx are the last 2 characters of the linkpath name as defined on the Directory.

control-key

Description *Required.* Points to a field containing the key of the record in the primary file named by the *linkpath* parameter. Many other primary records may need updating as a result of this DELVD command.

Format Variable length as defined on the Directory

Considerations

- ◆ If the control key value does not match the value in the record to be deleted (RRN), the PDM returns an error status code.
- ◆ This parameter specifies the key of the controlling primary record.

data-list

Description *Required.* This field is not examined; however, it must point to a valid address.

Consideration **VSE** When using a cross-address space central PDM in VSE/AF with XPCC=YES, the data area mapped by this *data-list* parameter is limited by the CSIPARM MAXIO value. This governs DML having no *length* parameter. Refer to the *SUPRA Server PDM and Directory Administration Guide (OS/390 & VSE)*, P26-2250, for CSIPARM information.

data-area

Description *Required.* This field is not examined; however, it must point to a valid address.

end

Description *Required.* Points to a field that delimits the parameter list.

Format END.

General considerations

- ◆ If a DELVD is immediately followed by a **READD**, it reads the record logically preceding the deleted record, according to the returned RRN. If a **READV** immediately follows, it reads the record logically following the deleted record.
- ◆ However, if a **READR** immediately follows, it skips the record logically before the deleted record and reads the next preceding record. This last case must be handled carefully if you are attempting to read all records. For example, to process a full reverse delete of the chain, you would code only one READR, as follows: READR, DELVD, **READD**, DELVD, READD, DELVD. A forward delete of the chain would proceed as follows: **READV**, DELVD, READV, DELVD, READV, DELVD.
- ◆ Do not execute the DELVD command while serially reading with the **RDNXT**, **FINDX**, and **READX** commands, as this could result in some records being missed and others being read multiple times.
- ◆ When you delete this record, the PDM maintains all populated secondary keys in the specified data file accordingly.

ENDLG

Use the ENDLG command to terminate the current logical System Log volume and initiate another one. With this command you can end a log volume at a particular point. When you end the log volume, the PDM writes to the database files all the database buffers that have been updated. After the PDM writes the database buffers, it adds an ENDLG command record to the current System Log volume. The PDM then writes all updated System Log buffers to the System Log File. At this point, you can start a new volume.

When ENDLG initiates a new logical volume, it increments the logical volume number by one. After opening the new logical volume, the PDM writes an initialization record, file initialization records, and the task records on the new data set.

ENDLG, *status*, *end*

status

- Description** *Required.* Points to a field into which the PDM places a status code indicating the result of the command.
- Format** 4-byte field
- Consideration** If the command fails or if the status code indicates some special condition other than failure, your program should include logic to handle and possibly correct the situation. For a list of all status codes, refer to the *SUPRA Server PDM Messages and Codes Reference Manual (RDM/PDM Support for OS/390 & VSE)*, P26-0126.

end

- Description** *Required.* Points to a field that delimits the parameter list.
- Format** END.

General considerations

The following considerations apply to all operating systems:

- ◆ If task logging is not active, the ENDLG command quiesces the database; that is, the command keeps any new command from starting and waits for any pending commands to finish before flushing buffers.
- ◆ If task logging is active, ENDLG does not quiesce the database. ENDLG only flushes database buffers. While the PDM closes the System Log File, it allows other commands to proceed. One of these other commands could produce more than one system log record, for example, a before image and a command record. If so, all the records may not appear on the same log volume. The records produced after the System Log File closes appear on another log volume.

OS/390

The following considerations apply to the OS/390 operating system:

- ◆ If the System Log File is a tape, its physical disposition depends on the LABEL and DISP parameters on your DD statement in the JCL describing the System Log File. For the exact results of various combinations of LABEL and DISP parameters, refer to your IBM documentation.
 - ◆ If you specified the CLOSE option in your active environment description, ENDLG closes the System Log File with the CLOSE macro and then reopens it with the OPEN macro. For more information on the OPEN and CLOSE macros, refer to your IBM documentation.
 - ◆ When you close and reopen the same file, you write over the same volumes. Therefore, when you use the Recover function of the DBA Utilities, the file may not contain all the images you need to recover successfully.
 - ◆ If you specified the FEOV option in your active environment description, the following can occur:
 - If the log is on tape, ENDLG starts a new volume for each System Log File group. The PDM issues the FEOV macro to the operating system to change tape volumes.
 - If the log is on disk, ENDLG operates differently depending on whether you specified the switching option in the Directory:
 - ◆ If you specified the switching option, ENDLG switches to the next log file.
 - If you are writing to the last log file in the group and you selected the wrap option, the PDM reuses the first log file.
 - If you elected not to wrap from the last log file to the first, the PDM ignores the ENDLG command and returns a status of *IGN (IGNORE).
 - ◆ If you did not specify the switching option, the command does nothing for that log group. The PDM returns a status of *IGN (IGNORE).
-

VSE

The following considerations apply to the VSE operating system:

- ◆ If you specified the CLOSE option in your active environment description, the ENDLG command closes the System Log File with the CLOSE macro and then reopens it with the OPEN macro. For more information on the OPEN and CLOSE macros, refer to your IBM documentation.
 - ◆ When you close and reopen the same file, you write over the same volumes. Therefore, when you use the Recover function of the DBA Utilities, the file may not contain all the images you need to recover successfully.
 - ◆ If you specified the FEOV option in your active environment description, the following can happen:
 - If the log is on tape, the ENDLG command starts a new volume for each System Log File group. However, the command operates differently depending on which access method you use:
 - ◆ For all BSAM and OUTPUT access methods, the PDM issues the FEOV macro to the operating system to change tape volumes.
 - ◆ For the WORK access method, the PDM closes the file and tells the system operator to mount a new tape volume. When the operator replies that the new volume is mounted, the PDM reopens the file.
 - If the log is on disk, ENDLG operates differently depending on whether you specified the switching option in the Directory:
 - ◆ If you specified the switching option, the command switches to the next specified log file.
 - If you are writing to the last log file in the group and you selected the wrap option, the PDM reuses the first log file.
 - If you elected not to wrap from the last log file to the first, the PDM ignores ENDLG and returns a status of *IGN (IGNORE).
 - ◆ If you did not specify the switching option, the command does nothing for that log group. The PDM returns a status of *IGN (IGNORE).
-

ENDTO

ENDTO terminates a multitask PDM that is operating in central mode. Use the ENDTO command only in a special single-purpose program. Terminating includes verifying shutdown password, checking if any tasks are signed-on, and closing the System Log File, Task Log File, and Statistics File (if active).

An alternative to coding a special application for ENDTO is to use the PDM Termination utility (refer to the *SUPRA Server PDM DBA Utilities User's Guide (OS/390 & VSE)*, P26-6260).

ENDTO, status, option-list, end

status

Description *Required.* Points to a field into which the PDM places a status code indicating the result of the command.

Format 4-byte field

Considerations

- ◆ If the command fails or if the status code indicates some special condition other than failure, your program should include logic to handle and possibly correct the situation. For a list of all status codes, refer to the *SUPRA Server PDM Messages and Codes Reference Manual (RDM/PDM Support for OS/390 & VSE)*, P26-0126.
- ◆ Code your program to handle the following status code:
 - ACTV If you specify the option FORCE=NO, this status code is returned to notify you that an interface still connected to the PDM has an active task.

option-list

Description	<i>Required.</i> Points to a field which defines the options you can use to shut down the multitask central PDM.
Format	<i>option-list,END.</i>
Options	None of the options are required except password if needed. They can be coded in any order. END. must be last.

SHUTDOWN=*shutdown-password*

Description *Conditional.* A field containing the password used to terminate the PDM if required by the Directory.

Format 1–8 alphanumeric characters

Considerations

- ◆ The correct password is required if you defined a shutdown password in your active environment description.
- ◆ You can omit this parameter if no shutdown password is defined in your active environment description. It is ignored if coded.
- ◆ To add or change an existing password, 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.

FORCE = $\begin{cases} \text{NO} \\ \text{YES} \end{cases}$

Description *Optional.* Points to a field which specifies whether to forcibly terminate the PDM.

Default NO

Options NO Do not terminate if any interface has an active task. Return a notification in *status*.

YES Terminate forcibly if there are tasks still signed on, otherwise unforced.

Considerations

- ◆ Code FORCE=NO if all interfaces terminated normally.
- ◆ Code FORCE=YES if an interface abended or was forced to terminate.
- ◆ If forced termination is required and task logging is active, all tasks still signed on to the PDM are **RESET** but not signed off. If task logging is not active, all tasks still signed on to the PDM are signed off.
- ◆ With forced termination, any open files are closed without unlocking them. Otherwise, any open files are closed and unlocked.

DBM=*pdm-name*

- Description** *Optional.* Identifies the PDM to be shut down.
- Default** The PDM name in the CSIPARM file, if specified. If not specified in CSIPARM, the current job name in OS/390 and VSE.
- Format** 1–8 alphanumeric characters
- Consideration** If you code this parameter for a single-task or multitask PDM for batch, it overrides the PDM name in the CSIPARM file (if PDM name is not supplied in the CSIPARM file, this name overrides the current job name).

INTERFACE=*interface-id*

- Description** *Optional.* Points to a field which identifies the interface connected to the central PDM for the ENDTO.
- Default** The interface name in the CSIPARM file, if specified. If not specified in CSIPARM, the current job name in OS/390 and VSE.
- Format** 1–8 alphanumeric characters
- Consideration** If you code this parameter for a single-task or multitask PDM for batch, it overrides the PDM name in the CSIPARM file (if PDM name is not supplied in the CSIPARM file, this name overrides the current job name).

END.

- Description** *Required.* A field that delimits the *option-list*.

end

- Description** *Required.* Points to a field that delimits the parameter list.
- Format** END.

General considerations

- ◆ Using `FORCE =NO` (the default), you cannot shut down the PDM while any applications are still signed on. You will receive an error status code if applications are still connected through any interface.
- ◆ If the PDM is in single-task operating mode, you will receive an error status code if you issue an `ENDTO` command. You shut down a single-task PDM by issuing a `SINOF` command from the (only) application.

FINDX

The FINDX (Find Record) command performs a serial search of a primary or related file, or a serial-sequential search of a related file, for a record that satisfies criteria you defined in the *argument* parameter. A found record is termed a value record in this description.

You can start the search at the beginning of the file, at a specific record location, or at a record identified by a control key. Data items according to your data list are retrieved into the data area. If using a coded data list, record codes omitted from the list are not retrieved.

After each read, the PDM updates the *qualifier* parameter to the next record's location in order to simplify the repeated FINDX commands. You can continue reexecuting the FINDX command until the end of the file is reached.

FINDX, *status*, *file*, *qualifier*, *argument*, *data-list*, *data-area*, *end*

status

Description *Required.* Points to a field into which the PDM places a status code indicating the result of the command.

Format 4-byte field

Considerations

- ◆ If the command fails or if the status code indicates some special condition other than failure, your program should include logic to handle and possibly correct the situation. For a list of all status codes, refer to the *SUPRA Server PDM Messages and Codes Reference Manual (RDM/PDM Support for OS/390 & VSE)*, P26-0126.
- ◆ If the command fails, the content of the *qualifier* parameter is unreliable. Your program should reinitialize it if you want to continue.

- ◆ Code your program to handle the following status codes:
 - MRNF For an equal search on a primary file with *qualifier* set to *KEY=control-key* (or KSDS *KEY=(partial-key)*), the specified control key value does not exist. For a related file with *qualifier* set to "*linkpathKEY=control-key*", the specified control key value does not exist on the primary file.
 - HELD When accessing a database record currently being held by another task, the PDM waits until a user-defined time-out counter expires before it reexamines the status of the lock request. If the record is still HELD, the PDM returns the HELD status to the application. The application may elect to terminate the task or reissue the command. If you choose recycle logic for HELD statuses, you should implement an upper limit to the mechanism in the program logic.
 - EMBR As with the HELD status, the PDM may return EMBR when another task is using the database record. Distinguishing the EMBR from the HELD status is the occurrence of a deadly embrace. Deadly embrace occurs when two tasks, already owners of a resource (database record), attempt to gather each other's resources. To handle an EMBR status, restart the logical unit of work. Alternatively, you can introduce a retry mechanism with an upper limit retry count in your program logic.
 - ENDC End of chain has been reached on this read. The data area has not been updated from the previous contents. For a related file first read, with the *qualifier* set to "*linkpathKEY=control-key*", there are no value records in this linkpath set. If the first 4 bytes of the *qualifier* does not contain NEXT, there are no records at all for this key. You cannot continue without changing the *qualifier*. For other reads using a linkpath *qualifier*, after the PDM sets *qualifier* to NEXT chain-head linkpath, ENDC indicates there were no more value records in the just-completed linkpath set. You can continue to the next set by not changing the *qualifier*.
 - END. End of file, or end of all linkpath sets when using a linkpath, has been reached on this read. The data area has not been updated from the previous contents. For a primary file, no matter what the *qualifier* started with, no (more) value records are found up to end of file. For a related file, if *qualifier* started with BEGN**bbbb**SERIAL or rrrr**bbbb**SERIAL, no (more) value records up to end of file are found. If the *qualifier* is using a linkpath, there are no (more) linkpath sets.

file

- Description** *Required.* Identifies the primary or related file to be acted upon. You can define a field containing the name or you can use the actual file name as a literal in the CALL statement.
- Format** 4 alphanumeric characters; first character must be alphabetic
- Consideration** The file can be either a related or primary file.
-

qualifier

- Description** *Required.* Points to a variable-length field that establishes the starting position and maintains the current position in the file being processed. The size and content of the *qualifier* field depend on the file type (primary or related) and the file access method (BDAM, ESDS or KSDS).
- Format** See “[FINDX qualifier for BDAM or ESDS primary files](#)” on page 129, “[FINDX qualifier for KSDS primary files](#)” on page 130, and “[FINDX qualifier for related files](#)” on page 133 for the different qualifier formats to use with FINDX.

argument

Description *Required.* Specifies your search criteria. The *argument* parameter directs the PDM to build a character string from a file record, using items named in the argument's element-list. It then compares the character string with the string you specify in the argument's value, using the specified operator.

If the comparison meets the condition, the PDM moves the data items you name in the *data-list* parameter to the data area. If the comparison does not meet that condition, the PDM moves no data fields, reads another record automatically (if you are not using equal key qualifier), and applies this comparison again.

Format *element-list,END..operator.valueEND.*

where:

element-list

Description *Required.* Names the data items in each record from which to build a character string for comparing to the value parameter.

Format Variable length consisting of one or more 8-character data item names followed by END.

Considerations

- ◆ Commas separating the element names are optional; the commas must always or never appear in the list. You can code the element list only as follows (with any number of elements):

```
element , element , element , END .
```

or

```
elementelementelementEND .
```

- ◆ For data items you want retrieved into the data area, name them separately in the *data-list* parameter.

END.

Description *Required.* Specifies the element list delimiter.

operator

Description	<i>Required.</i> Specifies the type of comparison to be made between the character string built from the items in the element list and the one you supply in value.
Format	2-character code enclosed in periods
Options	.GT. Greater than .LT. Less than .EQ. Equal to .NE. Not equal to .GE. Greater than or equal to .LE. Less than or equal to

value

Description	<i>Required.</i> Specifies the constant character string to be compared against the constructed string from each record. The element list of the argument maps both strings.
Format	1- <i>n</i> characters according to combined item lengths. No separators are allowed between individual values.

END.

Description *Required.* Specifies the value delimiter.

Considerations

- ◆ The PDM makes all comparisons as though the data were character. That is, there are no considerations for packed, zoned or other data types.
- ◆ **VSE** IN VSE/AF with XPCC=YES, the maximum argument length is limited by the CSIPARM MAXARG value. This length includes *element-list*, *operator*, and *value*.

data-list

Description *Required.* Points to a field containing a list of data items. This list acts as a map of the layout of the data area. Compose this list using data names (physical fields) defined on the Directory.

Format *dataitem1,dataitem2,...dataitemn,END.*

Considerations

- ◆ The commas between the entries are optional and only serve as separators; be consistent whether you use them or not.
- ◆ The data list can include the names of data items, control keys, and record codes. Do not name linkpaths or a root field. Do not list any name twice. If a name is not accepted, the PDM returns an error status code.
- ◆ You can list the data names in any order. They are processed in the order listed, not in the order defined on the Directory. However, for coded records, the record code must be first in the list. If not, unpredictable results occur. See “[Data list parameter keywords](#)” on page 62 for coded data list construction rules.
- ◆ **VSE** When using a cross-address space central PDM in VSE/AF with XPCC=YES, the data area mapped by this *data-list* parameter is limited by the CSIPARM MAXIO value. This governs DML having no *length* parameter. Refer to the [SUPRA Server PDM and Directory Administration Guide \(OS/390 & VSE\)](#), P26-2250, for CSIPARM information.

data-area

Description *Required.* Points to a field to contain the retrieved data from items named in the data list.

Format The structure and characteristics of the data area must conform exactly to the Directory definition of the data items (physical fields) named in the data list.

Considerations

- ◆ The data area must be large enough to hold values for all data items named in the data list. For example, if you have one data item in your data list describing a physical field which is 20 bytes in length, your data area must be at least 20 bytes.
- ◆ The data area and the data list have corresponding fields. The data list holds names, and the data area holds a value for each of those names.

end

Description *Required.* Points to a 4-character field that delimits the parameter list and determines the record-holding function (see “[Record holding](#)” on page 42).

Options	END.	Holds a retrieved record.
	RLSE	Does not hold a retrieved record; can read an uncommitted record.

General considerations

- ◆ The FINDX command performs a logical comparison between the data on the record and the value list in the *argument* parameter.
- ◆ Elements in the *argument* parameter need not be included in the *data-list* and *data-area* parameters.
- ◆ When serial-sequentially reading a related file, the PDM automatically changes from the end of a chain to the beginning of the next on the same linkpath. However, if you want to change linkpaths, you must reinitialize the qualifier to `BEGN` ~~with~~ *new-linkpath*, and reexecute the FINDX command.
- ◆ Do not execute the **ADD-M** and **DEL-M** commands while processing serial FINDX (on the same file). It may miss some records in the serial scan and may read others multiple times.
- ◆ You can execute the **WRITM** command while processing a primary file with the serial FINDX command.

FINDX qualifier for BDAM or ESDS primary files

The qualifier determines where to begin searching for records, either physical beginning of file or at a specified key. If the PDM finds a value record (one that satisfies the argument), it retrieves the record. The PDM also updates the qualifier to the retrieved BDAM or ESDS primary record's location in order to simplify the next FINDX execution in the serial sweep.

$$\left. \begin{array}{l} \text{BEGN} \\ \text{KEY} \left\{ \begin{array}{l} \text{.EQ.} \\ = \end{array} \right\} \text{control - key} \end{array} \right\}$$

Description

On the first FINDX of a series, your program must set the qualifier to one of the two options. For a successful retrieval, the PDM updates the data area and changes qualifier to the binary RRN of the record just read.

To read the next record in the file, your program must not change qualifier before issuing the next FINDX. The PDM finds the next serial record by scanning subsequent RRNs for a nonblank record. If starting with a particular key value, processing proceeds serially from that point. Any RRNs prior to that key are not processed.

When the PDM returns the last value record in the file, the next FINDX returns END. in *status*, sets *qualifier* to BEGN, and does not update the data area. This also occurs if the file is empty on the first read.

You can abandon the retrievals at any point. There is no context involved for BDAM or ESDS.

Format

Use one of the formats following. No blanks are allowed between characters.

BEGN

Description

The PDM starts the value search with the first record (RRN 0/) of the file.

$$\text{KEY} \left\{ \begin{array}{l} \text{.EQ.} \\ = \end{array} \right\} \text{control - key}$$

Description

The PDM starts by locating the record with the specified *control-key* value. If found, and is a value record, the PDM returns it and changes *qualifier* to rrrr. If found, but is not a value record, the PDM continues searching serially from this RRN. If not found, the PDM returns an error status (MRNF).

FINDX qualifier for KSDS primary files

The qualifier determines where to begin searching for records. If the PDM finds a value record (one that satisfies the argument), it retrieves the record. The PDM also updates the qualifier to the next primary record's location in order to simplify the next FINDX execution in the serial sweep. Note that for a KSDS, the serial sweep is also sequential.

BEGN <i>bbbb</i>	}	KEY { <i>.EQ.</i> =} } <i>control - key</i>
KEY.GE. <i>control - key</i>		KEY { <i>.EQ.</i> =} } (<i>partial - key</i>)
KEY.GE. (<i>partial - key</i>)		ENDS <i>cccc</i>

Description

On the first FINDX of a series, your program must set the qualifier to one of the first 5 options. (In other words, you cannot use ENDS for the first read.) For a successful retrieval, the PDM updates the data area and changes qualifier to NEXT*cccc*, where *cccc* points to the context area for this particular sweep. Each new BEGN or KEY sweep creates a new context area. Each area contains the control block for that particular KSDS serial read.

To read the next record, your program must not change *qualifier* before issuing the next FINDX. The PDM finds the next record by scanning forward from this record. If starting with a particular key, processing proceeds serially from this point. Any locations prior to this key are not processed.

When the PDM returns the last value record in the file, the next FINDX returns END. in *status*, sets *qualifier* to BEGN, and does not update the data area. This also occurs if the file is empty on the first read.

If you want to abandon the KSDS retrieval at some point, use the ENDS option.

Format

At least 8 bytes. Use one of the formats following. No blanks are allowed between characters.

BEGNBBBB

Description The PDM begins searching for a value record at the first record of the file (first key).

KEY $\left. \begin{array}{l} \{.EQ.\} \\ \{=\} \end{array} \right\} \text{control - key}$

Description The PDM starts by locating the record with the specified *control-key* value. If not found, the PDM returns an error status (MRNF). (In effect, the serial read cannot begin at a nonexistent point). If found and is a value record, the PDM returns it and changes *qualifier* to NEXTcccc. If found but is not a value record, the PDM continues searching from this point for the first value record.

KEY.GE.control-key

Description The PDM starts by locating the record with the specified *control-key* value. If the record is found and is a value record, the PDM returns it and changes *qualifier* to NEXTcccc. If not found, or if found but is not a value record, the PDM continues searching from this point for the first value record.

KEY $\left. \begin{array}{l} \{.EQ.\} \\ \{=\} \end{array} \right\} (\text{partial - key})$

Description The PDM starts by locating the first record whose *control-key* begins with the specified *partial-key* value. If none are found, the PDM returns an error status (MRNF). (In effect, the serial read cannot begin at a nonexistent point.) If any are found, the PDM returns the first which is a value record and changes *qualifier* to NEXTcccc. If found but none are value records, the PDM continues searching from this point for the first value record.

KEY.GE.(partial-key)

Description The PDM starts by locating the first record whose *control-key* begins with the specified *partial-key* value. If any are found, the PDM returns the first which is a value record and changes *qualifier* to NEXTcccc. If not found, or if found but none are value records, the PDM continues searching from this point for the first value record.

ENDScccc

Description After the PDM has returned at least one record and changed the qualifier to NEXTcccc, you can change the first 4 characters of *qualifier* to this keyword. Do not disturb the cccc pointer. This qualifier signals the end of serial processing before the PDM has returned END. in *status* (end of file). It allows the PDM to free the context area for this particular KSDS sweep before end of file is reached. You should use this if you do not process until END. is returned. Otherwise, ICOR status returns are probable for all tasks.

When your program changes NEXT to ENDS and issues another FINDX, the PDM frees the storage, returns END. in *status*, changes *qualifier* to BEGNbbbb, and does not update the data area. Your program can begin another serial sweep or issue any other DML command.

FINDX qualifier for related files

The qualifier determines where to begin searching for value records, and whether the sweep is to be serial or serial-sequential. If you are processing a coded file using a coded data list, only those records are examined for the search criteria. The PDM updates the qualifier to the retrieved record's binary RRN in order to simplify the next FINDX execution in the sweep. If serial-sequential (linkpath directed), the PDM also maintains the head-of-chain RRN in the qualifier.

```

{ BEGNbbbbSERIAL
  BEGNbbbblinkpath
  linkpathKEY = control - key
}

```

Description

On the first FINDX of a series, your program must set *qualifier* to one of the three options. For a successful retrieval, the PDM updates the data area and changes *qualifier* to the binary RRN of the record just read.

To read the next record in the file, your program must not change *qualifier* before issuing the next FINDX. For a SERIAL read, the PDM finds the next serial record by scanning subsequent RRNs for a nonblank record. For a serial-sequential read (linkpath directed), the PDM finds the physically first chain-head, processes that linkpath set sequentially, finds the next physical chain-head, and so on. If starting with a particular key, processing proceeds from that point. Any chain-heads physically prior to that one are not processed.

When the PDM returns the last value record in the file, or in a linkpath, the next FINDX returns END. status, sets *qualifier* to BEGN, and does not update the data area. This also occurs if the file is empty on the first read.

You can abandon the retrievals at any point. There is no context involved for BDAM or ESDS.

Format

Use one of the formats following. No blanks are allowed except the 4 blanks (**bbbb**) as shown, which are required.

BEGNrrrrSERIAL

Description Use this option to begin serially processing the file. The PDM begins sweeping the file for a value record at the first record of the file. If the file is empty or contains no value records, the PDM leaves *qualifier* unchanged, sets *status* to END., and does not update the data area.

For a successful retrieval, the PDM changes *qualifier* to rrrr**BEGN**SERIAL, where *rrrr* is the RRN of the value record. To search for the next value record, your program must pass this *qualifier*, unchanged, to the next **FINDX** in the series. This directs the PDM to continue serially sweeping the file for the next physical nonblank record.

When the PDM returns the last value record in the file, the next **FINDX** returns END. in *status*, sets *qualifier* to BEGN**BEGN**SERIAL, and does not update the data area.

BEGNbbbb**linkpath**

Description Use this option to begin serial-sequential processing of all chains on the given linkpath name (*ppppLKxx*). The PDM scans the file for the first head-of-chain record on this linkpath. If the file is empty, or contains no value records on the entire linkpath, the PDM sets *status* to END., leaves *qualifier* unchanged, and does not update the data area.

The first value record returned is usually from the first physical head-of-chain on the linkpath. The PDM changes the qualifier to *rrrrhhhhlinkpath*. The *rrrr* is the RRN of the returned record, and *hhhh* is the RRN of the chain-head. However, if there are no value records in the first set, the PDM signals with ENDC (see later in this description).

The first record returned is both a value record and chained on the named linkpath. Your program must pass this qualifier, unchanged, to the next FINDX in the series. This directs the PDM to examine the forward linkpath pointer and return the next logical record in the linkpath set (sequential processing) which meets the argument. For each subsequent returned record, the PDM changes the *rrrr* value to the newly retrieved record, leaving *hhhh* undisturbed.

When the PDM returns the last value record on a chain, the next FINDX returns ENDC in *status*, changes the first 4 bytes of *qualifier* to NEXT, leaving *hhhh* undisturbed, and does not update the data area. This informs your program that all records for this control key have been processed. To continue sweeping for value records on other chains of the same linkpath, your program must pass this qualifier, unchanged, to the next FINDX in the series. With the NEXT qualifier, the PDM continues with a serial search for the next physical chain-head after *hhhh* on this linkpath. When found, the PDM returns the record (if it is a value record), changes NEXT to *rrrr*, and *hhhh* to the new *hhhh* (as on the first read).

If, during linkpath processing, you want to bypass some records before end-of-chain (ENDC), your program may move NEXT to the first 4 bytes of the qualifier (leaving other information unchanged), and issue the FINDX. The PDM skips the remaining records of this set and returns the first value record of the next linkpath set, if any.

When the PDM returns the last value record in the entire linkpath, the next FINDX returns END. in *status*, sets the qualifier to BEGN**bbbb**linkpath, and does not update the data area.

linkpathKEY=control-key

Description Use this option to begin serial-sequential processing at the given chain on the given linkpath name (*ppppLkxx*). The PDM accesses the primary file for a record having the specified *control-key* value. This access is to obtain the RRN of the head-of-chain for this linkpath set. Any linkpath sets whose chain-heads physically precede this one are not processed.

If the key is not found, the PDM returns an error status (MRNF), leaves *qualifier* unchanged, and does not update the data area.

If the key is found, but has no related records on this linkpath, the PDM returns ENDC in *status*, sets *qualifier* to *mmmmmmmlinkpath*, and does not update the data area. The value *mmmmmmmm* denotes the absence of a valid RRN and chain-head to begin the serial-sequential read. If your program uses this qualifier on a subsequent **FINDX** command, the PDM rejects the command with an error status code. You must reinitialize the qualifier if you want to start the read elsewhere.

If the key is found, and has value records on this linkpath, processing proceeds as described under **BEGN***bbbblinkpath*. If there are no value records in this starting set, the PDM signals with ENDC and NEXT. You can continue from this point. It is possible that this set is the last physical chain-head. If so, the PDM signals with END. in *status*.

General consideration

When serial-sequentially reading a related file, the PDM automatically changes from the end of a chain to the beginning of the next on the same linkpath. However, if you want to change linkpaths, you must reinitialize the qualifier to **BEGN***bbbblinkpath*, and reexecute the **FINDX** command.

FREEX

The FREEX command releases any records explicitly held by the task issuing the command for the specified files. This command can be used only if task logging is not active.

FREEX, *status, files, end*

status

Description *Required.* Points to a field into which the PDM places a status code indicating the result of the command.

Format 4 numeric bytes

Consideration If the command fails or if the status code indicates some special condition other than failure, your program should include logic to handle and possibly correct the situation. For a list of all status codes, refer to the *SUPRA Server PDM Messages and Codes Reference Manual (RDM/PDM Support for OS/390 & VSE)*, P26-0126.

files

Description *Required.* Points to a field that identifies the files containing the explicitly held record(s).

Options *file-name-list* Releases all explicitly held records in each file in the file name list.

ALL Releases all explicitly held records in all files.

Consideration File-name-list is a list of file names not separated by commas or blanks.

end

Description *Required.* Points to a field that delimits the parameter list.

Format END.

General consideration

If task logging is active, the PDM returns an error status code.

MARKL

Use the MARKL command to add an arbitrary text record to the System Log File. With this command, you can place any information you want onto the System Log File.

MARKL, *status, length, data-area, end*

status

Description *Required.* Points to a field into which the PDM places a status code indicating the result of the command.

Format 4-byte field

Consideration If the command fails or if the status code indicates some special condition other than failure, your program should include logic to handle and possibly correct the situation. For a list of all status codes, refer to the *SUPRA Server PDM Messages and Codes Reference Manual (RDM/PDM Support for OS/390 & VSE)*, P26-0126.

length

Description *Required.* Points to a field defining the length, in bytes, of the text or data to be written to the System Log File.

Format Binary fullword

Considerations

- ◆ If length is less than zero, an error status code is returned.
- ◆ Since records are spread physically across system log blocks, there is no theoretical limit to the size of your data area. However, you must consider your program's available memory and your System Log File capacity.
- ◆ **VSE** When using a central PDM in a cross-address space VSE/AF SP2.1 (XPCC=YES), the MARKL data area's maximum length is limited by the CSIPARM MAXPACKET value. Refer to the *SUPRA Server PDM and Directory Administration Guide (OS/390 & VSE)*, P26-2250, for CSIPARM information.

data-area

- Description** *Required.* Points to a field containing the data to be written to the System Log File.
- Format** Variable length
- Consideration** The data area must be as large as the length specified in the *length* parameter.
-

end

- Description** *Required.* Points to a field that delimits the parameter list.
- Format** END.

General considerations

- ◆ To execute this command, system logging must be active or you receive an error status code. Task logging may also be active.
- ◆ You can execute this command at any time. A **QUIET** is not required preceding MARKL.

OPENX

The OPENX (Open file) command logically, and if necessary physically, opens one or more files for read-only or shared update for all tasks, or for exclusive update by this task only. Logically opening involves checking and updating each file's control record. Physically opening involves issuing the OPEN command of the host operating system, and checking labels.

You can open primary and related files. You cannot open index files, Directory files, log files, or the Statistics file.

In your environment description, there is an OPENX-OPTION. Its setting determines the outcome of your OPENX as follows:

- ◆ If OPENX-OPTION=IGNORE, the PDM does not carry out the request. You always receive a status indicating success.
- ◆ If OPENX-OPTION=PROCESS, OPENX can change the file mode, lock the file, and issue the operating system OPEN. These effects depend on the current status, lock condition, and mode of the file, and on the type of OPENX you issue.
- ◆ If OPENX-OPTION=CHECK, the PDM does not carry out the request, but the OPENX can return an error status.

You can find which setting is in effect by using a **SHOWX** command with the ENVDPNX data item. The tables at the end of this section present the various situations and results when the option is PROCESS or CHECK.

OPENX,status,realm,end

status

Description *Required.* Points to a field into which the PDM places a status code indicating the overall result of the command.

Format 4-byte field

Considerations

- ◆ If the command fails or if the status code indicates some special condition other than failure, your program should include logic to handle and possibly correct the situation. For a list of all status codes, refer to the *SUPRA Server PDM Messages and Codes Reference Manual (RDM/PDM Support for OS/390 & VSE)*, P26-0126.
- ◆ Code your program to handle the following status codes:
 - OERR The tables at the end of this section present various possible file conditions where an OPENX results in OERR.
 - FNAV File is not available for you to open. Issue a **SHOWX** to see the extended status. This is usually because another task has issued an OPENX or **CLOSX** on the file and has not yet committed.
- ◆ If you use ALL. in the *realm* parameter, the stat field in that parameter is not set. If you use a file-list, the stat field for the file causing the error is set to the same value as this status parameter unless the error is FNTF.

realm

Description *Required.* Points to a list of the REALM= keyword and one or more 12-character entries followed by END. This determines which files to open and in what mode.

Format $\left\{ \text{REALM} = \text{ALL}.\text{modestat}, \right.$
 $\left. \text{filemodestat1}, \text{filemodestat2}, \dots \right\} \text{END.}$

where:

ALL. Opens all user files related to the active environment description. This does not include Directory files.

file Identifies the 4-character name of a primary or related file you want to open.

mode Specifies the mode (read, update) you want for all files or for individual files. Options are:

EUPD Marks the file as locked, and for exclusive update by this task only. Other tasks can read the file but not update it. This task must logically close the file before it or any task can open it for a different mode.

IUPD Marks the file as not locked, and for read only by all tasks. No task can update the file, but any task can lock it by reopening for SUPD or EUPD without an intervening close. (This mode is not valid for KSDS.)

READ Marks the file as not locked, and for read only by all tasks. No task can update the file, and it cannot be locked without any task first closing it and then opening for SUPD or EUPD.

SUPD Marks the file as locked, and for reading and shared updating by all tasks. Any task can logically close the file and open it for a different mode.

stat Provides an area to receive the PDM status code indicating the result of opening an individual file. You initialize these fields to blank. If you use ALL., this field is not set upon completion but the command's *status* field is set.

END. Specifies the delimiter for the *realm* parameter.

Considerations

- ◆ The commas between each “*filemodestat*” entry are optional and only serve as separators; be consistent whether you use them or not. Do not use commas or blanks within the entry.
- ◆ KSDS files cannot be opened for IUPD. Therefore, you should not use a “fileIUPDstat” entry for KSDS in a file list. You also should not use “ALL.IUPDstat” if the active environment description includes KSDS. If you use IUPD, the PDM returns an error status.
- ◆ Index files cannot be named in an entry. The PDM opens them automatically as needed. See General Considerations, which follow.
- ◆ If you code a file list, and a file name is not found, the PDM does not set that entry’s stat field. Instead, it sets the command’s *status* field to FNTF, and the function is immediately terminated. (This measure protects memory from being over-written if the list delimiter (END.) is missing.) To determine which file caused the error, your program can issue a **SHOWX** for extended status.
- ◆ If the PDM processes some of the files successfully, then encounters an error, the PDM does the following:
 - a. If you used a file list, sets the error code in the stat field for the file in error (unless FNTF)
 - b. Sets the error code in the command *status* field
 - c. Reprocesses all involved files to their original state
 - d. Does not process the remaining files

end

Description *Required.* Points to a field that delimits the parameter list.

Format END.

General considerations

- ◆ In a multitask operating mode, the DBA should write a special application to open all files, or use automatic opens, rather than a per task basis. An automatic open for READ or SUPD is available with the file/environment description relationship on the Directory. When used, the PDM opens these files during its initialization.
- ◆ If any fatal errors occur on any file in the realm, the PDM restores all files to their states that existed before executing this OPENX. In a task logging environment, you should issue a **RESET** command after an OPENX command that receives a fatal error. Issue a **COMIT** after a successful OPENX execution.
- ◆ Do not issue both an OPENX and a **CLOSX** within the same logical unit of work.
- ◆ If you use task logging and your task changes a file's mode (even if backed out because of error), the PDM keeps other tasks from changing the file's mode until your task issues a **COMIT** or **RESET**. Any other task that tries to change the file's mode must wait. The wait is according to the environment description delay time for held records (the file lock record is being held). If your task does not issue a **COMIT** or **RESET** to free the file during this time, the other task receives an error status code.
- ◆ If you use task logging, a **SINOF** is not sufficient to commit or reset an OPENX or **CLOSX**.
- ◆ If any task opens a file for EUPD, that task is the only task permitted to update the file or to close it. Therefore, using EUPD as a mode is not recommended in teleprocessing environments.
- ◆ In a single-task environment, a successful open for SUPD means that no other task can update the file because no other task can use the same copy of the PDM. In multitask, any task using the same PDM can update the file.
- ◆ If the user environment description ACCESS MODE parameter has RDNLY at PDM initialization, an OPENX for SUPD or EUPD results in locked files. Since the file is locked, no other task using a different copy of the PDM can update the file. No task sharing this PDM can update the file because the specified access mode is RDNLY, not because the file is locked.

- ◆ If a task opens a file for IUPD, all tasks can read but cannot update that file. However, any task can lock the file with an open for SUPD. If this occurs, all tasks sharing the same copy of the PDM can update the file. This may or may not be contrary to the intentions of the task that originally opened the file.
- ◆ You can open primary or related files but not index files. When your task opens a primary or related file that has index files, the PDM opens its index files if not already open. The same mode is used except for EUPD. The following chart shows this effect:

Primary/related file open mode	Index file open mode
READ	READ
IUPD	IUPD
SUPD	SUPD
EUPD	SUPD

If an index file is already open in a lower mode than your specified mode, it escalates. If already open in a higher mode than your specified mode, it stays the same.

- ◆ OPENX processing and results depend on several factors:
 - The current open or closed state of the file.
 - The current value of the lock record.
 - The current file mode of the file.
 - The mode you request in the *realm* parameter.
 - The environment description PDM access mode (RONLY or UPDATE).
 - The OPENX-OPTION in the active user environment description. If the option is IGNORE, you receive a status of ****, but no open processing occurs. If the option is PROCESS or CHECK, the OPENX is processed as shown in the following tables:

Current file condition				Resultant file condition			
File state	File lock record	File mode	Command attempted	File state	File lock record	File mode	Status code returned
Closed	*	*	OPENX (READ)	Open	-	READ	****
Open	*	READ		-	-	-	****
Open	*	IUPD		-	-	-	****
Open	*	SUPD		-	-	-	****
Open	*	EUPD (S)		-	-	-	****
Open	*	EUPD (D)		-	-	-	****
Closed	*	*	OPENX (IUPD)	Open	-	IUPD	****
Open	*	READ		-	-	-	OERR
Open	*	IUPD		-	-	-	****
Open	*	SUPD		-	-	-	****
Open	*	EUPD (S)		-	-	-	****

Current file condition				Resultant file condition			
File state	File lock record	File mode	Command attempted	File state	File lock record	File mode	Status code returned
Open	*	EUPD (D)		-	-	-	****
Closed	Not Locked	-	OPENX (SUPD)	Open	Locked	SUPD	****
Closed	Locked	-	-	-	-	-	OERR
Open	*	READ		-	-	-	OERR
Open	Not Locked	IUPD		-	Locked	SUPD	****
Open	Locked	IUPD		-	-	-	OERR
Open	*	SUPD		-	-	-	****
Open	-	EUPD (S)		-	-	-	OERR
Open	-	EUPD (D)		-	-	-	OERR
Closed	Not Locked	*	OPENX (EUPD)	Open	Locked	EUPD	****
Closed	Locked	*		-	-	-	OERR
Open	*	READ		-	-	-	OERR
Open	Not Locked	IUPD		-	Locked	EUPD	****
Open	Locked	IUPD		-	-	-	OERR
Open	*	SUPD		-	-	-	OERR
Open	-	EUPD (S)		-	-	-	OERR
Open	-	EUPD (D)		-	-	-	OERR

- * PDM does not check
- PDM does not change
- (S) Same task attempts the open
- (D) Different task attempts the open

Current file condition				Resultant file condition			
File state	File lock record	File mode	Command attempted	File state	File lock record	File mode	Status code returned
Closed	*	*	OPENX (READ)	-	-	-	OERR
Open	*	READ		-	-	-	****
Open	*	SUPD		-	-	-	****
Closed	*	*	OPENX (IUPD)	-	-	-	OERR
Open	*	READ		-	-	-	OERR
Open	*	SUPD		-	-	-	****
Closed	Not Locked	-	OPENX (SUPD)	-	-	-	OERR
Closed	Locked	-		-	-	-	OERR
Open	*	READ		-	-	-	OERR
Open	*	SUPD		-	-	-	****
Closed	Not Locked	-	OPENX (EUPD)	-	-	-	OERR
Closed	Locked	-		-	-	-	OERR
Open	*	READ		-	-	-	OERR
Open	*	SUPD		-	-	-	OERR

- * PDM does not check
- PDM does not change

Concerning the preceding table, when the OPENX-OPTION=CHECK, the file can be only Closed or Open for READ or SUPD. This is according to the file/environment description relationship, where EUPD is not an option.

QMARK

The QMARK command is a combination of **QUIET** and **MARKL** commands. QMARK flushes the I/O buffers (physically applying all pending updates to the database) and adds your data area content to the QUIET record. This command produces only one record on the System Log File.

QMARK, *status, length, data-area, end*

status

Description *Required.* Points to a field into which the PDM places a status code indicating the result of the command.

Format 4-byte field

Consideration If the command fails or if the status code indicates some special condition other than failure, your program should include logic to handle and possibly correct the situation. For a list of all status codes, refer to the *SUPRA Server PDM Messages and Codes Reference Manual (RDM/PDM Support for OS/390 & VSE)*, P26-0126.

length

Description *Required.* Points to a field defining the length, in bytes, of data to be written to the System Log File.

Format Binary fullword

Considerations

- ◆ If the length is less than zero, the PDM returns an error status code.
- ◆ Since records are spread physically across system log blocks, there is no theoretical limit to the size of your data area. However, you must consider your program's available memory and your System Log File capacity.
- ◆ **VSE** When using a central PDM in a cross-address space VSE/AF SP2.1 (XPCC=YES), the QMARK data area's maximum length is limited by the CSIPARM MAXPACKET value. Refer to the *SUPRA Server PDM and Directory Administration Guide (OS/390 & VSE)*, P26-2250, for CSIPARM information.

data-area

Description *Required.* Points to a field containing the data to be written to the System Log File.

Format Variable length

Consideration The data area must be as large as the length specified in the *length* parameter.

end

Description *Required.* Points to a field that delimits the parameter list.

Format END.

General consideration

To execute this command, system logging must be active but task logging must not be active; otherwise you receive an error status code.

QUIET

Use the QUIET command to flush the I/O buffers (physically applying all pending updates to the database) and to write a QUIET record on the System Log File. This synchronizes the physical database with data being manipulated in memory. Synchronization provides a point in the processing where you can restart the system.

QUIET, *status*, *count*, *end*

status

Description *Required.* Points to a field into which the PDM places a status code indicating the result of the command.

Format 4-byte field

Consideration If the command fails or if the status code indicates some special condition other than failure, your program should include logic to handle and possibly correct the situation. For a list of all status codes, refer to the *SUPRA Server PDM Messages and Codes Reference Manual (RDM/PDM Support for OS/390 & VSE)*, P26-0126.

count

Description *Required.* *Count* must be present to complete the parameter list; however, the data is ignored. You must supply a valid address.

end

Description *Required.* Points to a field that delimits the parameter list.

Format END.

General considerations

- ◆ To execute this command, task logging must not be active; otherwise you receive an error status code. You can issue a QUIET command even if system logging is not active; however, there is no record of the QUIET command.
- ◆ For the QUIET command to provide a definite restart point, all PDM tasks must be idle and at a recovery point.
- ◆ You could use a QUIET and **MARKL**, or a **QMARK**, to write limited recovery point information on the System Log File. For instance, you can save task tables or control blocks.

RDNXT

The RDNXT (Read Next) command performs a serial read of a primary or a related file, or a serial-sequential read of a related file. You can start the read at the beginning, at a specific record location, or at a record identified by a control key. Data items according to your data list are retrieved into the data area. If using a coded data list, record codes omitted from the list are not retrieved.

After each read, the PDM updates the *qualifier* parameter to the next record's location, in order to simplify the repeated RDNXT commands. You can continue reexecuting the RDNXT command until the end of the file is reached.

RDNXT processing is essentially the same as **FINDX** except that RDNXT does not use a search argument.

RDNXT, *status*, *file*, *qualifier*, *data-list*, *data-area*, *end*

status

Description *Required.* Points to a field into which the PDM places a status code indicating the result of the command.

Format 4-byte field

Considerations

- ◆ If the command fails or if the status code indicates some special condition other than failure, your program should include logic to handle and possibly correct the situation. For a list of all status codes, refer to the *SUPRA Server PDM Messages and Codes Reference Manual (RDM/PDM Support for OS/390 & VSE)*, P26-0126.
- ◆ If the command fails, the content of the *qualifier* parameter is unreliable. Your program should reinitialize it if you want to continue.

- ◆ Code your program to handle the following status codes:
 - MRNF For an equal search on a primary file with *qualifier* set to *KEY=control-key* (or KSDS *KEY=(partial-key)*), the specified *control-key* value does not exist. For a related file with the *qualifier* set to "*linkpathKEY=control-key*", the specified control key does not exist on the primary file.
 - HELD When accessing a database record currently being held by another task, the PDM waits until a user-defined time-out counter expires before it reexamines the status of the lock request. If the record is still HELD, the PDM returns the HELD status to the application. The application may elect to terminate the task or reissue the command. If you choose recycle logic for HELD statuses, you should implement an upper limit to the mechanism in the program logic.
 - EMBR As with the HELD status, the PDM may return EMBR when another task is using the database record. Distinguishing the EMBR from the HELD status is the occurrence of a deadly embrace. Deadly embrace occurs when two tasks, already owners of a resource (database record), attempt to gather each other's resources. To handle an EMBR status, restart the logical unit of work. Alternatively, you can introduce a retry mechanism with an upper limit retry count in your program logic.
 - ENDC End of chain has been reached on this read. The data area has not been updated from the previous contents. For a related file *first* read, with the *qualifier* set to "*linkpathKEY=control-key*", there are no records in this linkpath set. You cannot continue without changing the *qualifier*. For other reads using a linkpath *qualifier*, after the PDM sets *qualifier* to NEXT chain-head linkpath, ENDC indicates there were no more records in the just-completed linkpath set. You can continue to the next set by not changing the *qualifier*.
 - END. End of file, or end of all linkpath sets when using a linkpath, has been reached on this read. The data area has not been updated from the previous contents. For a primary file, no matter what the *qualifier* started with, no (more) records are found up to end of file. For a related file, if *qualifier* started with *BEGN* or *rrrr*, no (more) records up to end of file are found. If the *qualifier* is using a linkpath, there are no (more) chain-heads.

file

Description *Required.* Identifies the primary or related file to be acted upon. You can define a field containing the name or you can use the actual file name as a literal in the CALL statements.

Format 4 alphanumeric characters; first character must be alphabetic

Consideration The file can be either a primary or related file.

qualifier

Description *Required.* Points to a variable length field that establishes the starting position and maintains the current position in the file being processed. The size and content of the *qualifier* field depend on the file type (primary or related) and the file access method (BDAM or VSAM).

Format See “[RDNXT qualifier for BDAM or ESDS primary files](#)” on page 157, “[RDNXT qualifier for KSDS primary files](#)” on page 158, “[RDNXT qualifier for related files](#)” on page 161 for the different qualifier formats to use with RDNXT.

data-list

Description *Required.* A variable length field containing a list of data items. This list acts as a map of the layout of the data area. Compose this list using data names (physical fields) defined on the Directory.

Format *dataitem1,dataitem2,...dataitemn,END.*

Considerations

- ◆ The commas between the entries are optional and only serve as separators; be consistent whether you use them or not.
- ◆ The data list can include the names of data items, control keys, and record codes. Do not name linkpaths. Do not list any name twice. If a name is not accepted, the PDM returns an error status code.
- ◆ You can list the data names in any order. They are processed in the order listed, not in the order defined on the Directory. However, for coded records, the record code must be first in the list. If not, unpredictable results occur. See “[Data list parameter keywords](#)” on page 62 for coded data list construction rules.
- ◆ **VSE** When using a cross-address space central PDM in VSE/AF with XPCC=YES, the data area length is limited by the CSIPARM MAXIO value. Refer to the [SUPRA Server PDM and Directory Administration Guide \(OS/390 & VSE\)](#), P26-2250, for more information about the MAXIO keyword.

data-area

Description *Required.* Points to a field containing the retrieved data items named in the data list.

Format The structure and characteristics of the data area must conform exactly to the Directory definition of the data items (physical fields) named in the data list.

Considerations

- ◆ The data area must be large enough to hold values for all data items named in the data list. For example, if you have one data item in your data list and it describes a physical field which is 20 bytes in length, your data area must be at least 20 bytes long.
- ◆ The data area and the data list have corresponding fields. The data list holds names, and the data area holds a value for each of those names.

end

Description *Required.* Points to a 4-character field that delimits the parameter list and determines the record-holding function (see “**Record holding**” on page 42).

Options END. Holds a retrieved record.

 RLSE Does not hold a retrieved record; can read an uncommitted record.

General considerations

- ◆ When serial-sequentially reading a related file, the PDM automatically changes from the end of a chain to the next on the same linkpath. However, if you want to change linkpaths, you must reinitialize the qualifier to `BEGNbbbbnew-linkpath`, and reexecute the RDNXT command.
- ◆ Do not execute the `ADD-M` and `DEL-M` commands while processing RDNXT. It may miss some records in the serial scan and may read others multiple times.
- ◆ You can execute the `WRITM` command while processing a primary file with the RDNXT command.

RDNXT qualifier for BDAM or ESDS primary files

The qualifier determines where to begin retrieving records, either physical beginning of file or at a specified key. The PDM updates the qualifier to the retrieved BDAM or ESDS primary record's location in order to simplify the next RDNXT execution in the serial sweep.

$$\left. \begin{array}{l} \text{BEGN} \\ \text{KEY } \left\{ \begin{array}{l} \text{.EQ.} \\ = \end{array} \right\} \text{control - key} \end{array} \right\}$$

Description On the first RDNXT of a series, your program must set the qualifier to one of the two options. For a successful retrieval, the PDM updates the data area and changes *qualifier* to the binary RRN of the record just read.

To read the next record in the file, your program must not change *qualifier* before issuing the next RDNXT. The PDM finds the next serial record by scanning subsequent RRNs for a nonblank record. If starting with a particular key value, processing proceeds serially from that point. Any RRNs prior to that key are not processed.

When the PDM returns the last record in the file, the next RDNXT returns END. in *status*, sets *qualifier* to BEGN, and does not update the data area. This also occurs if the file is empty on the first read.

You can abandon the retrievals at any point. There is no context involved for BDAM or ESDS.

Format Use one of the formats following. No blanks are allowed between characters.

BEGN

Description The PDM starts with the first record (RRN 0/) of the file.

$$\text{KEY } \left\{ \begin{array}{l} \text{.EQ.} \\ = \end{array} \right\} \text{control - key}$$

Description The PDM starts by locating the record with the specified *control-key* value. If found, the PDM changes *qualifier* to *rrrr*. If not found, the PDM returns an error status (MRNF).

RDNXT qualifier for KSDS primary files

The qualifier determines where to begin retrieving records. The PDM updates the qualifier to the retrieved KSDS primary record's location in order to simplify the next RDNXT execution in the serial sweep. Note that for a KSDS, the serial sweep is also sequential.

BEGN*bbbb*

KEY $\left. \begin{array}{l} \{ .EQ. \} \\ \{ = \} \end{array} \right\} \textit{control - key}$

KEY.GE.*control-key*

KEY $\left. \begin{array}{l} \{ .EQ. \} \\ \{ = \} \end{array} \right\} (\textit{partial - key})$

KEY.GE.*(partial-key)*

ENDS*cccc*

Description

On the first RDNXT of a series, your program must set *qualifier* to one of the first 5 options. (In other words, you cannot use ENDS for the first read.) For a successful retrieval, the PDM updates the data area and changes *qualifier* to NEXT*cccc*, where *cccc* points to the context area for this particular sweep. Each new BEGN or KEY sweep creates a new context area. Each area contains the control block for that particular KSDS serial read.

To read the next record, your program must not change *qualifier* before issuing the next RDNXT. The PDM finds the next record by scanning forward from this record. If starting with a particular key, processing proceeds serially from this point. Any locations prior to this key are not processed.

When the PDM returns the last record in the file, the next RDNXT returns END. in *status*, sets *qualifier* to BEGN, and does not update the data area. This also occurs if the file is empty on the first read.

If you want to abandon the KSDS retrievals at some point, use the ENDS option.

Format

At least 8 bytes. Use one of the formats following. No blanks are allowed between characters.

BEGNBBBB

Description The PDM starts with the first record of the file (first key).

KEY $\left. \begin{array}{l} \{ .EQ. \\ = \end{array} \right\} \text{control - key}$

Description The PDM starts by locating the record with the specified control key value. If not found, the PDM returns an error status (MRNF). (In effect, the serial read cannot begin at a nonexistent point.) If found, the PDM returns it and changes *qualifier* to NEXTcccc.

KEY.GE.control-key

Description The PDM starts by locating the record with the specified control key value. If found, the PDM returns it and changes *qualifier* to NEXTcccc. If not found, the PDM returns the record whose control key is the next larger value, and changes *qualifier* to NEXTcccc.

KEY $\left. \begin{array}{l} \{ .EQ. \\ = \end{array} \right\} (\text{partial - key})$

Description The PDM starts by locating the first record whose control key begins with the specified *partial-key* value. If none are found, the PDM returns an error status (MRNF). (In effect, the serial read cannot begin at a nonexistent point.) If any are found, the PDM returns the first one and changes *qualifier* to NEXTcccc.

KEY.GE.(partial-key)

Description The PDM starts by locating the first record whose control key begins with the specified *partial-key* value. If any are found, the PDM returns the first one and changes *qualifier* to NEXTcccc. If none are found, the PDM returns the record whose control key is the next larger value, and changes *qualifier* to NEXTcccc.

ENDScccc

Description After the PDM has returned at least one record and changed the qualifier to NEXTcccc, you can change the first 4 characters of *qualifier* to this keyword. Do not disturb the cccc pointer. This qualifier signals the end of serial processing before the PDM has returned END. in *status*. It allows the PDM to free the context area for this particular KSDS sweep before end of file is reached. You should use this if you do not process until END. Is returned. Otherwise, ICOR status returns are probable for all tasks.

When your program changes NEXT to ENDS and issues another **RDNXT**, the PDM frees the storage, returns END. in *status*, changes *qualifier* to BEGNbbbb, and does not update the data area. Your program can begin another serial sweep or issue any other DML command.

RDNXT qualifier for related files

The qualifier determines where to begin retrieving records, and whether the sweep is to be serial or serial-sequential. If you are processing a coded file using a coded data list, only those records are retrieved. The PDM updates the qualifier to the retrieved record's binary RRN in order to simplify the next RDNXT execution in the sweep. If serial-sequential (linkpath directed), the PDM also maintains the head-of-chain RRN in the qualifier.

BEGNbbbb**SERIAL**

BEGNbbbb**linkpath**

linkpathKEY=control-key

Description

On the first RDNXT of a series, your program must set *qualifier* to one of the three options. For a successful retrieval, the PDM updates the data area and changes *qualifier* to the binary RRN of the record just read.

To read the next record in the file, your program must not change *qualifier* before issuing the next RDNXT. For a SERIAL read, the PDM finds the next serial record by scanning subsequent RRNs for a nonblank record. For a serial-sequential read (linkpath directed), the PDM finds the physically first chain-head, processes that linkpath set sequentially, finds the next physical chain-head, and so on. If starting with a particular key, processing proceeds from that point. Any chain-heads physically prior to that one are not processed.

When the PDM returns the last record in the file, or in a linkpath, the next RDNXT returns END. in *status*, sets *qualifier* to BEGN, and does not update the data area. This also occurs if the file is empty on the first read.

You can abandon the retrievals at any point. There is no context involved for BDAM or ESDS.

Format

Use one of the formats following. No blanks are allowed except the 4 blanks (**bbbb**) shown, which are required.

BEGN**rrrr**SERIAL

Description Use this option to begin serially processing the file. The PDM starts with the first physical record. If the file is empty, the PDM sets *status* to END., leaves *qualifier* unchanged, and does not update the data area.

For a successful retrieval, the PDM changes *qualifier* to *rrrr***rrrr**SERIAL, where *rrrr* is the RRN of the returned record.

To read the next record, your program must pass this qualifier, unchanged, to the next **RDNXT** in the series. This directs the PDM to continue serially sweeping the file for the next physical nonblank record.

When the PDM returns the last allocated record in the file, the next **RDNXT** returns END. in *status*, sets *qualifier* to BEGN**rrrr**SERIAL, and does not update the data area.

BEGNBBBBlinkpath

Description Use this option to begin serial-sequential processing of all chains on the given linkpath name (*ppppLKxx*). The PDM scans the file for the first head-of-chain record on this linkpath. If the file is empty, or contains no records on the named linkpath, the PDM sets *status* to END., leaves *qualifier* unchanged, and does not update the data area.

The first record returned is the first physical head-of-chain on the linkpath. The PDM changes the qualifier to *rrrrhhhhlinkpath*. The *rrrr* is the RRN of the returned record, and *hhhh* is the RRN of the chain-head.

Your program must pass this qualifier, unchanged, to the next **RDNXT** in the series. This directs the PDM to examine the forward linkpath pointer and return the next logical record in the linkpath set (sequential processing). For each subsequent returned record, the PDM changes the *rrrr* value to the newly retrieved record, leaving *hhhh* undisturbed.

When the PDM returns the last record on a chain, the next **RDNXT** returns ENDC in *status*, changes the first 4 bytes of the qualifier to NEXT, leaving *hhhh* undisturbed, and does not update the data area. This informs your program that all records for this control key have been processed. To continue sweeping the linkpath, your program must pass this qualifier, unchanged, to the next **RDNXT** in the series. With the NEXT qualifier, the PDM continues with a serial search for the next physical chain-head after *hhhh* on this linkpath. When found, the PDM returns the record, changes NEXT to *rrrr*, and *hhhh* to the new *hhhh* (as on the first read).

If, during linkpath processing, you want to bypass some records before end-of-chain (ENDC), your program may move NEXT to the first 4 bytes of the qualifier (leaving other information unchanged), and issue the **RDNXT**. The PDM skips the remaining records of this set and returns the first record of the next linkpath set.

When the PDM returns the last record of the last chain in the linkpath, the next **RDNXT** returns END. in *status*, sets the qualifier to **BEGNBBBBlinkpath**, and does not update the data area.

linkpathKEY=control-key

Description Use this option to begin serial-sequential processing at the given chain on the given linkpath name (*ppppLKxx*). The PDM accesses the primary file for a record having the specified *control-key* value. This access is to obtain the RRN of the head-of-chain for this linkpath set. Any linkpath sets whose chain-heads physically precede this one are not processed.

If the key is not found, the PDM returns an error status (MRNF), leaves *qualifier* unchanged, and does not update the data area.

If the key is found, but has no related records on this linkpath, the PDM returns ENDC in *status*, sets *qualifier* to *mmmmmmmlinkpath*, and does not update the data area. The value *mmmmmmmm* denotes the absence of a valid RRN and chain-head to begin the serial-sequential read. If your program uses this *qualifier* on a subsequent **RDNXT** command, the PDM rejects the command with an error status code. You must reinitialize the *qualifier* if you want to start the read elsewhere.

If the key is found, and has related records on this linkpath, processing proceeds as detailed under **BEGN***bbbblinkpath*.

General consideration

When serial-sequentially reading a related file, the PDM automatically changes from the end of a chain to the next on the same linkpath. However, if you want to change linkpaths, you must reinitialize the *qualifier* to **BEGN***bbbbnew-linkpath*, and reexecute the **RDNXT** command.

READD

The READD (Read Direct) command reads the related record specified by the *reference* parameter. No other related record is accessed for this read. Your program must have issued another DML command before this READD to determine the RRN to use.

READD, *status, file, reference, linkpath, control-key, data-list, data-area, end*

status

Description *Required.* Points to a field into which the PDM places a status code indicating the result of the command.

Format 4-byte field

Considerations

- ◆ If the command fails or if the status code indicates some special condition other than failure, your program should include logic to handle and possibly correct the situation. For a list of all status codes, refer to the *SUPRA Server PDM Messages and Codes Reference Manual (RDM/PDM Support for OS/390 & VSE)*, P26-0126.
- ◆ Code your program to handle the following status codes:
 - HELD When accessing a database record currently being held by another task, the PDM waits until a user-defined time-out counter expires before it reexamines the status of the lock request. If the record is still HELD, the PDM returns the HELD status to the application. The application may elect to terminate the task or reissue the command. If you choose recycle logic for HELD statuses, you should implement an upper limit to the mechanism in the program logic.
 - EMBR As with the HELD status, the PDM may return EMBR when another task is using the database record. Distinguishing the EMBR from the HELD status is the occurrence of a deadly embrace. Deadly embrace occurs when two tasks, already owners of a resource (database record), attempt to gather each other's resources. To handle an EMBR status, restart the logical unit of work. Alternatively, you can introduce a retry mechanism with an upper limit retry count in your program logic.

file

- Description** *Required.* Identifies the related file to be acted upon. You can define a field containing the name or you can use the actual file name as a literal in the CALL statement.
- Format** 4 alphanumeric characters; first character must be alphabetic
- Consideration** The file must be a related file, or the PDM returns an error status code.
-

reference

- Description** *Required.* Points to a 4-character field containing the RRN of the related record to read.
- Format** 4 alphanumeric characters or a binary fullword
- Consideration** When the READD command completes successfully, the *reference* parameter still contains the RRN of the retrieved record. That is, the PDM does not change *reference*.
-

linkpath

- Description** *Required.* This parameter must be present to complete the parameter list and must contain a valid address; however, the data to which it points is not examined.
-

control-key

- Description** *Required.* This parameter must be present to complete the parameter list and must contain a valid address; however, the data to which it points is not examined.

data-list

Description *Required.* Points to a field containing a list of data items. This list acts as a map of the layout of the data area. Compose this list using data names (physical fields) defined on the Directory.

Format *dataitem1,dataitem2,...dataitemn,END.*

Considerations

- ◆ The commas between the entries are optional and only serve as separators; be consistent whether you use them or not.
- ◆ The data list can include the names of data items, control keys, and record codes. Do not name linkpaths. Do not list any name twice. If a name is not accepted, the PDM returns an error status code.
- ◆ You can list the data names in any order. They are processed in the order listed, not in the order defined on the Directory. However, for coded records, the record code must be first in the list. If not, unpredictable results occur. See “[Data list parameter keywords](#)” on page 62 for coded data list construction rules.
- ◆ **VSE** When using a cross-address space central PDM in VSE/AF with XPCC=YES, the data area mapped by this *data-list* parameter is limited by the CSIPARM MAXIO value. This governs DML having no *length* parameter. Refer to the [SUPRA Server PDM and Directory Administration Guide \(OS/390 & VSE\)](#), P26-2250, for CSIPARM information.

data-area

- Description** *Required.* Points to a field to receive the data items named in the data list.
- Format** The structure and characteristics of the data area must conform exactly to the Directory definition of the data items (physical fields) named in the *data-list*.

Considerations

- ◆ The data area must be large enough to hold values for all data items named in the data list. For example, if you have one data item in your data list and it describes a physical field which is 20 bytes in length, your data area must be at least 20 bytes long.
- ◆ The data area and the data list have corresponding fields. The data list holds names, and the data area holds a value for each of those names.

end

- Description** *Required.* Points to a 4-character field that delimits the parameter list and determines the record holding function (see “**Record holding**” on page 42).
- Options** END. Holds the retrieved record.
- RLSE Does not hold the retrieved record; can read an uncommitted record.

General considerations

- ◆ If your program executes a **READR** or a **READV** immediately before the **READD**, the **READD** (using the same *reference* field) reads the same record again.
- ◆ If your program issues a **READV** or **READR** following a **READD**, the **PDM** reads the next record (if **READV**) or the preceding record (if **READR**).
- ◆ If your program executes a **DELVD** immediately after a **READD**, the **PDM** deletes the related record, and the *reference* parameter then points to the preceding record. See the **DELVD** command for further **DELVD** considerations.

READM

The READM (Read Primary) command reads the record specified by the control key and places the record into the data area according to the *data-list* parameter.

READM, status, file, control-key, data-list, data-area, end

status

Description *Required.* Points to a field into which the PDM places a status code indicating the result of the command.

Format 4-byte field

Considerations

- ◆ If the command fails or if the status code indicates some special condition other than failure, your program should include logic to handle and possibly correct the situation. For a list of all status codes, refer to the [SUPRA Server PDM Messages and Codes Reference Manual \(RDM/PDM Support for OS/390 & VSE\)](#), P26-0126.
- ◆ Code your program to handle the following status codes:
 - MRNF The specified *control-key* parameter value does not exist on the file.
 - HELD When accessing a database record currently being held by another task, the PDM waits until a user-defined time-out counter expires before it reexamines the status of the lock request. If the record is still HELD, the PDM returns the HELD status to the application. The application may elect to terminate the task or reissue the command. If you choose recycle logic for HELD statuses, you should implement an upper limit to the mechanism in the program logic.
 - EMBR As with the HELD status, the PDM may return EMBR when another task is using the database record. Distinguishing the EMBR from the HELD status is the occurrence of a deadly embrace. Deadly embrace occurs when two tasks, already owners of a resource (database record), attempt to gather each other's resources. To handle an EMBR status, restart the logical unit of work. Alternatively, you can introduce a retry mechanism with an upper limit retry count in your program logic.

file

- Description** *Required.* Identifies the primary file to be acted upon. You can define a field containing the name or you can use the actual file name as a literal in the CALL statement.
- Format** 4 alphanumeric characters; first character must be alphabetic
- Consideration** The file must be a primary file, or the PDM returns an error status code.
-

control-key

- Description** *Required.* Points to a field containing the key of the primary record to be processed. The PDM uses this key to identify the primary record being read.
- Format** Variable length as defined on the Directory
-

data-list

- Description** *Required.* A variable length field containing a list of data items. This list acts as a map of the layout of the data area. Compose this list using data names (physical fields) defined on the Directory.
- Format** *dataitem1,dataitem2,...dataitemn,END.*

Considerations

- ◆ The commas between the entries are optional and only serve as separators; be consistent whether you use them or not.
- ◆ The data list can include the names of data items and control keys. Do not name linkpaths or the root field. Do not list any name twice. If a name is not accepted, the PDM returns an error status code.
- ◆ You can list the data names in any order. They are processed in the order listed, not in the order defined on the Directory.
- ◆ **VSE** When using a cross-address space central PDM in VSE/AF with XPCC=YES, the data area mapped by this *data-list* parameter is limited by the CSIPARM MAXIO value. This governs DML having no *length* parameter. Refer to the *SUPRA Server PDM and Directory Administration Guide (OS/390 & VSE)*, P26-2250, for CSIPARM information.

data-area

Description *Required.* Points to a field to receive the data from the data items named in the data list.

Format The structure and characteristics of the data area must conform exactly to the Directory definition of the data items (physical fields) named in the data list.

Considerations

- ◆ The data area must be large enough to hold values for all data items named in the data list. For example, if you have one data item which is 20 bytes long, your data area must be at least 20 bytes long.
- ◆ The data area and the data list have corresponding fields. The data list holds names, and the data area holds a value for each of those names.

end

Description *Required.* Points to a 4-character field that delimits the parameter list and determines the record holding function (see “[Record holding](#)” on page 42).

Options	END.	Holds a retrieved record.
	RLSE	Does not hold a retrieved record; can read an uncommitted record.

READR

The READR (Read Reverse) command reads a related record. Use it to follow a chain backward along a specified linkpath. You start the first read with LKxx in the *reference* parameter. The PDM accesses the control key primary record to find the last logical record in this chain.

After the READR, the PDM updates the reference to contain the located record RRN. Your program can then pass that information as reference value to the next READR. The PDM uses it to find the preceding logical record on the chain, and so on until head-of-chain is read.

READR, *status, file, reference, linkpath, control-key, data-list, data-area, end*

status

Description *Required.* Points to a field into which the PDM places a status code indicating the result of the command.

Format 4-byte field

Considerations

- ◆ If the command fails or if the status code indicates some special condition other than failure, your program should include logic to handle and possibly correct the situation. For a list of all status codes, refer to the *SUPRA Server PDM Messages and Codes Reference Manual (RDM/PDM Support for OS/390 & VSE)*, P26-0126.
- ◆ Code your program to handle the following status code:
 - MRNF The specified *control-key* parameter value does not exist in the respective primary file.
 - HELD When accessing a database record currently being held by another task, the PDM waits until a user-defined time-out counter expires before it reexamines the status of the lock request. If the record is still HELD, the PDM returns the HELD status to the application. The application may elect to terminate the task or reissue the command. If you choose recycle logic for HELD statuses, you should implement an upper limit to the mechanism in the program logic.
 - EMBR As with the HELD status, the PDM may return EMBR when another task is using the database record. Distinguishing the EMBR from the HELD status is the occurrence of a deadly embrace. Deadly embrace occurs when two tasks, already owners of a resource (database record), attempt to gather each other's resources. To handle an EMBR status, restart the logical unit of work. Alternatively, you can introduce a retry mechanism with an upper limit retry count in your program logic.
- ◆ You receive **** in *status* even when the *reference* parameter is updated to END., signifying the last record was processed by the previous read.

file

- Description** *Required.* Identifies the related file to be acted upon. You can define a field containing the name or you can use the actual file name as a literal in the CALL statement.
- Format** 4 alphanumeric characters; first character must be alphabetic
- Consideration** The file must be a related file, or the PDM returns an error status code.
-

reference

- Description** *Required.* Points to a field identifying the position in a related record chain. You use the *reference* parameter by placing a certain value in this field to tell the PDM which record to begin with. The PDM returns a certain value to inform you which related record was processed.
- Format** 4-byte field
- Options**
- | | |
|------|---|
| LKxx | You set this value to the same as the last 4 characters of the <i>linkpath</i> parameter. Substitute the actual characters for xx. This value directs the PDM, for READR, to read the last logical record in the chain for this control key. |
| rrrr | This is set by the PDM to identify the RRN of the record just read. When you issue a READR with <i>reference</i> still set to <i>rrrr</i> from the previous READR, the PDM uses this RRN to locate the preceding logical record in the chain. |

Considerations

- ◆ If the *reference* parameter contains an RRN, the READR uses the back pointer in that record to find the RRN of the preceding record. The preceding record is retrieved and its RRN is placed in the *reference* field. Therefore, *reference* always contains the RRN of the record just read.
- ◆ The PDM places the keyword END. in the *reference* field if you attempt to go beyond the end of the chain. This signifies that the previous read returned the first logical record in the chain (head-of-chain). You can reinitialize the parameters to begin another chain process.
- ◆ If the PDM sets END. for a first READR execution with LKxx, the chain is empty.
- ◆ Since END. is not valid as input reference for READR, your program must change this value before it issues another READR. You must, therefore, include logic in your program to test the *reference* field for END. after each command to detect the logical end of the search (beginning of the chain).

linkpath

Description *Required.* Points to a field containing the name of the linkpath as defined on the Directory.

Format *ppppLKxx* *pppp* identifies the name of an associated primary file, LK is a constant and *xx* are the last 2 characters of the linkpath name as defined on the Directory.

Consideration If you specify an invalid linkpath, the PDM returns an error status code.

control-key

Description *Required.* Points to a field containing the key of the primary record controlling the chain. The PDM uses this key to link from a primary record to a related record.

Format Variable length as defined on the Directory

data-list

Description *Required.* Points to a field containing a list of data items. This list acts as a map of the layout of the data area. Compose this list using data names (physical fields) defined on the Directory.

Format *dataitem1,dataitem2,...dataitemn,END.*

Considerations

- ◆ The commas between the entries are optional and serve only as separators; be consistent whether you use them or not.
- ◆ The data list can include the names of data items, control keys, and record codes. Do not name linkpaths. Do not list any name twice. If a name is not accepted, the PDM returns an error status code.
- ◆ You can list the data names in any order. They are processed in the order listed, not in the order defined on the Directory. However, for coded records, the record code must be first in the list. If not, unpredictable results occur. See “[Data list parameter keywords](#)” on page 62 for coded data list construction rules.
- ◆ **VSE** When using a cross-address space central PDM in VSE/AF with XPCC=YES, the data area mapped by this *data-list* parameter is limited by the CSIPARM MAXIO value. This governs DML having no *length* parameter. Refer to the [SUPRA Server PDM and Directory Administration Guide \(OS/390 & VSE\)](#), P26-2250, for CSIPARM information.

data-area

Description *Required.* Points to a field to be used as an output area for the data items named in the data list.

Format The structure and characteristics of the data area must conform exactly to the Directory definition of the data items (physical fields) named in the data list.

Considerations

- ◆ The data area must be large enough to hold values for all data items named in the data list. For example, if you have one data item which is 20 bytes long, your data area must be at least 20 bytes long.
- ◆ The data area and the data list have corresponding fields. The data list holds names, and the data area holds a value for each of those names.

end

Description *Required.* Points to a 4-character field that delimits the parameter list and determines the record holding function (see “[Record holding](#)” on page 42).

Options	END.	Holds a retrieved record.
	RLSE	Does not hold a retrieved record; can read an uncommitted record.

General considerations

To begin processing a chain in reverse direction, your program initializes the parameters as follows:

- ◆ Sets the *control-key* parameter to the value you want to process
- ◆ Sets the *file* parameter to the name of the related file containing the chain
- ◆ Sets the *linkpath* parameter to the name of the linkpath relating the primary and related files
- ◆ Sets the *reference* parameter to LKxx, where xx are the actual characters

Your program then executes READR and the following occurs:

- ◆ The PDM recognizes LKxx as a request to read the last record in the chain.
- ◆ The PDM uses the *linkpath* parameter to identify a primary file. It verifies in the Directory that the linkpath relates that primary file and the related file named in the *file* parameter.
- ◆ The PDM uses the control key to read a record from that primary file.
- ◆ The PDM uses the *linkpath* field in that primary record to read the last record on the chain from the related file.
- ◆ The PDM uses the *data-list* parameter to move fields from the related record into the data area.
- ◆ The PDM changes the value of the *reference* parameter to the RRN of the record just read, and returns control to your program.

To read the preceding logical record in the chain, your program can issue another READR with the same parameter list with undisturbed values. The PDM uses the RRN in *reference* to locate the preceding logical record and return its RRN and data.

READV

The READV (Read Forward) command reads a related record. Use it to follow a chain forward along a specified linkpath. You start the first read with LKxx in the *reference* parameter. The PDM accesses the control key primary record to find the first logical record in this chain. After the READV, the PDM updates the *reference* to contain the located record RRN. Your program can then pass that information as *reference* value to the next READV. The PDM uses it to find the next logical record on the chain, and so on until end-of-chain is read.

READV, *status, file, reference, linkpath, control-key, data-list, data-area, end*

status

Description *Required.* Points to a field into which the PDM places a status code indicating the result of the command.

Format 4-byte field

Considerations

- ◆ If the command fails or if the status code indicates some special condition other than failure, your program should include logic to handle and possibly correct the situation. For a list of all status codes, refer to the *SUPRA Server PDM Messages and Codes Reference Manual (RDM/PDM Support for OS/390 & VSE)*, P26-0126.
- ◆ Code your program to handle the following status code:
 - MRNF The specified *control-key* parameter value does not exist in the respective primary file.
 - HELD When accessing a database record currently being held by another task, the PDM waits until a user-defined time-out counter expires before it reexamines the status of the lock request. If the record is still HELD, the PDM returns the HELD status to the application. The application may elect to terminate the task or reissue the command. If you choose recycle logic for HELD statuses, you should implement an upper limit to the mechanism in the program logic.
 - EMBR As with the HELD status, the PDM may return EMBR when another task is using the database record. Distinguishing the EMBR from the HELD status is the occurrence of a deadly embrace. Deadly embrace occurs when two tasks, already owners of a resource (database record), attempt to gather each other's resources. To handle an EMBR status, restart the logical unit of work. Alternatively, you can introduce a retry mechanism with an upper limit retry count in your program logic.
- ◆ You receive **** in *status* even when the *reference* parameter is updated to END., signifying the last record was processed by the previous read.

file

Description *Required.* Identifies the related file to be acted upon. You can define a field containing the name or you can use the actual file name as a literal in the CALL statement.

Format 4 alphanumeric characters; first character must be alphabetic

Consideration The file must be a related file or the PDM returns an error status code.

reference

Description	<i>Required.</i> Points to a field identifying the current position in a related record chain. You use the <i>reference</i> parameter by placing a certain value in this field to tell the PDM which record to begin with. The PDM returns a certain value to inform you which related record was processed.	
Format	4-byte field	
Options	LKxx	You set this value to the same as the last 4 characters of the <i>linkpath</i> parameter. Substitute the actual characters for xx. This value directs the PDM, for READV, to read the first logical record in the chain for this control key.
	rrrr	This is set by the PDM to identify the RRN of the record just read. When you issue a READR with <i>reference</i> still set to <i>rrrr</i> from the previous READR, the PDM uses this RRN to locate the next logical record in the chain.

Considerations

- ◆ If the *reference* parameter contains an RRN, the READV uses the forward pointer in that record to find the RRN of the next record. The next record is retrieved and its RRN is placed in the *reference* field. Therefore, *reference* always contains the RRN of the record just read.
- ◆ The PDM places the keyword END. in the *reference* field if you attempt to go beyond the end of the chain. This signifies that the previous read returned the last logical record in the chain (end-of-chain). You can reinitialize the parameters to begin another chain process.
- ◆ If the PDM sets END. for a first READV execution with LKxx, the chain is empty.
- ◆ Since END. is not valid as input *reference* for READV, your program must change this value before it issues another READV. You must, therefore, include logic in your program to test the *reference* field for END. after each command to detect the logical end of the search (end of the chain).

linkpath

Description *Required.* Points to a field containing the name of the linkpath as defined on the Directory.

Format *ppppLKxx* *pppp* identifies the name of an associated primary file, LK is a constant and *xx* are the last 2 characters of the linkpath name as defined on the Directory.

Consideration If you specify an invalid linkpath, the PDM returns an error status code.

control-key

Description *Required.* Points to a field containing the key of the primary record controlling the chain. The PDM uses this key to link from a primary record to a related record.

Format Variable length as defined on the Directory

data-list

Description *Required.* Points to a field containing a list of data items. This list acts as a map of the layout of the data area. Compose this list using data names (physical fields) defined on the Directory.

Format *dataitem1,dataitem2,...dataitemn,END.*

Considerations

- ◆ The commas between the entries are optional and only serve as separators; be consistent whether you use them or not.
- ◆ The data list can include the names of data items, control keys, and record codes. Do not name linkpaths. Do not list any name twice. If a name is not accepted, the PDM returns an error status code.
- ◆ You can list the data names in any order. They are processed in the order listed, not in the order defined on the Directory. However, for coded records, the record code must be first in the list. If not, unpredictable results occur. See “[Data list parameter keywords](#)” on page 62 for coded data list construction rules.
- ◆ **VSE** When using a cross-address space central PDM in VSE/AF with XPCC=YES, the data area mapped by this *data-list* parameter is limited by the CSIPARM MAXIO value. This governs DML having no *length* parameter. Refer to the [SUPRA Server PDM and Directory Administration Guide \(OS/390 & VSE\)](#), P26-2250, for CSIPARM information.

data-area

Description *Required.* Points to a field to be used as an input/output area for the data items named in the data list.

Format The structure and characteristics of the data area must conform exactly to the Directory definition of the data items (physical fields) named in the data list.

Considerations

- ◆ The data area must be large enough to hold values for all data items named in the data list. For example, if you have one data item which is 20 bytes long, your data area must be at least 20 bytes long.
- ◆ The data area and the data list have corresponding fields. The data list holds names, and the data area holds a value for each of those names.

end

Description *Required.* Points to a 4-character field that delimits the parameter list and determines the record holding function (see “[Record holding](#)” on page 42).

Options	END.	Holds a retrieved record.
	RLSE	Does not hold a retrieved record; can read an uncommitted record.

General considerations

To begin processing a chain in forward direction, your program initializes the parameters as follows:

- ◆ Sets the *control-key* parameter to the value you want to process
- ◆ Sets the *file* parameter to the name of the related file containing the chain
- ◆ Sets the *linkpath* parameter to the name of the linkpath relating the primary and related files
- ◆ Sets the *reference* parameter to LKxx, where xx are the actual characters

Your program then executes READV and the following occurs:

- ◆ The PDM recognizes LKxx as a request to read the first logical record in the chain.
- ◆ The PDM uses the *linkpath* parameter to identify a primary file. It verifies in the
- ◆ Directory that the linkpath relates that primary file and the related file named in the *file* parameter.
- ◆ The PDM uses the control key to read that record from the primary file.
- ◆ The PDM uses the *linkpath* field in that primary record to read the first logical record on the chain from the related file.
- ◆ The PDM uses the *data-list* parameter to move fields from the related record into data area.
- ◆ The PDM changes the value of the *reference* parameter to the RRN of the record just read, and returns control to your program.

To read the next logical record in the chain, your program can issue another READV with the same parameter list with undisturbed values. The PDM uses the RRN in *reference* to locate the next logical record and return its RRN and data.

READX

The READX command accesses primary or related file record(s) using a secondary key from an index that exists for the file. You can start the read at the beginning or end of the index file, or at a generic or specific secondary key value. Data items according to your data list are retrieved into the data area. If using a coded data list, record codes omitted from the list are not retrieved. After each read, the PDM updates the *qualifier* parameter to the next record's location, and updates the internal context area, in order to simplify the repeated READX commands. You can continue reexecuting the READX command until the end of the file is reached.



This command causes the PDM to create a context area for use with repeated reads. It is important that you control the release of this context area by checking the qualifier's *qqqq* content.

READX,*status,file,option-list,qualifier,data-area-len,data-list,data-area,end*

status

Description *Required.* Points to a field into which the PDM places a status code indicating the result of the command.

Format 4-byte field

Considerations

- ◆ If the command fails or if the status code indicates some special condition other than failure, your program should include logic to handle and possibly correct the situation. For a list of all status codes, refer to the *SUPRA Server PDM Messages and Codes Reference Manual (RDM/PDM Support for OS/390 & VSE)*, P26-0126.

◆ Code your program to handle the following warning status codes:

- *IPO** The requested key was not located at the expected offset in the index tables, or if at least one internal reference point (RRN) was not the same on this iteration as on the last. Check the returned record to make sure it is not a duplicate, or restart the function to make sure data was not bypassed. This status code is also returned if the index block containing the requested key on the last call no longer exists and an index restart was required. In this case, the returned data is correct. Use **SHOWX** for extended status to determine which event occurred.
- *NXT** Returned if the requested key is not on the index, so the next best key is returned.
- *PON** Returned if index positioning was off and the next key was returned, or if repositioning took place but there were no more records that matched the key.
- HELD** When accessing a database record currently being held by another task, the PDM waits until a user-defined time-out counter expires before it reexamines the status of the lock request. If the record is still HELD, the PDM returns the HELD status to the application. The application may elect to terminate the task or reissue the command. If you choose recycle logic for HELD statuses, you should implement an upper limit to the mechanism in the program logic.
- EMBR** As with the HELD status, the PDM may return EMBR when another task is using the database record. Distinguishing the EMBR from the HELD status is the occurrence of a deadly embrace. Deadly embrace occurs when two tasks, already owners of a resource (database record), attempt to gather each other's resources. To handle an EMBR status, restart the logical unit of work. Alternatively, you can introduce a retry mechanism with an upper limit retry count in your program logic.
- LOST** The requested key was not located at the expected offset in the index tables, or if at least one internal reference point (RRN) was not the same on this iteration as on the last. Check the returned record to make sure it is not a duplicate, or restart the function to make sure data was not bypassed. This status code is also returned if the index block containing the requested key on the last call no longer exists and an index restart was required. In this case, the returned data is correct. Use **SHOWX** for extended status to determine which event occurred.

The key you are trying to position at was deleted by another task. Reestablish your position by restarting the function.

file

Description *Required.* Identifies the primary or related file to be acted upon. You can define a field containing the name of the or you can use the actual file name as a literal in the CALL statement.

Format 4 alphanumeric characters; first character must be alphabetic

Consideration The file can be either a related or primary file.

option-list

Description *Required.* Points to a field defining the options to be used in accessing the index file.

Format Variable length; maximum field length is 62 bytes. You can use the long form or abbreviated form of the keywords. Do not mix them. No blanks are allowed.

$$\text{SECKEY} = (\text{seckey} - \text{name}) \left[\begin{array}{l} \text{,DIRECTION} = \left\{ \begin{array}{l} \text{FORWARD} \\ \text{REVERSE} \end{array} \right\} \\ \text{,KEYCOUNT} = \left\{ \begin{array}{l} \text{NO} \\ \text{YES} \end{array} \right\} \end{array} \right] \left[\text{,MASK} = \left\{ \begin{array}{l} \text{NO} \\ \text{YES} \end{array} \right\} \right] \text{,END.}$$

or

$$\text{SK} = (\text{seckey} - \text{name}) \left[\begin{array}{l} \text{,DR} = \left\{ \begin{array}{l} \text{F} \\ \text{R} \end{array} \right\} \\ \text{,KC} = \left\{ \begin{array}{l} \text{N} \\ \text{Y} \end{array} \right\} \end{array} \right] \left[\text{,MK} = \left\{ \begin{array}{l} \text{N} \\ \text{Y} \end{array} \right\} \right] \text{,END.}$$

Options The options you choose must coordinate with the *qualifier* parameter. The options can be coded in any order but must end with END. Use a comma between each option you code. No blanks are allowed between options. The options are as follows:

SECKEY=(seckey-name)

Description *Required.* Identifies the secondary key to use. The secondary key name must be defined on the Directory for the file you named in the *file* parameter.

DIRECTION = { **FORWARD** }
 { **REVERSE** }

Description Mutually exclusive with KEYCOUNT.

Format Specifies a sequence for the PDM to read the secondary keys on the index file, and, to read the “hit” list for each secondary key value.

Default FORWARD

KEYCOUNT = { **NO** }
 { **YES** }

Restriction Mutually exclusive with DIRECTION.

Description Specifies whether you want a count instead of data records.

Default NO

Options NO The PDM returns an associated data record to the data area for each READX.

YES A one-time read to scan the index and return the totaled index count of the number of data records having the full or partial value specified in the *qualifier* parameter (*kkkk...*). The PDM returns the count as a binary number in the first 4 bytes of the data area.

Considerations

- ◆ This allows you to traverse the secondary tree structure without accessing the primary or related file.
- ◆ You can use KEYCOUNT in conjunction with MASK=YES if *ssss* and *kkkk...* in the *qualifier* denote the full key length.

$$\text{MASK} = \left\{ \begin{array}{l} \text{NO} \\ \text{YES} \end{array} \right\}$$

Description Specifies whether the *kkkk...* value in the *qualifier* parameter is a literal value or a masked value.

Default NO

Options NO *kkkk* in the *qualifier* is an actual full or partial secondary key value. (A partial value provides a form of generic search.)

YES *kkkk* is a full secondary key masked value. (This provides another form of generic search.) See Considerations of the *qualifier* parameter for the rules of key mask construction.

END.

Description Specifies the options list delimiter.

qualifier

Description *Required.* Points to a variable length field in which you establish the starting point, and the PDM maintains the current position, for the serial indexed reads.

Format *qqqqccccrrrrssss[kkkk...]*

where:

qqqq A 4-character field that directs the PDM action as follows:

BEGN You code this for the first read. The PDM binds your *option-list* parameter content and creates an internal context area for it, pointed to by *cccc*.

NEXT Returned after each read so you can continue without resetting.

REBD You can set *qqqq* to REBD any time after the first read, to change the DIRECTION option (in *option-list* parameter). The PDM rebinds your options and performs a read. This returns the next record in the opposite direction. (The KEYCOUNT and MASK options must be removed if you do this).

END. Returned on the next read following the final applicable record, signaling termination of the process and freeing of context. Also, you can set *qqqq* to END. any time after the first read, to terminate the process before the PDM does, thus freeing the context.

cccc A binary fullword you initialize to blanks. This field points to the context area the PDM creates for the bound *option-list*. Do not alter this field.

rrrr A binary fullword you initialize to blanks. This field receives the returned RRNs (BDAM and ESDS only) of each data record for each secondary key. If the associated data file is KSDS, this value has no meaning. Also, if KEYCOUNT is used, this value has no meaning.

ssss A zoned numeric field you initialize to zero if no *kkkk* given, or to the length (size) of *kkkk*. In COBOL, this is a PIC 9(4) field.

kkkk... An optional character field you can set to a partial or full secondary key value if *ssss* is not zero. This begins the serial read at a location other than the beginning or end of the index file.

Considerations

- ◆ On the first READX of a series, your program must set *qqqq* to BEGN. If DIRECTION=FORWARD in the *options* parameter, the PDM starts with the first secondary key in the index (lowest), or the first entry that matches *kkkk*. The “hit” list for this lowest key is retrieved from beginning to end. If DIRECTION=REVERSE, the PDM starts with the last (highest), or the last entry that matches *kkkk*. The “hit” list for this highest key is retrieved from end to beginning. Upon successful retrieval of a data record, the PDM changes *qqqq* to NEXT, sets *cccc* to point to the unique bound identifier, and sets *rrrr* to the RRN of the returned data record.

To read the next record in the same DIRECTION, your program must not change *qualifier* before issuing the next READX. After the PDM has returned the last record in the same DIRECTION, the next READX sets *qqqq* to END. and frees the internal context.

- ◆ If you provide a full unmasked *kkkk*, only one record will match the secondary key when the index is unique or is over a primary file control key. In this case, the PDM returns END. in *qqqq* instead of NEXT.
- ◆ Once a READX returns NEXT in the *qqqq* field, the program can alter it to either REBD or END. If the program alters it to any other value, or if it alters other fields in the qualifier, unpredictable results or errors could occur.
- ◆ Use END. in *qqqq* (leaving *cccc* undisturbed) to terminate processing prior to the PDM END. signal in *qqqq*. This frees the context area (each execution with BEGN creates a new context area). If a great number of READX commands by one or various tasks are abandoned without freeing the context, ICOR status returns are probable for all tasks.
- ◆ You can use REBD in *qqqq* (leaving *cccc* undisturbed), remove the MASK or KEYCOUNT options if coded, and reinitialize the DIRECTION in the *option-list* parameter. This changes the direction of the secondary key search before END. is reached. The next READX returns data field values in accordance with the change in direction. If a MASK was being used, that masking is still in effect.
- ◆ Initialize the *ssss* parameter to the length of a *kkkk* value you provide (partial or full key value). If you are not using *kkkk*, initialize *ssss* to zero for the first read (BEGN). The READX series will read every record, starting with the lowest or highest secondary key (according to DIRECTION).

- ◆ The optional *kkkk...* parameter directs the PDM to process only the records that match this beginning or full value. If *kkkk* value is a partial key value, *ssss* should denote the partial length. The search stops with END. when there are no more keys matching the partial or full value.

A full length *kkkk* can be masked by using MASK=YES in the *options* parameter and using the masking characters (as described in the following) within the *kkkk*. This provides a generic search.

- ◆ A mask can consist of any valid EBCDIC value. The characters &, #, and @ in the value have special meaning to the mask processor. All other characters in the mask mean that a secondary key value must match exactly in those positions to qualify for selection. The comparison is made as if all bytes are character. That is, there are no considerations for packed, zoned or other data types. The special meanings are:
 - a. The character & means that any valid EBCDIC value can be in this position of the secondary key value.
 - b. The character # means that a number 0-9 must be in this position of the secondary key value.
 - c. The character @ means that a letter (upper or lower) must be in this position of the secondary key value.

As an example mask, if you are looking for B100####AA records, use B100####AA. If you want only the 1009's, use B1009##@ @ or B1009&&&&. A forward search ends when the first record beginning with C is encountered. A reverse search ends with the first A record. However, if you use &1009####AA, the entire secondary key is scanned before END. because the & in first position requests all keys (any character in this position).

- ◆ Since the comparison is made as if all bytes in the key are character (2.1.6 only), take care if they are not. The binary representation of the mask characters may be the same as a data value you want to search.
- ◆ If you use a mask, be sure to mask the entire length of the secondary key. If you make it shorter (partial), no match is found. The *ssss* field of the qualifier should specify the full length.
- ◆ The READX returns only those data items named in your data list. For coded data lists, only the records with those codes are processed for data retrieval (or for KEYCOUNT).

data-area-len

Description	<i>Required.</i> Required for compatibility.
Format	4-byte field, set to unsigned zero

data-list

Description *Required.* Points to a field containing a list of data items. This list acts as a map of the layout of the data area. Compose this list using data names (physical fields) defined on the Directory.

Format *dataitem1,dataitem2,...dataitemn,END.*

Considerations

- ◆ The commas between the entries are optional and serve only as separators; be consistent whether you use them or not.
- ◆ The data list can include the names of data items, control keys, and record codes. Do not name linkpaths or the root field. Do not list any name twice. If a name is not accepted, the PDM returns an error status code.
- ◆ You can list the data names in any order. They are processed in the order listed, not in the order defined on the Directory. However, for coded records, the record code must be first in the list. If not, unpredictable results occur. See “[Data list parameter keywords](#)” on page 62 for coded data list construction rules.
- ◆ **VSE** When using a cross-address space central PDM in VSE/ESA with XPCC=YES, the data area mapped by this *data-list* parameter is limited by the CSIPARM MAXIO value. This governs DML having no *length* parameter. Refer to the *SUPRA Server PDM and Directory Administration Guide (OS/390 & VSE)*, P26-2250, for CSIPARM information.
- ◆ If all the fields in the *data-list* are contained in the secondary key and the end parameter is RLSE, your DML request may qualify for the performance option for indexed reads. See “[Diagnosing application DML errors](#)” on page 36 for more information.
- ◆ If all the fields in the *data-list* are contained in the secondary key and the end parameter is RLSE, your DML request may qualify for the performance option for indexed reads. See “[Reading records via secondary keys](#)” on page 35 for more information.

data-area

- Description** *Required.* Points to a field or fields to be used as an input area for the data to be retrieved from the record.
- Format** The structure and characteristics of the data area must conform exactly to the Directory definition of the data items (physical fields) named in the data list.

Considerations

- ◆ The data area must be large enough to hold values for all data items named in the data list.
- ◆ The *data-area* parameter and the *data-list* parameter have corresponding fields. The data list holds names, and the data area holds a value for each of those names.

end

Description *Required.* Points to a 4-character field that delimits the parameter list and determines the record holding function (see “Record holding” on page 42).

- Options**
- | | |
|------|---|
| END. | Holds a retrieved record. |
| RLSE | Does not hold a retrieved record; can read an uncommitted record. |

Consideration END. does not affect the secondary key records on the index file. They can still undergo maintenance during the READX series.

General considerations

- ◆ Do not execute the **ADD-M**, **DEL-M**, or **DELVD** commands while processing a file with READX, since the index structure could be altered by the add/delete operation. A repeat READX will then not be positioned correctly. It may miss some records in the scan and may read others multiple times.
- ◆ This command causes the PDM to create a unique context area for use with repeated reads. It is very important that you control the release of this context area. See the *qualifier* parameter description for details.

RESET

The RESET command serves two purposes. If issued during regular processing, RESET restores updated records and file status to their condition at the most recent **COMIT** of this task, or its **SINON**. This backs out an incomplete logical unit of work. If the RESET is issued in response to a RSTR status at SINON, the PDM retrieves the task's last-saved COMIT data area.

RESET, *status*, *comit-id*, *length*, *data-area*, END.

status

Description *Required.* Points to a field into which the PDM places a status code indicating the result of the command.

Format 4-byte field

Consideration If the command fails or if the status code indicates some special condition other than failure, your program should include logic to handle and possibly correct the situation. For a list of all status codes, refer to the *SUPRA Server PDM Messages and Codes Reference Manual (RDM/PDM Support for OS/390 & VSE)*, P26-0126.

RESET *comit-id*

Description	<i>Required.</i> Points to an area to contain the literal LAST or a commit number denoting last commit ID.				
Format	4-byte field				
Options	<table><tr><td>LAST</td><td>The PDM finds the most recent commit number for this task.</td></tr><tr><td><i>comit-number</i></td><td>A binary fullword denoting the most recent commit number for this task.</td></tr></table>	LAST	The PDM finds the most recent commit number for this task.	<i>comit-number</i>	A binary fullword denoting the most recent commit number for this task.
LAST	The PDM finds the most recent commit number for this task.				
<i>comit-number</i>	A binary fullword denoting the most recent commit number for this task.				

Considerations

- ◆ The PDM cannot reset further back than the most recent commit.
- ◆ If you use a commit number, it must be equal to the commit number the PDM returned to the task's most recent **COMIT**. This form of the parameter is provided so that you can use the COMIT command's corresponding field without reinitializing. If you RESET before issuing any COMIT during regular processing, use zero (or LAST).
- ◆ If you RESET because of a RSTR status on **SINON**, use LAST instead of *comit-number*.
- ◆ After completion, the PDM updates this field to the commit number of the most recent **COMIT** issued by this task or a zero if this task has issued no COMITs.

length

Description *Required.* Points to a 4-byte area containing the length of the *data-area* parameter.

Format Binary fullword

Considerations

- ◆ Set this parameter to zero for a regular RESET unless you really want data you saved at your last **COMIT**. Set it to the appropriate length for the special RESET following a RSTR status at **SINON**.
- ◆ *Length* must be at least the length of the data saved by the most recent **COMIT**, or the PDM returns a warning status code (*SIZ). In this case, the first *n* bytes of commit data are moved to the data area, where *n* is the length you specified.
- ◆ After completion, the PDM updates this field to contain the exact length of the data that was saved with the most recent **COMIT**. The length actually moved to the data area is less if the *status* return is *SIZ. If this updated value is zero, there was no saved data, or the program had not issued a COMIT before this point.
- ◆ Since records are spread physically across system log blocks, there is no theoretical limit to the size of your data area. However, you must consider your program's available memory and your Task Log File capacity.
- ◆ **VSE** When using a central PDM in a cross-address space VSE/AF SP2.1 (XPCC=YES), the RESET data area's maximum length is limited by the CSIPARM MAXPACKET value. Refer to the *SUPRA Server PDM and Directory Administration Guide (OS/390 & VSE)*, P26-2250, for CSIPARM information.

data-area

Description *Required.* Points to a field to receive the data you saved with the most recent **COMIT**.

Considerations

- ◆ The data area must be at least as long as the length specified in the associated *length* parameter.
- ◆ If you issue a **RESET** after receiving an **RSTR** status from **SINON**, the PDM returns the commit data you saved at the most recent commit point taken before program failure.

end

Description *Required.* Points to a field that delimits the parameter list.

Format **END.**

General considerations

- ◆ If task logging is not active when you issue this command, the PDM returns an error status code.
- ◆ If this **RESET** is issued before the program's first **COMIT**, the PDM backs out the incomplete logical unit of work back to the **SINON**, but it is still signed on. The PDM returns a zero in the *commit-id* parameter.
- ◆ If your task fails before it issues the first **COMIT** or **RESET**, the PDM backs out the incomplete logical unit of work back to the **SINON**, and signs off the task.

RQLOC

The RQLOC (Request Location) command executes the PDM hashing algorithm which determines primary file record locations (see “[Adding a primary record](#)” on page 22). It accepts a control key value as input and generates an RRN as output. It performs no I/O operations. With this command, you can, among other things, build a file of control keys and their respective RRNs. This file can be sorted on RRNs and the data then processed at maximum speed against the primary file.

RQLOC, status, file, control-key, data-area, end

status

Description *Required.* Points to a field into which the PDM places a status code indicating the result of the command.

Format 4-byte field

Consideration If the command fails or if the status code indicates some special condition other than failure, your program should include logic to handle and possibly correct the situation. For a list of all status codes, refer to the [SUPRA Server PDM Messages and Codes Reference Manual \(RDM/PDM Support for OS/390 & VSE\)](#), P26-0126.

file

Description *Required.* Identifies the primary file to be acted upon. You can define a field containing the name or you can use the actual file name as a literal in the CALL statement.

Description 4 alphanumeric characters; first character must be alphabetic

Considerations

- ◆ The file must be accessed via ESDS or BDAM, or the PDM returns an error status code.
- ◆ The file must be a primary file, or the PDM returns an error status code.

control-key

- Description** *Required.* Points to a field containing the key to process. The PDM uses this key to determine the value of the home location in the file.
- Format** Variable length as defined in the Directory
-

data-area

- Description** *Required.* Points to a field the PDM uses to return the RRN.
- Format** Binary fullword
-

end

- Description** *Required.* Points to a field that delimits the parameter list.
- Format** END.

General consideration

Your task need not open a file before it issues RQLOC for a key value in that file. The key value need not even exist.

RSTAT

Use the RSTAT command to write statistics to the Statistics File when the Directory specifies STATS=yes. (See “[Generating statistics](#)” on page 51 for a general discussion of RSTAT usage.)

RSTAT, *status, option-list, end*

status

- Description** *Required.* Points to a field into which the PDM places a status code indicating the result of the command.
- Format** 4-byte field
- Consideration** If the command fails or if the status code indicates some special condition other than failure, your program should include logic to handle and possibly correct the situation. For a list of all status codes, refer to the [SUPRA Server PDM Messages and Codes Reference Manual \(RDM/PDM Support for OS/390 & VSE\)](#), P26-0126.

option-list

Description *Required.* Points to a field that controls the resetting of the statistics counters or determines which records, if any, the PDM writes to the Statistics File.

Format There are two possible formats. When constructing your list, separate the options with commas and terminate the list with END. No blanks are allowed between characters.

ACTION=CLEAR,END.

or

$$\text{ACTION} = \text{WRITE} \left[, \text{SYSTEM} = \begin{Bmatrix} \text{NO} \\ \text{YES} \end{Bmatrix} \right] \left[, \text{FILE} = \begin{Bmatrix} \text{ALL} \\ (\text{file - name}) \end{Bmatrix} \right] , \text{END.}$$

Options The options depend on which format you chose.

ACTION=CLEAR

Description *Required.* The PDM resets the statistics counters to zero and no record is written. All subsequent statistics are gathered relative to this point, not the start of this PDM execution.

ACTION=WRITE

Description *Required.* The PDM writes the accumulated statistics to the Statistics File, according to the next two parameters. From this file you can print a report with the Execution Statistics utility.



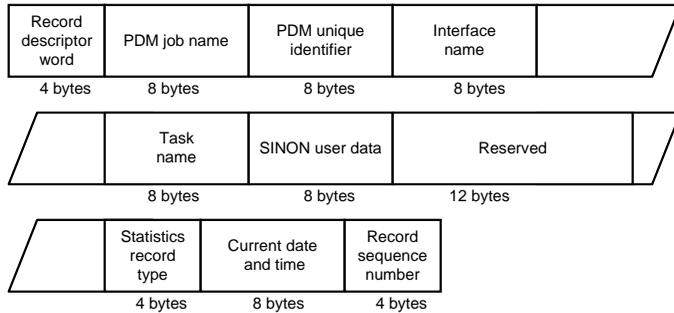
When the PDM initializes or terminates, it automatically writes an initialization or termination record and all statistics to the Statistics File when STATS=YES. Your RSTAT commands write additional records in between. See General Considerations.

General considerations

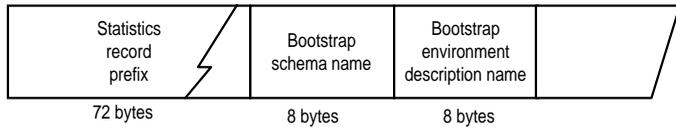
- ◆ RSTAT operations are not recoverable. Updates to the Statistics File do not generate before image or after image log records. Also, the PDM does not generate a function log record for this command.
- ◆ If you want statistics generated, use Directory Maintenance to enter a Y for the Statistics Indicator on your environment description. If you do not enter Y and then issue an RSTAT command, the PDM returns an error status code.
- ◆ The PDM generates the first group of records when it initializes. This group of records contains only initialization statistics. The PDM then resets the statistics counters to zero, so that all performance data is separate from initialization data.
- ◆ The PDM writes a final group of statistics records before closing the Statistics File at PDM termination. They contain all values since the last time the statistics counters were reset; that is, since initialization, or since the last RSTAT command with ACTION=CLEAR issued by any user.
- ◆ Due to the asynchronous processing between the PDM and applications, the command counts in the accumulated statistics records may differ slightly.
- ◆ The RSTAT command writes only accumulated statistics to the Statistics File. The Execution Statistics utility produces calculated statistics in addition. For information on the calculated statistics and the utility, refer to the *SUPRA Server PDM DBA Utilities User's Guide (OS/390 & VSE)*, P26-6260.
- ◆ To process the file yourself, you need to know its layout, which is shown in the following 4 figures. The Statistics File is a sequential output file defined in the active realm.

- ◆ The table following the file layout figures presents all the statistics gathered by the RSTAT command. The table shows each statistic's identifier on the Execution Statistics report, the field size of the data area required for holding each statistic, and an explanation of the statistic. This table also cross references an RSTAT statistic to a corresponding **SHOWX** item request where applicable.

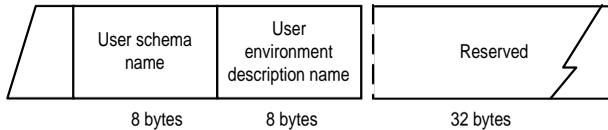
Statistics record prefix (common to all records)



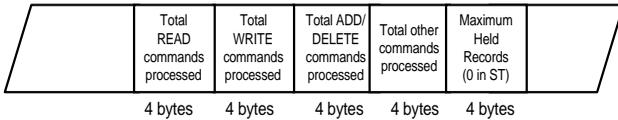
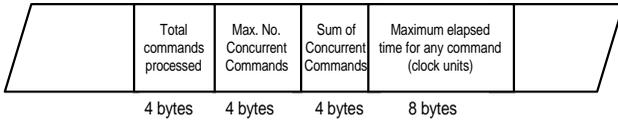
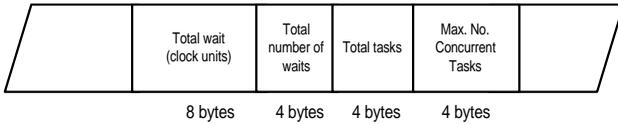
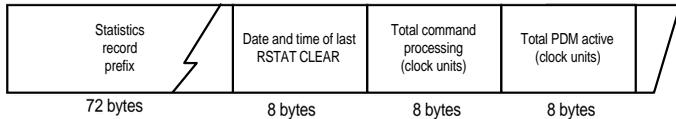
Initialization and termination records



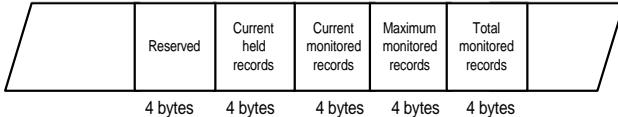
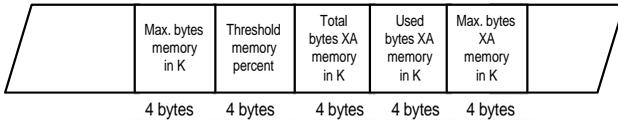
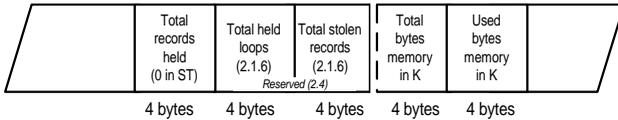
2.4 and higher only

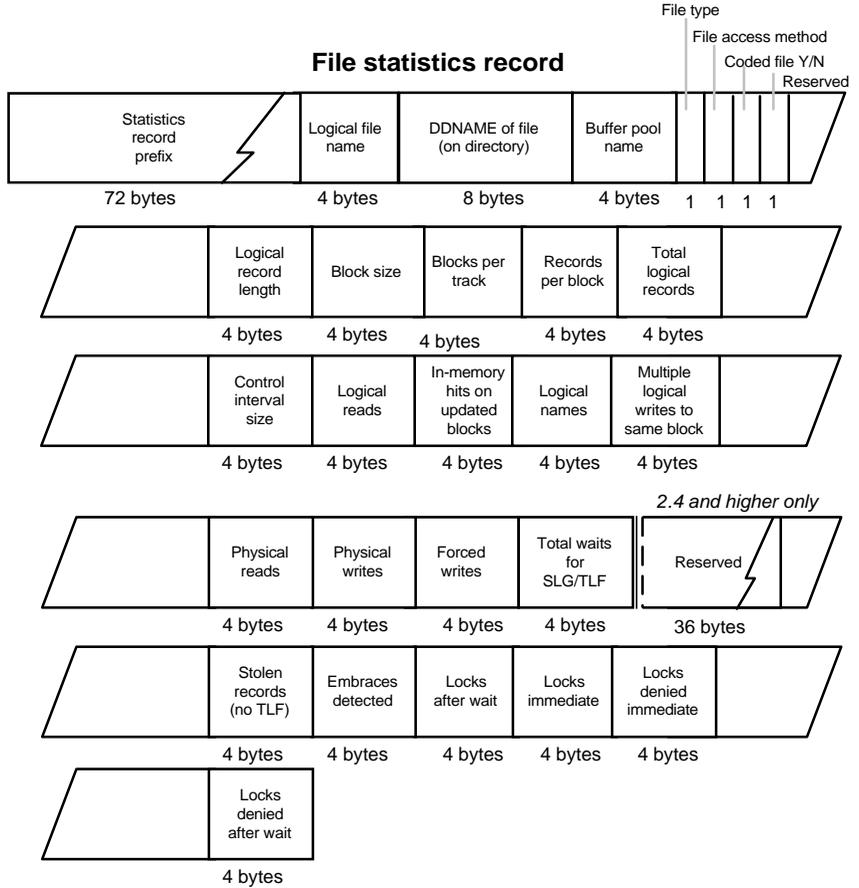


System statistics record



2.4 and higher only





Common prefix information

Recorded statistics	Statistic identifier	Field size	Matching SHOWX item*	Additional explanation
Record Descriptor Word	n/a	Fullword	n/a	2 bytes binary zeros, 2 bytes record length. For internal use.
PDM Name	n/a	8 bytes	DBMXNAME	Job name of PDM or name from DBM=parameter in CSIPARM file.
PDM Unique Identifier	n/a	8 bytes	DBMXNAME	For internal PDM use.
Interface Name	n/a	8 bytes	IFCENAME	Job name of interface or name from INTERFACE=parameter in CSIPARM.
Task Name	n/a	8 bytes	TASKNAME	User program name.
SINON User Data	n/a	8 bytes	n/a	User data supplied to PDM SINON command.
Statistics Record Type	n/a	4 bytes	n/a	INIT=initialization TERM=termination DBMX=system FILE=file
Date and time this record was generated	n/a	Two 4-byte packed decimal numbers, with Julian date <i>YYYYDDDF</i> and Time <i>OHHMMSSF</i>	n/a	The date and time when the PDM received the RSTAT command. All records written by the same RSTAT command have the same time and date.
Record Sequence number	n/a	Fullword	n/a	Sequential numbering of records written during one RSTAT. For internal PDM use.

* This column shows how to relate RSTAT statistics to a corresponding item request with the **SHOWX** command.

Initialization/termination record

Recorded statistics	Statistic identifier	Field size	Matching SHOWX item*	Additional explanation
Bootstrap Schema Name	n/a	8 bytes	SCHMLNAM	Loaded schema name.
Bootstrap Environment Description Name	n/a	8 bytes	ENVDLNAM	Loaded environment description name.
Schema Name	n/a	8 bytes	SCHMNAME	The name of your active schema.
Environment Description Name	n/a	8 bytes	ENVDDNAME	Your active environment description name.

* This column shows how to relate RSTAT statistics to a corresponding item request with the **SHOWX** command.

System statistics record

Recorded statistics	Statistic identifier	Field size	Matching SHOWX item*	Additional explanation
Date and time statistics were last reinitialized	S1.01	Two 4-byte packed decimal numbers, with Julian date <i>YYYYDDDF</i> , and Time <i>OHHMMSSF</i>	DBMXRSTZ	The date and time statistics counters were last initialized (by startup or by the CLEAR option of the RSTAT command).
Total elapsed time of commands issued to the PDM	S5.05 (2.1.6) S6.01 (2.4)	Doubleword (clock units)	DBMXPRTM	The total elapsed time of all applications. This measures the time from when PDM interfaces issue a command until they receive command completion.
Amount of time PDM was active	S7.01 (2.1.6) S8.01 (2.4)	Doubleword (clock units)	DBMXDBTM	The total amount of elapsed time the PDM was executing.
Amount of time PDM was inactive	S7.02 (2.1.6) S8.02 (2.4)	Doubleword (clock units)	DBMXWTTM	The total amount of elapsed time the PDM was in an operating system wait mode.
Number of times PDM was inactive	S6.01 (2.1.6) S7.01 (2.4)	Fullword	DBMXWAIT	The total number of times the PDM issued an operating system wait because there was no processing to do.
Total tasks	S2.01	Fullword	DBMXTSKS	The total number of tasks that signed onto the PDM. For single-task PDM environments, this value is always one.

* This column shows how to relate RSTAT statistics to a corresponding item request with the **SHOWX** command.

Recorded statistics	Statistic identifier	Field size	Matching SHOWX item*	Additional explanation
Maximum concurrent tasks	S2.02	Fullword	DBMXMCNT	The highest number of tasks that were ever signed on simultaneously. For single-task PDM environments, this value is always one.
Total commands issued to the PDM	S5.01	Fullword	DBMXCMDS	The total number of commands issued to the PDM by PDM interfaces.
Maximum number of commands at command starts	S5.02	Fullword	DBMXMCNC	The highest number of commands processing at any one command start (includes the one starting and those awaiting processing, see the figure following these tables).
Sum of commands at command starts	S5.03	Fullword	DBMXSCNC	Sum of commands processing at each command's start (see the figure following these tables).
Maximum elapsed time for any command issued to the PDM	S5.07 (2.1.6) S6.03 (2.4)	Doubleword (clock units)	DBMXMCMT	The highest elapsed time to process a single command out of all applications.

* This column shows how to relate RSTAT statistics to a corresponding item request with the **SHOWX** command.

Recorded statistics	Statistic identifier	Field size	Matching SHOWX item*	Additional explanation
Total read commands	S4.01	Fullword	DBMXDMLR	The total number of times FINDX , RDNXT , READD , READM , READR , READV , and READX commands were executed. This includes only those commands recognized by the PDM.
Total update commands	S4.02	Fullword	DBMXDMLW	The total number of times the WRITD , WRITM , and WRITV commands were executed. This includes only those commands recognized by the PDM.
Total add and delete commands	S4.03	Fullword	DBMXDMLA	The total number of times the ADDVR , ADD-M , ADDVA , ADDVB , ADDVC , DEL-M and DELVD commands were executed. This includes only those commands recognized by the PDM.
Total other commands	S4.04	Fullword	DBMXDMLO	The total number of commands executed other than reads, writes, adds or deletes. This includes only those commands recognized by the PDM.
Maximum held records	S3.02	Fullword	DBMXMHRC	The highest number of records held at any one time. Always zero (0) for ST.
Total held records	S3.01 (2.1.6)	Fullword	DBMXTHRC	The total number of held records. Always zero (0) for ST.

* This column shows how to relate RSTAT statistics to a corresponding item request with the **SHOWX** command.

Recorded statistics	Statistic identifier	Field size	Matching SHOWX item*	Additional explanation
Current held records	S3.01 (2.4)	Fullword	DBMXCHRC	For 2.4, this has no identifier (is not reported). The current number of held records.
Total records held by other tasks	S3.03 (2.1.6)	Fullword	DBMXHLDL	The total number of times a record desired by a task was held by other tasks. Always zero (0) for ST.
Current monitored records	S3.03 (2.4)	Fullword	DBMXCIRC	The current number of shared records.
Total monitored records	None (2.4)	Fullword	DBMXFIDC	The total number of shared records.
Total held records stolen by another task	S3.04 (2.1.6)	Fullword	DBMXSREC	The total number of times a record was stolen from a TP Monitor task for use by another task in the PDM. When task logging is active, and for single-task PDM, this is always zero. For 2.4, this is a File statistic.

- * This column shows how to relate RSTAT statistics to a corresponding item request with the **SHOWX** command.

Recorded statistics	Statistic identifier	Field size	Matching SHOWX item*	Additional explanation
Maximum monitored records	S3.04 (2.4)	Fullword	DBMXMIRC	The highest number of shared records.
2.4 and higher only				
Total memory in K	S9.01 S10.01	Fullword	DBMXMEMT DBMXXEMT	The amount of memory and XA memory available to the PDM.
Current used memory in K	S9.02 S10.02	Fullword	DBMXMEMU DBMXXEMU	The amount of memory and XA memory currently being used.
Maximum memory in K	S9.03 S10.03	Fullword	DBMXMEMH DBMXXEMH	The highest amount of memory and XA memory used by the PDM.
Threshold memory in K	S9.04 S10.04	Fullword	DBMXMEMP	The percentage used-to-available at which the PDM begins releasing non-critical memory.

* This column shows how to relate RSTAT statistics to a corresponding item request with the **SHOWX** command.

File statistics record (one record for each file accessed)

Recorded statistics	Statistic identifier	Field size	Matching SHOWX item*	Additional explanation
Logical file name	n/a	4 characters	FILENAME	The file name.
DDNAME of file on Directory	n/a	8 characters	FILEDDNM	The Directory DDNAME for this file.
Buffer pool used	n/a	4 characters	FILEBPOL	The name of the buffer pool used by this file.
File type	n/a	1 character	FILETYPE	P=Primary R=Related I=Index file S=Statistics file L=System Log file T=Task Log file
File access method	n/a	1 character	FILEACES	B=BDAM E=ESDS K=KSDS S=BSAM O=Output VSE W=Work VSE
Coded file notice	n/a	1 character	FILECDSW	Y=a coded file N=not a coded file
Total logical reads	F1.01	Fullword	FILELRED	The number of times a record in this file was logically read.
Total in-memory hits on updated buffer	F1.04	Fullword	FILEIHIT	The number of logical reads for this file that found the block of data in a storage buffer which had already been updated.

* This column shows how to relate RSTAT statistics to a corresponding item request with the **SHOWX** command.

Recorded statistics	Statistic identifier	Field size	Matching SHOWX item*	Additional explanation
Total logical updates	F3.01	Fullword	FILELWRT	The number of times a record in this file was logically written to.
Total multiple logical updates to same buffer	F3.03	Fullword	FILEMLTW	The number of logical writes for this file to storage buffers that had already been updated.
Total physical reads	F1.02	Fullword	FILEPRED	The number of times a record in this file was physically read by the PDM.
Total physical updates	F3.02	Fullword	FILEPWRT	The number of times the contents of a buffer for this file was physically written.
Total physical updates forced by a physical read	F1.05	Fullword	FILEFWRT	The number of logical reads to this file that forced a physical update in order to obtain the buffer for a physical read.
Total waits for logging	F6.01	Fullword	FILELOGW	The number of physical updates to this file that had to wait for a physical write to the Task Log or System Log.

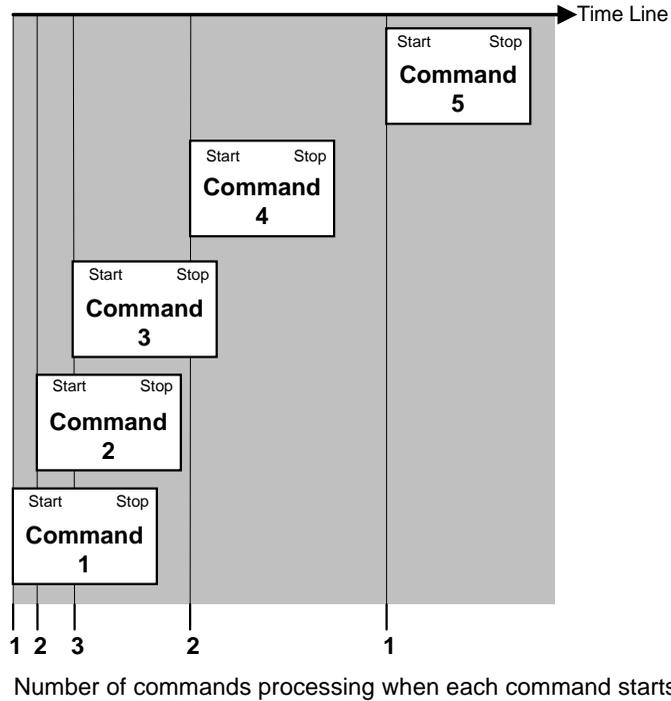
* This column shows how to relate RSTAT statistics to a corresponding item request with the **SHOWX** command.

2.4 and higher only

Recorded statistics	Statistic identifier	Field size	Matching SHOWX item*	Additional explanation
Total stolen records	F7.01	Fullword	FILESREC	The number of times a record in this file held by a task was released involuntarily and granted to another task. This can occur only when task logging is not active.
Total embraces detected	F7.07	Fullword	FILEEMBR	The number of times a lock request for a record in this file was denied because granting would cause an unresolvable locking conflict.
Total locks after wait	F7.03	Fullword	FILEGDWT	The number of times a record in this file was locked after waiting part of DELAY time for the record to become available.
Total locks immediate	F7.02	Fullword	FILEGDIM	The number of times a record in this file was available and locked immediately upon request.
Total locks denied immediate	F7.05	Fullword	FILEHDIM	The number of times a record in this file was locked at the time of request for immediate.
Total locks denied after wait	F7.06	Fullword	FILEHDWT	The number of times a record in this file was still locked after waiting DELAY time.

* This column shows how to relate RSTAT statistics to a corresponding item request with the **SHOWX** command.

How the PDM gathers command start statistics



The PDM counts the number of commands processing each time a new command starts. It counts the command that is starting as the first one. In the example shown in the preceding figure, only one command is processing when Command 1 starts—Command 1 itself. When Command 2 starts, two commands are processing—Command 1 and Command 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—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 (the S5.02 identifier in preceding tables), PDM picks the highest number from the figures along the base line—3.

To arrive at the sum of commands processing when commands start (the S5.03 identifier), the PDM adds the numbers along the base line (1+2+3+2+1=9).

SHOWX

Use the SHOWX command to retrieve internal PDM information. There are two kinds of information:

- ◆ For diagnosing application DML errors (non-**** status returns), see “[Diagnosing application DML errors](#)” on page 36 for how to use, and “[SHOWX for status returns](#)” on page 221 for the syntax description.
- ◆ For monitoring PDM performance and resource usage, see “[Monitoring resources](#)” on page 53 for how to use, and “[SHOWX for monitoring resources](#)” on page 224 for the syntax description.

SHOWX for status returns

Use the first format of SHOWX with TASKEXT to retrieve task extended status information. This returns internal information stored by the PDM about a DML error (unsuccessful status return).

SHOWX, status, option-list, qualifier, data-list, data-area, end.

status

Description *Required.* Points to a field into which the PDM places a status code indicating the result of the command.

Format 4-byte field

Consideration If the command fails or if the status code indicates some special condition other than failure, your program should include logic to handle and possibly correct the situation. For a list of all status codes, refer to the [SUPRA Server PDM Messages and Codes Reference Manual \(RDM/PDM Support for OS/390 & VSE\)](#), P26-0126.

option-list

Description *Required.* Points to a field containing the options to use for this SHOWX.

Format *option-list*,END.

Options

TASK=*	Current task
TASK=(<i>task-name</i>)	Task name from SINON

qualifier

Description *Required.* Points to an area containing your qualifier.

Format 16 bytes for *qqqqiiiioooocccc*

where:

qqqq A 4-character field that directs the PDM action as follows:

BEGN You code for the first SHOWX.

NEXT Returned after execution.

ENDS You may place over NEXT without modifying *cccc*.

END. Returned after second execution.

iiii A binary fullword you initialize to the binary length of your data area (including END.). In COBOL, this is a PIC S9(8) COMP field. This length must be at least 0036 for SHOWX with TASKEKST. Ensure that this is set properly. If the space is actually smaller than the length you specify here, the PDM writes over any data or code already there, with unpredictable results.

oooo A binary fullword you initialize to blank. This field receives the actual length of information returned to the data area, including END. The PDM supplies this value after the BEGN execution.

cccc A binary fullword you initialize to blanks. This field receives the context-ID. This is the identifier the PDM uses to locate the context memory where your SHOWX information is stored. Do not modify this field; leave it as the PDM returns it to you. You reinitialize this to blanks only for a new SHOWX request.

Consideration See [“Diagnosing application DML errors”](#) on page 36 for discussion and example qualifier usage with SHOWX for TASKEKST.

data-list

Description	<i>Required.</i> Specifies the keyword you use to request task error information.
Format	TASKEXST,END.

data-area

Description	<i>Required.</i> Points to a field to receive the SHOWX information.
Format	At least a 36-character area

Considerations

- ◆ You must code the data area at least as long as the length you specified in the *length-in* field in the *qualifier* parameter.
- ◆ The returned data is a repeat of the status you received on the prior failed command, with a 4-byte identification number, and with text helping to identify the error.
- ◆ You should not code your application program to handle specific status identification numbers because they may change with different releases of SUPRA Server.

end

Description	<i>Required.</i> Points to a field that delimits the parameter list.
Format	END.

General considerations

- ◆ Before you can issue a SHOWX command, a task must be signed on. In other words, you cannot issue a SHOWX for an error on a **SINON**. If you do, the PDM returns an error status code to the SHOWX command.
- ◆ The SHOWX command is not recoverable. If the PDM fails before you have retrieved the error information, you must re-create the error and attempt SHOWX again after PDM restart.

SHOWX for monitoring resources

Use the second format of **SHOWX** to retrieve performance and monitoring information. For how to use this command, see “[Monitoring resources](#)” on page 53.

SHOWX, status, option-list, qualifier, data-list, data-area, end.

status

Description *Required.* Points to a field into which the PDM places a status code indicating the result of the command.

Format 4-byte field

Consideration If the command fails or if the status code indicates some special condition other than failure, your program should include logic to handle and possibly correct the situation. For a list of all status codes, refer to the [SUPRA Server PDM Messages and Codes Reference Manual \(RDM/PDM Support for OS/390 & VSE\)](#), P26-0126.

option-list

Description	<i>Required.</i> Points to a field containing the hierarchy and identification of the group level for which you want results.
Format	<i>option1</i> [, <i>option2</i> ,...],END.
Options	The options and their possible values are listed below. Use commas to separate the options. The value ALL can be used only when the option is at the end of the hierarchy.
DBM=*	Current executing PDM.
SCHEMA=*	Your current active schema.
SCHEMA=(<i>schema-name</i>)	Your current active schema name.
ENVDESC=*	Your current environment description.
ENVDESC=(<i>envdesc-name</i>)	Your current environment description name.
BUFFPOOL=(<i>buffpool-name</i>)	A buffer pool in this environment.
BUFFPOOL=ALL	All buffer pools in this environment.
FILE=(<i>file-name</i>)	A file in this environment.
FILE=ALL	All files in this environment.
FIELD=(<i>field-name</i>)	A physical field name for this file or secondary key.
FIELD=ALL	All physical fields in this file or secondary key.
SKID=(<i>skid-name</i>)	A secondary key name for this file.
SKID=ALL	All secondary keys for this file.
INTERFACE=*	The interface this task is using.
INTERFACE=(<i>iface-name</i>)	Any interface name.
INTERFACE=ALL	All known interfaces.
TASK=*	This task.
TASK=(<i>task-name</i>)	Any task name.
TASK= <i>all</i>	All known tasks for this interface.
END.	The required option list delimiter.

Considerations

- ◆ Code the option list to define one group, as shown in the following table. This group determines which keywords from the tables at the end of this section you can choose for your data list. (The tables at the end of this section show the groups in the same order presented here.) If you omit a bracketed section, the current entity(s) is assumed.

Option list to define group	Group in next table
DBM=*,END.	DBMX
[DBM=*,]SCHEMA=*,END.	SCHM
[DBM=*,]SCHEMA=(name),END.	SCHM
[DBM=*,SCHEMA=...,]ENVDESC=*,END.	ENVVD
[DBM=*,SCHEMA=...,]ENVDESC=(name),END.	ENVVD
[DBM=*,SCHEMA=...,ENVDESC=...,]BUFFPOOL=(name),END.	BPOL
[DBM=*,SCHEMA=...,ENVDESC=...,]BUFFPOOL=ALL,END.	BPOL
[DBM=*,SCHEMA=...,ENVDESC=...,]FILE=(name),END.	FILE
[DBM=*,SCHEMA=...,ENVDESC=...,]FILE=ALL,END.	FILE
[DBM=*,SCHEMA=...,ENVDESC=...,]FILE=(name)FIELD=(name),END.	FELD
[DBM=*,SCHEMA=...,ENVDESC=...,]FILE=(name)FIELD=ALL,END.	FELD
[DBM=*,]INTERFACE=*,END.	IFCE
[DBM=*,]INTERFACE=(name),END.	IFCE
[DBM=*,]INTERFACE=ALL,END.	IFCE
[DBM=*,INTERFACE=*,]TASK=*,END.	TASK
[DBM=*,INTERFACE=*,]TASK=(name),END.	TASK
[DBM=*,INTERFACE=*,]TASK=ALL,END.	TASK
[DBM=*,]INTERFACE=(name),TASK=*,END.	TASK
[DBM=*,]INTERFACE=(name),TASK=(name),END.	TASK
[DBM=*,]INTERFACE=(name),TASK=ALL,END.	TASK
[DBM=*,SCHEMA=...,ENVDESC=...,]FILE=(name),SKID=(name),END.	SKID
[DBM=*,SCHEMA=...,ENVDESC=...,]FILE=(name),SKID=ALL,END.	SKID
[DBM=*,SCHEMA=...,ENVDESC=...,]FILE=(name),SKID=(name),FIELD=(name),END.	SKPF
[DBM=*,SCHEMA=...,ENVDESC=...,]FILE=(name),SKID=(name),FIELD=ALL,END.	SKPF

- ◆ You can use the keyword ALL only for the last option in the list. For example, if the items you want for the data list are in the FILE group, you can use either FILE=ALL or FILE=(file-name) for the file option. However, if the items are in the FELD group, you can use only FILE=(file-name) for the file option.
- ◆ See “[Diagnosing application DML errors](#)” on page 36 and “[Monitoring resources](#)” on page 53 for discussions and example option list usage with SHOWX.

qualifier

Description *Required.* Points to an area to contain your qualifier. You initialize the *qualifier* parameter the first time with BEGN in the first field, the binary length of the data area in the second, and blanks in the 3rd and 4th fields.

Format 16 bytes for *qqqqiiiioooocccc*

where:

qqqq A 4-character field that directs the PDM action as follows:

 BEGN You code for the first SHOWX.

 NEXT Returned after each execution.

 ENDS You may place over NEXT to stop retrieval and free the PDM context. Do not modify the *cccc* when doing this.

 END. Returned when there is no more information for your chosen option (PDM has freed the context).

iiii A binary fullword you initialize to the binary length of your data area (including END.). In COBOL, this is a PIC S9(8) COMP field. Ensure that this is set properly. If the space is actually smaller than the length you specify, the PDM writes over any data or code already there.

oooo A 4-character field you initialize to blanks. This field receives the actual length of information returned to the data area (including END.). The PDM supplies this value after each execution.

cccc A 4-character field you initialize to blanks. This field receives the binary ID of the context area the PDM creates to hold your SHOWX information. The PDM supplies this value after the first execution. Do not modify this field for repeated SHOWX iterations. You change it to blanks only for a new SHOWX request.

Considerations

- ◆ If you receive a status of **** and the PDM has returned the qualifier containing END., the PDM has already freed the context area. The PDM returns END. only when it is returning no data. In this case, you do not need to repeat the SHOWX command with the qualifier containing ENDS.
- ◆ See “[Diagnosing application DML errors](#)” on page 36 and “[Monitoring resources](#)” on page 53 for discussions and example qualifier usage with SHOWX.

data-list

Description *Required.* Specifies a list of keyword items, from the one group defined in your option list, for which you are requesting information.

Format *dataitem1[dataitem2,...],END.*

Considerations

- ◆ See the table of SHOWX command keywords at the end of this section for valid data item names for the group you specified in your *option-list* parameter.
- ◆ These selected data list items plus END. map the required format and length of the data area. The data area must be large enough to contain the result of the entire list; it can be larger than needed.
- ◆ The data area can be a multiple if you use ALL for the last option in your option list (see the table of option listings earlier in this section). The PDM will return multiple entries of the data list for such a SHOWX. If your data area is not large enough to hold all the multiple entries, the PDM returns as many complete entries as will fit, and updates the qualifier to NEXT. If you want the rest, you repeat the SHOWX command without changing the qualifier.
- ◆ See “[Diagnosing application DML errors](#)” on page 36 and “[Monitoring resources](#)” on page 53 for discussions and example data list usage with SHOWX.

data-area

Description *Required.* Points to a field to receive the data list items.

Format The structure and characteristics must conform to the definition of the data list items you selected from the table of SHOWX command keywords at the end of this section, plus 4 bytes for END..

Considerations

- ◆ You must make the data area at least large enough to hold a single entry, that is, one entire data list, plus END. If larger than needed, the PDM returns as many entries as will fit in the data area.
- ◆ You must make the data area at least as long as the *length-in* value in the *qualifier* parameter.

end

Description *Required.* Points to a field that delimits the parameter list.

Format END.

General considerations

- ◆ Before you can issue a SHOWX command, your task must be signed on. Therefore, you cannot issue a SHOWX for an error on a **SINON**. If you do, the PDM returns an error status code to the SHOWX command.
- ◆ Although the *data-list* item `TASKEXST,END` returns a status identification number in the data area, you should not code your application program to handle specific numbers. They may change with different releases of SUPRA Server.
- ◆ When you use the SHOWX command for other than `TASKEXST`, code it only in special routines that you can modify easily. The SHOWX command will change as Cincom enhances SUPRA Server.
- ◆ The SHOWX command is not recoverable since it is a read-type command. If the PDM fails before you have obtained all your data, you must execute the SHOWX command again after restart. Inconsistencies could occur across a failure.
- ◆ File statistics and SEK information might not contain valid data unless the specified file is currently open.
- ◆ The system and file statistics might not contain valid data unless `STATS=YES`.
- ◆ The data set names for the Directory files and the Task Log File are not returned with SHOWX (element `FILEDSNM`). When running in non-Directory driven mode, the data set names are not returned for the above and for the Statistics File or System Log File on a SHOWX.
- ◆ The following tables present the valid SHOWX data list items in logical order within their groups.

Information about PDM system (DBMX group)

Data item	Format	Length	Returned value	Meaning
DBMXNAME	Character	8	---	Name of PDM.
DBMXTYPE	Character	2	ST	Single task PDM.
			MT	Multitask PDM.
DBMXATCH	Character	1	Y	PDM is in attached mode.
			N	PDM is not in attached mode.
DBMXCTRL	Character	1	Y	PDM is in central mode.
			N	PDM is not in central mode.
DBMXDLOC	Character	1	Y	PDM is in the same address space (ASID) as your application.
			N	PDM is not in same address space as your application.
DBMXRSTZ	Two 4-byte packed decimal numbers Julian date as YYYYDDDF and Time as OHHMSSF	8	---	Date and time the RSTAT command most recently reset the statistics counters to zero.
DBMXSYSN	Character	4	---	SUPRA Subsystem Name.
DBMXJOBN	Character	8	---	Real Job Name. Different from DBMXNAME if PDM name supplied.

Statistics gathered for PDM system

Data item	Format	Length	Returned value	Meaning
DBMXDMLR	Binary integer	4	---	DML READ commands.
DBMXDMLW	Binary integer	4	---	DML WRITE commands.
DBMXDMLA	Binary integer	4	---	DML ADD and DELETE commands.
DBMXDMLO	Binary integer	4	---	Other DML commands.
DBMXTHRC	Binary integer	4	---	Total number of held records.
DBMXCHRC	Binary integer	4	---	Current number of held records.
DBMXMHRC	Binary integer	4	---	Highest number of held records.
DBMXHLDL	Binary integer	4	---	Number of rejected requests for records (2.1.6 only).
DBMXSREC	Binary integer	4	---	Total stolen records (2.1.6). For 2.4, this is a File Statistic (FILEREC).
DBMXCIRC	Binary integer	4	---	Current number of shared records (2.4).
DBMXMIRC	Binary integer	4	---	Highest number of shared records (2.4).
DBMXFIDC	Binary integer	4	---	Total number of shared records (2.4) .
DBMXTSKS	Binary integer	4	---	Total number of tasks.
DBMXCCNT	Binary integer	4	---	Concurrent tasks at present time.
DBMXMCNT	Binary integer	4	---	Maximum concurrent tasks.
DBMXCMDS	Binary integer	4	---	Total number of commands.
DBMXSCNC	Binary integer	4	---	Sum of concurrent commands.
DBMXCCNC	Binary integer	4	---	Concurrent commands at this time.

Data item	Format	Length	Returned value	Meaning
DBMXMCNC	Binary integer	4	---	Maximum concurrent commands.
DBMXMCMT	Binary integer	8	---	Highest command elapsed time.*
DBMXPRTM	Binary integer	8	---	Total elapsed time for command processing.*
DBMXDBTM	Binary integer	8	---	Total amount of time the PDM was active.*
DBMXWTTM	Binary integer	8	---	Total amount of time the PDM waited.*
DBMXWAIT	Binary integer	4	---	Total number of times the PDM waited.
DBMXMXIO	Binary integer	4	---	Maximum I/O Area length.
DBMXMXAR	Binary integer	4	---	Maximum Argument Field Length.
DBMXMXPk	Binary integer	4	---	Maximum Packet Size.
2.4 and higher only				
DBMXMEMT	Binary integer	4	---	Total memory in K available to PDM.
DBMXMEMU	Binary integer	4	---	Used memory in K.
DBMXMEMH	Binary integer	4	---	Highest memory in K used by PDM.
DBMXMEMP	Binary integer	4	---	Percent used-to-available memory at which noncritical is released (threshold).
DBMXXEMT	Binary integer	4	---	Total XA memory in K available to PDM.
DBMXXEMU	Binary integer	4	---	Used XA memory in K.
DBMXXEMH	Binary integer	4	---	Highest XA memory in K used by PDM.

* Elapsed times are in clock units. Refer to IBM Principles of Operation.

Information about schema (SCHM group)

Data item	Format	Length	Returned value	Meaning
SCHMNAME	Character	8	---	Your schema name (spaces mean this item is not applicable).
SCHMNMLN	Binary integer	4	---	Length of your schema name.
SCHMLNAM	Character	8	---	Loaded schema name.

Information about environment description (ENVD group)

Data item	Format	Length	Returned value	Meaning
ENVDNAME	Character	8	---	Your environment description name (spaces mean this item is not applicable).
ENVDNMLN	Binary integer	4	---	Length of your environment description name.
ENVDLNAM	Character	8	---	Loaded environment description name.
ENVDSTAT	Character	1	Y	Statistics are being gathered.
			N	Statistics not being gathered.
ENVDTSLG	Character	1	Y	Task logging is active.
			N	Task logging is not active.
ENVDSYLG	Character	1	Y	System logging is active.
			N	System logging is not active.
ENVDLGBF	Character	1	Y	Before images are being logged to the System Log File.
			N	Before images are not being logged to the System Log File.

Data item	Format	Length	Returned value	Meaning
ENVDLGAF	Character	1	Y	After images are being logged to the System Log File.
			N	After images are not being logged to the System Log File.
ENVDLGAS	Character	1	Y	All sign-on and sign-off commands are being logged to the System Log File.
			N	Sign-on and sign-off commands are not logged to the System Log File.
ENVDLGUS	Character	1	Y	Sign-on commands are logged at sign-on, not when PDM processes first update command.
			N	Sign-on commands are being logged when PDM processes first update command.
ENVDLGAC	Character	1	Y	All functions are being logged to the System Log File.
			N	All functions are not being logged to the System Log File.
ENVDLGUC	Character	1	Y	Update functions are being logged to the System Log File.
			N	Update functions are not being logged to the System Log File.

Data item	Format	Length	Returned value	Meaning
ENVDLGIM	Character	1	Y	Before or after images are logged to the System Log File.
			N	Before or after images are not logged to the System Log File.
ENVDMXFL	Binary integer	4	---	Maximum number of files in system.
ENVDMXLR	Binary integer	4	---	Maximum Logical Record Length (LRECL) in any file. (Does not include system files.)
ENVDMXBZ	Binary integer	4	---	Maximum block size.
ENVDMXEL	Binary integer	4	---	Maximum data items in any file.
ENVDMXLK	Binary integer	4	---	Maximum linkpaths in any file.
ENVDMXKY	Binary integer	4	---	Maximum key size.
ENVDPNX	Character	1	I	OPENX and CLOSX commands are ignored.
			P	OPENX and CLOSX commands are processed.
			C	OPENX and CLOSX commands are checked against the current file mode list. No open or close commands are actually processed.
ENVDFEOV	Character	1	Y	FEOV macro is used to switch volumes of the System Log File during processing of ENDLG .
			N	The System Log File is closed and reopened.

Information about buffer pools (BPOL group)

Data item	Format	Length	Returned value	Meaning
BPOLNAME	Character	4	---	Buffer pool name.
BPOLBKSZ	Binary integer	4	---	Buffer pool block size.
BPOLDIRB	Binary integer	4	---	Number of direct buffers.
BPOLSEQB	Binary integer	4	---	Number of sequential buffers.
BPOLSEQT	Binary integer	4	---	Number of sequential threads.

Information about files (FILE group)

Data item	Format	Length	Returned value	Meaning
FILENAME	Character	4	---	File name.
FILEDDNM	Character	8	---	File DD name.
FILEDSNM	Character	44	---	FILE DSN (Data Set Name) in Directory.
FILEBPOL	Character	4	---	FILE buffer pool name.
FILETYPE	Character	1	P	Primary file.
			R	Related file.
			I	Index file.
			S	Statistics file.
			L	System Log File.
			T	Task Log File.
FILELTYP	Character	1	N	Lock record is native.
			C	Lock record is converted.
			O	Lock record is other.
FILEOPEN	Character	1	C	File is closed.
			R	File is open for read only.
			I	File is open for intent to update (IUPD).
			S	File is open for shared update (SUPD).
			E	File is open for exclusive update (EUPD).
FILEETSK	Character	16	---	Identifier of the task that owns the file for exclusive update (EUPD), composed of the interface name and task name.

Data item	Format	Length	Returned value	Meaning
FILEACES	Character	1	B	File is BDAM.
			E	File is VSAM ESDS.
			K	File is VSAM KSDS.
			S	File is BSAM.
			O	File is output (VSE typefile).
			W	File is work (VSE typefile).
FILECDSW	Character	1	Y	A coded file.
			N	Not a coded file.
FILECREC	Binary integer	4	---	Logical record length.
FILEBKSZ	Binary integer	4	---	Block size.
FILEBTRK	Binary integer	4	---	Blocks per track.
FILERBLK	Binary integer	4	---	Records per block.
FILERFIL	Binary integer	4	---	Records per file (not including the file lock record).
FILECVCI	Binary integer	4	---	Control interval size.
FILEPRNM	Binary integer	4	---	Prime number.
FILEDRSW	Character	1	Y	File is a Directory file.
			N	File is not a Directory file.
FILESPSW	Character	1	Y	File is a support file (TLF, SLF, STAT).
			N	File is not a support file.
FILEDVIC	Character	4	---	Device type.
FILECSKN	Character	4	---	Control key's secondary key name.

Statistics gathered for files

Data item	Format	Length	Returned value	Meaning
FILELRED	Binary integer	4	---	Number of times this file was logically read.
FILEIHIT	Binary integer	4	---	Total number of logical reads that found the desired block of data already in a storage buffer.
FILELWRT	Binary integer	4	---	Number of times this file was logically written to.
FILEMLTW	Binary integer	4	---	Number of logical updates to buffers that have already been updated.
FILEPRED	Binary integer	4	---	Number of times this file was physically read.
FILEPWRT	Binary integer	4	---	Number of times this file was physically updated.
FILEFWRT	Binary integer	4	---	Number of physical updates forced by a physical read.
FILELOGW	Binary integer	4	---	Number of times writes had to wait for a System or Task Log File block to be written first.
FILEIXUB	Binary integer	4	---	Number of blocks in use.
FILEIXTS	Binary integer	4	---	Total number of secondary key control records.
FILEIXUS	Binary integer	4	---	Number of secondary key control records in use.

2.4 and higher only

Data item	Format	Length	Returned value	Meaning
FILEEMBR	Binary integer	4	---	Number of records not held because of deadly embrace.
FILEGDIM	Binary integer	4	---	Number of records successfully held with no waiting.
FILEGDWT	Binary integer	4	---	Number of records successfully held after waiting.
FILEHDIM	Binary integer	4	---	Number of records not held with no waiting.
FILEHDWT	Binary integer	4	---	Number of records not held after waiting.
FILESREC	Binary integer	4	---	Number of records stolen (only if task logging is not active).
FILELSBI	Character	1	Y	Suppress before images.
			N	Log before images to System Log.
FILELSAI	Character	1	Y	Suppress after images.
			N	Log after images to System Log.
FILELSFN	Character	1	Y	Suppress function (command) images.
			N	Log function images to System Log.
FILELTLG	Character	1	Y	Suppress logging to Task Log.
			N	Perform logging to Task Log.

Information about physical fields (FELD group)

Data item	Format	Length	Returned value	Meaning
FELDNAME	Character	8	---	Field name.
FELDLENG	Binary integer	4	---	Field length.
FELDDISP	Binary integer	4	---	Field displacement.
FELDCODE	Character	2	---	Record code (high values mean the field is valid for all records in the file).
FELDROOT	Character	1	Y N	Root data item. Not the root data item.
FELDOVLY	Character	1	Y N	Overlay data item. Not the overlay data item.
FELDKEYS	Character	1	Y N	Control key. Not the control key.
FELDCDSW	Character	1	Y N	Code data item. Not a code data item.
FELDPLNK	Character	1	Y N	Primary linkpath. Not a primary linkpath.
FELDRLNK	Character	1	Y N	Related linkpath. Not a related linkpath.
FELDKNAM	Character	8	---	The field name for the key (spaces mean this item is not applicable).
FELDKFNM	Character	4	---	The name of the primary file (spaces mean this item is not applicable).

Data item	Format	Length	Returned value	Meaning
FELDKDSP	Binary integer	4	---	Key displacement (high values mean this item is not applicable).
FELDKLEN	Binary integer	4	---	Length of key (high values mean this item is not applicable).
2.4 and higher only				
FELDNDEC	Binary integer	4	---	Number of decimal places.
FELDNVAL	Character	32	---	Directory null value, blank padded on right.
FELDNLEN	Binary integer	4	---	Length of null value.
FELDSIGN	Character	1	Y	Signed data.
			N	Not signed data.
FELDTYPE	Character	1	B	Binary field.
			C	Character field.
			F	Floating point field.
			K	Kanji field.
			P	Packed decimal field.
			Z	Zoned decimal field.

Information about interfaces (IFCE group)

Data item	Format	Length	Returned value	Meaning
IFCENAME	Character	8	---	Interface identifier.
IFCECOND	Character	1	C	The interface is connected.
			N	The interface is not active.
IFCETYPE	Character	1	B	STST interface.
			S	STMT interface.
			C	CICS Connector.

Information about tasks (TASK group)

Data item	Format	Length	Returned value	Meaning
TASKNAME	Character	8	---	Task identifier.
TASKEXST	Character	32	---	User extended status from SINON .
TASKCMID	Binary integer	4	---	Most recent COMIT ID .
TASKIFCE	Character	8	---	Interface identifier.
TASKCMST	Binary integer	8	---	Time of most recent COMIT (stored clock value).
TASKCOND	Character	1	A	Task is active (signed on).
			N	Task is not active. (It is available for restart.)
TASKRSMD	Character	8	NORMAL	Restart mode. Task is considered restartable.
			NONE	Task is not restartable.
TASKACRD	Character	1	Y	Access is READ.
			N	Access is not READ.
TASKACUP	Character	1	Y	Access is UPDATE.
			N	Access is not UPDATE.
TASKACRV	Character	1	Y	Access is RECOVER.
			N	Access is not RECOVER.
TASKACLK	Character	1	Y	Access is LINKS.
			N	Access is not LINKS.

Information about secondary key IDs (SKID group)

Data item	Format	Length	Returned value	Meaning
SKIDNAME	Character	8	---	Secondary key name (ffffSKxx).
SKIDINDX	Character	4	---	Index file for secondary key.
SKIDKEYL	Binary integer	4	---	Length of secondary key, external definition.
SKIDPTRL	Binary integer	4	---	Pointer length, external; RRN or key.
SKIDDENS	Binary integer	4	---	Pointer density.
SKIDUTMN	Character	1	Y	Utility maintenance in progress.
			N	NOT in utility maintenance.
SKIDOPTS	Character	4	ABND	SKID option is abend.
			CONT	SKID option is continue.
			OPER	SKID option is operator option.
SKIDVERS	Character	1	---	Version # of C\$TASKCR control block; shows which version of PDM used to format/populate. Used by Support. RRNS in time order sequence.
SKIDLGRD	Binary integer	4	---	Logical READS.
SKIDLGUP	Binary integer	4	---	Logical UPDATES.
SKIDSPLT	Binary integer	4	---	Total number of SPLITS.
SKIDSPLM	Binary integer	4	---	Total number of SPLITS involving multiple levels.
SKIDSPLR	Binary integer	4	---	Total number of SPLITS involving ROOT.

Data item	Format	Length	Returned value	Meaning
SKIDSTAT	Character	4	POPU	Secondary key is populated and usable.
			PURG	Secondary key is depopulated and purged.
			DPOP	Secondary key is depopulated and not usable.
SKIDUNIQ	Binary integer	4	---	Number of unique keys.
SKIDNPTR	Binary integer	4	---	Number of pointers.
SKIDNBLK	Binary integer	4	---	Number of blocks in tree.
SKIDNLVL	Binary integer	4	---	Number of levels (tree height).
SKIDLLBK	Binary integer	4	---	Number of low level blocks (tree width).
2.4 and higher only				
SKIDKYIL	Binary integer	4	---	Length of SEK, internal storage.
SKIDPORD	Character	1	S	Sorted pointer ordering.
			F	FIFO pointer ordering.
SKIDPTIL	Binary integer	4	---	Pointer length, internal storage; RRN or key size.
SKIDPTYP	Character	1	D	Direct pointer.
			I	Indirect pointer.
SKIDSTYP	Character	1	Y	Data sensitive.
			N	Not data sensitive.
SKIDTRAN	Character	1	Y	Data translate.
			N	No data translate.
SKIDUNQK	Character	1	Y	Unique secondary key.
			N	Not unique.

Information about SEK physical fields (SKPF group)

Data item	Format	Length	Returned value	Meaning
SKPFNAME	Character	8	---	Physical field name.
SKPFKYCD	Character	2	---	Key code for this SKID.
SKPFPFCD	Character	2	---	Record code for this physical field. The value is FF if field is in base portion; key code if field is in variable portion.
2.4 and higher only				
SKPFILEN	Binary integer	4	---	Index storage length of this key part.

SINOF

The SINOF (Sign-off) command performs task termination activities in a multitask environment, or PDM and task termination activities in a single-task environment.

SINOF,*status*,end

or

SINOF,*status,task*,end (TIS and TOTAL compatibility)

status

Description *Required.* Points to a field into which the PDM places a status code indicating the result of the command.

Format 4-byte field

Consideration If the command fails or if the status code indicates some special condition other than failure, your program should include logic to handle and possibly correct the situation. For a list of all status codes, refer to the [SUPRA Server PDM Messages and Codes Reference Manual \(RDM/PDM Support for OS/390 & VSE\)](#), P26-0126.

task

Description *Optional.* This field is not examined; however, it must contain a valid address.

end

Description *Required.* Points to a field that delimits the parameter list.

Format END.

General considerations

- ◆ The 4-parameter SINOF is presented only for compatibility with TIS and TOTAL.



We recommend you use the 3-parameter SINOF.

- ◆ Any DML command the task issues after the SINOF command, unless another **SINON** command, returns an error status code.
- ◆ In a single-task operating mode, the SINOF command closes and unlocks the files that were open for that task and writes their database buffers. If a file is open in EUPD and is not KSDS, the SINOF command closes it partially (it changes the mode to IUPD). For KSDS files open in EUPD, the close is complete. (Do not depend on the SINOF when using task logging; always use **COMIT** or **RESET** following opens and closes to prevent locked files.)
- ◆ In all operating modes where task logging is active, a SINOF writes a **COMIT** record to the Task Log File. A SINOF also frees internal resources. In a single-task operating mode, the SINOF command also writes the termination statistics record to the Statistics File (if active).

SINON

The SINON (Sign-on) command performs task initialization activities in a multitask environment, or PDM and task initialization activities in a single-task environment.

SINON, *status*[, *option-list*], *end*

status

Description *Required.* Points to a field into which the PDM places a status code indicating the result of the command.

Format 4-byte field

Considerations

- ◆ If the command fails or if the status code indicates some special condition other than failure, your program should include logic to handle and possibly correct the situation. For a list of all status codes, refer to the *SUPRA Server PDM Messages and Codes Reference Manual (RDM/PDM Support for OS/390 & VSE)*, P26-0126.
- ◆ Code your program to handle the following status code:
 - RSTR Returned if you are reexecuting your program after a failure if you had been issuing **COMIT**s. You can issue a **RESET** to retrieve commit data you saved. This status code is possible only if task logging is active.
- ◆ If your program has not issued a **COMIT** and then either it or the PDM abends, a **** status is returned to the SINON when you restart your program.

option-list

Description	<i>Optional.</i> Points to a field containing a list of SINON options that may affect the task's processing.
Format	<i>option-list,END.</i>
Options	If this parameter is used, the pointed-to field must contain the keyword END. even if it contains no options. The option list can contain one or more of the options described in the following. Use a comma between each option you code.

REALM=(SCHEMA=*schema-id*,ENVDESC=*env-desc-id*)

Description	<i>Optional.</i> A field identifying the 1-8 byte names (padded with spaces if desired) of your schema and environment description for which this SINON is used.
Default	The schema and environment description names in the REALM parameter of the CSIPARM file.
Consideration	If you code this parameter for multitask operating mode, the realm you identify must match the active schema and environment description.

DBM=*pdm-name*

Description	<i>Optional.</i> A field identifying the 1-8 byte name of the PDM with which this SINON communicates.
Default	The PDM name in the CSIPARM file, if specified. If not specified in CSIPARM, the current job name in OS/390 and VSE, or the virtual machine name in CMS.

Considerations

- ◆ If you code this parameter for a single-task or multitask PDM for batch, it overrides the PDM name in the CSIPARM file (if PDM name is not supplied in the CSIPARM file, this name overrides the current job name).
- ◆ If you code this parameter for a task communicating through a CICS connector, the name must match the PDM name in the CSIPARM file.

INTERFACE=interface-id

Description *Optional.* A field identifying the 1-8 character interface name from which this SINON is issued.

Default The interface name in the CSIPARM file, if specified. If not specified in CSIPARM, the current job name.

Considerations

- ◆ If you code this parameter for a task communicating through a CICS connector, it must match the interface identifier supplied in the CSIPARM file. If there is no interface identifier in the CSIPARM file, it must match the current job name.
- ◆ In an STST or STMT application task, you can also change the interface identifier on the CSIPARM file.

TASK=task-id

Description *Optional.* A field identifying the 1-8 alphanumeric character identifier for this task.

Default *Interface-ID*

Considerations

- ◆ If you code this parameter for a task communicating through a CICS connector, it must match the task identifier generated by the CICS connector.
- ◆ Multiple STMT interfaces may have the same task-ID; however, the interface names must be unique.

RESTART = $\left\{ \begin{array}{l} \text{NORMAL} \\ \text{NONE} \end{array} \right\}$

Description *Optional.* A field indicating whether the task is to be treated as restartable after a task or PDM failure in a task logging environment.

Default The RESTART option of the CSIPARM file.

Options NORMAL The task is restartable if it has issued at least one **COMIT** command.

 NONE The task is not restartable.

Considerations

- ◆ Coding this parameter overrides the RESTART mode coded in the CSIPARM file.
- ◆ You cannot code RESTART=NONE for STST tasks; you must code RESTART=NORMAL or omit the option.
- ◆ When a task or system abnormally terminates, whether the task is restartable depends on its environment. If the environment does not include Task Level Recovery (TLR), no task is restartable, and all tasks must find their restart points.
- ◆ If the task is restartable, the PDM backs out the task's updates to the most recent commit point and leaves its files locked. When the task signs on again, it receives an RSTR status and can retrieve its commit text from the Task Log File with a **RESET**. If not restartable, the PDM backs out its updates to the last commit point, frees its resources, and signs off the task. When the task signs on again, it receives an **** status.

ACCESS=(*access-option-list*)

- Description** *Optional.* A field identifying the access mode to be supported (type of DML allowed).
- Default** Access mode in active user environment description.
- Options** UPDATE= NO Controls whether the task can
YES Read, write, add, and delete.
RECOVR= NO Controls whether the task can
YES Execute a **WRITD** in addition to having UPDATE access.
- Consideration** Cincom does not support RECOVR access for anything other than the PDM Recover/Restore Utilities. Do not use this PDM access for user applications. This access disables some automatic processing and may result in loss of database integrity.

USER=(*user-data*)

Description *Optional.* A field containing a character string of user data. The task can use the string with PDM exits.

Considerations

- ◆ The user data can be 1-8 alphanumeric, \$, #, or @ characters. It cannot begin with a number. If less than 8 characters, the PDM pads it with blanks on the right.
- ◆ This field is normally used only by the DBA in a special program. For example, use this parameter to pass security data to the command initialization exit. For PDM exit information, refer to the *SUPRA Server PDM and Directory Administration Guide (OS/390 & VSE)*, P26-2250.

END.

Description *Required.* A field that delimits the *option-list*.

end

Description *Required.* Points to a field that delimits the parameter list.

Format END.

General considerations

- ◆ The SINON command must be the first PDM function executed in a task, or the PDM returns an error status code.
- ◆ For an STST application, you can issue a SINON command (after the **SINOF** command has been executed) to change the value of realm. (In a multitask operating mode or a single-task operating mode, you can issue SINON after SINOF, but only STST allows you to change the value of realm.)
- ◆ In a task-logging environment, the SINON causes an automatic **COMIT** record on the Task Log File. The commit ID is null.

SINON (CICS compatibility)

The SINON (Sign-on) command performs the task initialization activities. This form of the SINON command is presented for compatibility with previous TIS and TOTAL CICS programs. This discussion does not present all the detail information about each parameter. Refer to the *TIS Recovery System For CICS/VS Interface Guide*, P16-7290, *TOTAL CICS (With DTB) Reference Manual*, P03-1001, or the *Series 80 Recovery System for CICS/VS Interface Guide*, P03-1002, for those details.

SINON,status,csaddr,end

status

Description *Required.* Points to a field into which the PDM places a status code indicating the result of the command.

Format 4-byte field

Considerations

- ◆ If the command fails or if the status code indicates some special condition other than failure, your program should include logic to handle and possibly correct the situation. For a list of all status codes, refer to the *SUPRA Server PDM Messages and Codes Reference Manual (RDM/PDM Support for OS/390 & VSE)*, P26-0126.
- ◆ Code your program to handle the following status code:
 - RSTR Returned if you are reexecuting your program after a failure if you had been issuing **COMIT**s. You can issue a **RESET** to retrieve commit data you had saved. This status code is possible only if task logging is active.
- ◆ If your program has not issued a **COMIT** and then either it or the PDM abends, a **** status is returned on the SINON when you restart your program.

csaddr

Description *Required.* Points to a field containing the Common System Area (CSA) address to use. The CSA is a standard CICS control block.

Consideration SUPRA Server does not require csaddr from a CICS application program. The PDM does not examine this parameter; however, it must point to a valid address.

end

Description *Required.* Points to a field that delimits the parameter list and indicates the record holding function.

Format END.

Consideration Other forms of this field, (such as RLSE) are not supported.

SINON (TIS 1.x compatibility)

The SINON (Sign-on) command performs task initialization activities in a multitask environment or system and task initialization in a single-task environment. This form of the SINON command is presented for compatibility with TIS.

SINON, status, access, bootmod, task, options, end

status

Description *Required.* Points to a field into which the PDM places a status code indicating the result of the command.

Format 4-byte field

Considerations

- ◆ If the command fails or if the status code indicates some special condition other than failure, your program should include logic to handle and possibly correct the situation. For a list of all status codes, refer to the *SUPRA Server PDM Messages and Codes Reference Manual (RDM/PDM Support for OS/390 & VSE)*, P26-0126.
- ◆ Code your program to handle the following status code:
 - RSTR Returned if you are reexecuting your program after a failure if you had been issuing **COMIT**s. You can issue a **RESET** to retrieve commit data you had saved. This status code is possible only if task logging is active.
- ◆ If your program has not issued a **COMIT** and then either it or the PDM abends, a **** status is returned on the SINON when you restart your program.

access

Description *Required.* Points to a field identifying the access mode to be supported (type of DML allowed).

Options

RDONLY	Task intends to issue only read commands.
UPDATE	Task intends to issue writes, adds, and deletes in addition to reads.
RECOVR	Task intends to issue WRITD .

Considerations

- ◆ The PDM examines access to determine if the SINON is a compatibility SINON.
- ◆ Although this parameter must be a valid option, the user environment description determines actual access, not this parameter.

bootmod

Description *Required.* Points to a field containing the bootmod to use (a TIS entity).

Consideration SUPRA Server does not require a bootmod from application programs. The PDM does not examine this parameter; however, it must point to a valid address.

task

Description *Required.* Points to a field containing the name of the task.

Considerations

- ◆ For batch tasks, the PDM uses this field for the task name.
- ◆ For CICS, if this *task* parameter does not match the task name generated by CICS, you receive an error status code.

options

Description *Required.* Points to a field containing the TIS options to use.

Consideration SUPRA Server does not require these options from an application program. The PDM does not examine this parameter; however, it must point to a valid address.

end

Description *Required.* Points to a field that delimits the parameter list.

Format END.

SINON (TOTAL compatibility)

The SINON (sign-on) command performs task initialization activities in a multitask environment, or system and task initialization in a single-task environment. This form of the SINON command is presented for compatibility with TOTAL.

SINON, status, access, dbmod, task, options, end

status

Description *Required.* Points to a field into which the PDM places a status code indicating the result of the command.

Format 4-byte field

Considerations

- ◆ If the command fails or if the status code indicates some special condition other than failure, your program should include logic to handle and possibly correct the situation. For a list of all status codes, refer to the *SUPRA Server PDM Messages and Codes Reference Manual (RDM/PDM Support for OS/390 & VSE)*, P26-0126.
- ◆ Code your program to handle the following status code:
 - RSTR Returned if you are reexecuting your program after a failure if you had been issuing **COMIT**s. You can issue a **RESET** to retrieve commit data you had saved. This status code is possible only if task logging is active.
- ◆ If your program has not issued a **COMIT** and then either it or the PDM abends, a **** status is returned on the SINON when you restart your program.

access

Description *Required.* Points to a field identifying the access mode to be supported (type of DML allowed).

Options

RONLY	Task intends to issue only read commands.
UPDATE	Task intends to issue writes, adds, and deletes in addition to reads.
RECOVR	Task intends to issue WRITD .

Considerations

- ◆ The PDM examines access to determine if the SINON is a compatibility SINON.
- ◆ Although this parameter must be a valid option, the user environment description determines actual access, not this parameter.

dbmod

Description *Required.* Points to a field containing the TOTAL DBMOD to use.

Consideration SUPRA Server does not require a DBMOD name from application programs. The PDM does not examine this parameter; however, it must point to a valid address.

task

Description *Required.* Points to a field containing the name of the task.

Considerations

- ◆ For batch tasks, the PDM uses this field for the task name.
- ◆ For CICS, if this *task* parameter does not match the task name generated by CICS, you receive an error status code.

options

Description *Required.* Points to a field containing the TOTAL options to use.

Consideration SUPRA Server does not require these options from an application program. The PDM does not examine this parameter; however, it must point to a valid address.

end

Description *Required.* Points to a field that delimits the parameter list.

Format END.

WRITD

The WRITD command writes an entire logical record into the specified relative record location. For database recovery, this command places a before or after image from the System Log file into the location occupied before the failure.

WRITD, status, file, reference, data-area, end

status

Description *Required.* Points to a field into which the PDM places a status code indicating the result of the command.

Format 4-byte field

Consideration If the command fails or if the status code indicates some special condition other than failure, your program should include logic to handle and possibly correct the situation. For a list of all status codes, refer to the [SUPRA Server PDM Messages and Codes Reference Manual \(RDM/PDM Support for OS/390 & VSE\)](#), P26-0126.

file

Description *Required.* Identifies the primary or related file to be acted upon. You can define a field containing the name or you can use the actual file name as literal text in the CALL statement.

Format 4 alphanumeric characters; the first character must be alphabetic

Considerations

- ◆ The file must be accessed via ESDS or BDAM. If it is KSDS, the PDM returns a FUNC error status code.
- ◆ The file cannot be a PDM log file or the Statistics File. If it is, the PDM returns an error status code.
- ◆ The requesting task must have opened the file for exclusive update. If not, the PDM returns an error status code.

reference

Description	<i>Required.</i> Points to the 4-byte area containing the relative record number (RRN) of the record to be written.
Format	4 alphanumeric characters or a binary fullword

data-area

Description	<i>Required.</i> Points to a field containing the image to be written.
Format	Variable length, depending on the logical record length of the file as defined on the Directory.

end

Description	<i>Required.</i> Points to a field that delimits the parameter list.
Format	END.

General considerations

- ◆ If task logging is active when you issue a WRITD, the PDM returns an error status code.
- ◆ If you did not define the access mode as RECOVER in your environment description, the PDM returns an error status code. Cincom does not support RECOVER access for anything other than PDM Recover and Restore utilities. RECOVER access disables some automatic processing and may result in loss of database integrity.
- ◆ Exercise extreme care when using WRITD since the PDM only checks to determine whether the file actually exists, and whether the file is large enough for the record location specified; it checks nothing else.

WRITM

The WRITM (Write-Primary) command updates a primary record. The PDM locates the record to be updated as identified by the control key. The PDM moves the data elements in the data area to the record according to the *data-list* parameter and updates the record.

WRITM, status, file, control-key, data-list, data-area, end

status

Description *Required.* Points to a field into which the PDM places a status code indicating the result of the command.

Format 4-byte field

Considerations

- ◆ If the command fails or if the status code indicates some special condition other than failure, your program should include logic to handle and possibly correct the situation. For a list of all status codes, refer to the *SUPRA Server PDM Messages and Codes Reference Manual (RDM/PDM Support for OS/390 & VSE)*, P26-0126.
- ◆ Code your program to handle the following status codes:
 - HELD When accessing a database record currently being held by another task, the PDM waits until a user-defined time-out counter expires before it reexamines the status of the lock request. If the record is still HELD, the PDM returns the HELD status to the application. The application may elect to terminate the task or reissue the command. If you choose recycle logic for HELD statuses, you should implement an upper limit to the mechanism in the program logic.
 - EMBR As with the HELD status, the PDM may return EMBR when another task is using the database record. Distinguishing the EMBR from the HELD status is the occurrence of a deadly embrace. Deadly embrace occurs when two tasks, already owners of a resource (database record), attempt to gather each other's resources. To handle an EMBR status, restart the logical unit of work. Alternatively, you can introduce a retry mechanism with an upper limit retry count in your program logic.

file

Description *Required.* Identifies the primary file to be acted upon. You can define a field containing the name or you can use the actual file name as a literal in the CALL statement.

Format 4 alphanumeric characters; first character must be alphabetic

Consideration The file must be a primary file.

control-key

Description *Required.* Points to a field containing the key of the primary record to be processed. The PDM uses this key to locate a primary record.

Format Variable length as defined on the Directory

Consideration During the command processing, if the *control-key* parameter does not match the corresponding field in your data area, a status code informs you of the failure. To avoid this, you should name the control key field in the data area rather than define a separate field.

data-list

Description *Required.* Points to a field containing a list of data items. This list acts as a map of the layout of the data area. Compose this list using data names (physical fields) defined on the Directory.

Format *dataitem1,dataitem2,...dataitemn,END.*

Considerations

- ◆ The commas between the entries are optional and only serve as separators; be consistent whether you use them or not.
- ◆ The data list can include the names of data items and control keys. Do not name linkpaths or the root field. Do not list any name twice. If a name is not accepted, the PDM returns an error status code.
- ◆ You can list the data names in any order. They are processed in the order listed, not in the order defined on the Directory.
- ◆ **VSE** When using a cross-address space central PDM in VSE/AF with XPCC=YES, the data area mapped by this *data-list* parameter is limited by the CSIPARM MAXIO value. This governs DML having no *length* parameter. Refer to the *SUPRA Server PDM and Directory Administration Guide (OS/390 & VSE)*, P26-2250, for CSIPARM information.

data-area

Description *Required.* Points to a field to be used as an output area for the data items named in the data list.

Format The structure and characteristics of the data area must conform exactly to the Directory definition of the data items (physical fields) named in the data list.

Considerations

- ◆ The data area must be large enough to hold values for all data items named in the data list.
- ◆ The data area and the data list have corresponding fields. The data list holds names, and the data area holds a value for each of those names. If you name the control key in the data list, its value must be equal in both the data area and control key.

end

Description *Required.* Points to a field that delimits the parameter list.

Format END.

General considerations

- ◆ Do not use this command to attempt to change the control key. If you must change the value of the control key in the primary record, your task must read the record, delete it, then add it with a new control key. Also, if there is a chain of related records subordinate to the primary record, your task must delete the related chain before it can delete the primary record.
- ◆ With the changing of the record, the PDM maintains all populated secondary keys in the specified data file accordingly.
- ◆ In a multitask operating mode or if task logging is active, the record must have been read and held when this command is executed.

WRITV

The WRITV (Write Related) command updates the record whose RRN is in the *reference* field. Your task determines the RRN by executing a preceding read DML command. The PDM moves the data elements in the data area to the record according to the *data-list* parameter and updates the record.

WRITV, status, file, reference, linkpath, control-key, data-list, data-area, end

status

Description *Required.* Points to a field into which the PDM places a status code indicating the result of the command.

Format 4-byte field

Considerations

- ◆ If the command fails or if the status code indicates some special condition other than failure, your program should include logic to handle and possibly correct the situation. For a list of all status codes, refer to the *SUPRA Server PDM Messages and Codes Reference Manual (RDM/PDM Support for OS/390 & VSE)*, P26-0126.
- ◆ Code your program to handle the following status codes:
 - HELD When accessing a database record currently being held by another task, the PDM waits until a user-defined time-out counter expires before it reexamines the status of the lock request. If the record is still HELD, the PDM returns the HELD status to the application. The application may elect to terminate the task or reissue the command. If you choose recycle logic for HELD statuses, you should implement an upper limit to the mechanism in the program logic.
 - EMBR As with the HELD status, the PDM may return EMBR when another task is using the database record. Distinguishing the EMBR from the HELD status is the occurrence of a deadly embrace. Deadly embrace occurs when two tasks, already owners of a resource (database record), attempt to gather each other's resources. To handle an EMBR status, restart the logical unit of work. Alternatively, you can introduce a retry mechanism with an upper limit retry count in your program logic.

file

- Description** *Required.* Identifies the related file to be acted upon. You can define a field containing the name or you can use the actual file name as a literal in the CALL statement.
- Format** 4 alphanumeric characters; the first character must be alphabetic
- Consideration** The file must be a related file, or the PDM returns an error status code.

reference

- Description** *Required.* Points to a field identifying the RRN of the specific record to be updated. You place the RRN in this field to tell the PDM which record to process.
- Format** 4 alphanumeric characters or a binary fullword
- Options** *rrrr* Identifies the RRN of the record to be updated.
- Consideration** WRITV does not change the RRN of the record.

linkpath

- Description** *Required.* Points to a field containing the name of the linkpath as defined on the Directory. This parameter indicates which related record chain is being processed.
- Format** *ppppLKxx* where *pppp* identifies the name of an associated primary file, LK is a constant, and *xx* are the last 2 characters of the linkpath name as defined on the Directory.
- Consideration** If you specify an invalid linkpath, the PDM returns a status code.

control-key

Description *Required.* Points to a field containing the key of the primary record to process. The PDM uses this key to link a related record to a primary record.

Format Variable length as defined on the Directory

Considerations

- ◆ During the command processing, if the control key does not match the corresponding field in your data area, an error status code informs you of the failure.



To avoid this, we recommend that the *control-key* parameter point to the data area rather than a separate field.

- ◆ This parameter specifies the key of the controlling primary record.

data-list

Description *Required.* Points to a field containing a list of data items. This list acts as a map of the layout of the data area. Compose this list using data names (physical fields) defined on the Directory.

Format dataitem1,dataitem2,...dataitemn,END.

Considerations

- ◆ The commas between the entries are optional and only serve as separators; be consistent whether you use them or not.
- ◆ The data list can include the names of data items, control keys, and record codes. Do not name linkpaths. Do not list any name twice. If a name is not accepted, the PDM returns an error status code.
- ◆ You can list the data names in any order. They are processed in the order listed, not in the order defined on the Directory. However, for coded records, the record code must be first in the list. If not, unpredictable results occur. See “[Data list parameter keywords](#)” on page 62 for coded data list construction rules.
- ◆ **VSE** When using a cross-address space central PDM in VSE/AF with XPCC=YES, the data area mapped by this *data-list* parameter is limited by the CSIPARM MAXIO value. This governs DML having no *length* parameter. Refer to the [SUPRA Server PDM and Directory Administration Guide \(OS/390 & VSE\)](#), P26-2250, for CSIPARM information.

data-area

- Description** *Required.* Points to a field to be used as an output area for the data items named in the data list.
- Format** The structure and characteristics of the data area must conform exactly to the Directory definition of the data items (physical fields) named in the data list.

Considerations

- ◆ The data area must be large enough to hold values for all data items named in the data list.
- ◆ The data area and the data list have corresponding fields. The data list holds names, and the data area holds a value for each of those names. If you name the control key in the data list, its value must be equal in both the data area and control key.

end

- Description** *Required.* Points to a field that delimits the parameter list.
- Format** END.

General considerations

- ◆ Do not attempt to modify linkpaths, control keys or record codes because the WRITV command will not perform link maintenance; however, you can include record codes and keys in the data- list and *data-area* parameters. To modify linkpaths, control keys, or record codes, use the **ADDVR** command.
- ◆ In a multitask operating mode or if task logging is active, the record must have been read and held when this command is executed.

4

Programming examples

This chapter contains Data Manipulation Language (DML) coding examples showing common application programming operations. The examples are written in COBOL, but you can use any general purpose computer language that supports a CALL statement.

The program structures and COBOL coding techniques in the following examples are for illustration only and are not intended to show sophistication of programming.

The example sequences use ANS-COBOL reserved words as user-supplied names in some instances (e.g., STATUS). Before running any of the sequences presented here, compare the user-supplied names to an ANS-COBOL reserved word list to make sure they have no preassigned meanings.

Example of IDENTIFICATION, ENVIRONMENT, and DATA divisions

```
IDENTIFICATION DIVISION.
```

```
PROGRAM-ID. EXAMPLE.
```

Comment

The IDENTIFICATION, ENVIRONMENT, and DATA divisions listed here apply to all of the examples in this chapter.

```
ENVIRONMENT DIVISION.
```

```
INPUT-OUTPUT SECTION.
```

```
SELECT TAPE-IN ASSIGN TO SYS006-UT-2400-S.
```

```
SELECT TAPE-OUT ASSIGN TO SYS007-UT-2400-S.
```

Comment Input and output tapes must be set up according to device type, operating system, etc.

```

DATA DIVISION.
FILE SECTION.
FD TAPE-IN
RECORDING MODE IS F
LABEL RECORDS ARE STANDARD
DATA RECORD IS TAPE-IN-REC.

01 TAPE-IN-REC.
   05 TAPE-CODE           PIC X(2).
   05 TAPE-CONTROL       PIC X(7).
   05 TAPE-NAME          PIC X(20).
   05 TAPE-ADDRESS       PIC X(25).
   05 TAPE-CITY          PIC X(20).
   05 TAPE-STATE         PIC X(2).
   05 TAPE-ZIP           PIC X(5).
   05 TAPE-VALUE         PIC S9(5)V99.
   05 TAPE-TERMS         PIC X(10).
   05 TAPE-DISC          PIC SV99.

FD TAPE-OUT
RECORDING MODE IS F
LABEL RECORDS ARE STANDARD
DATA RECORD IS TAPE-OUT-REC.

01 TAPE-OUT-REC.
   05 TO-CODE            PIC X(2).
   05 TO-CONTROL         PIC X(7).
   05 TO-NAME            PIC X(20).
   05 TO-ADDRESS         PIC X(25).
   05 TO-CITY            PIC X(20).
   05 TO-STATE           PIC X(2).

```

```

05 TO-ZIP          PIC X(5) .
05 TO-VALUE       PIC X9(5)V99 .
05 TO-TERMS       PIC X(10) .
05 TO-DISC        PIC SV99 .
05 TAPE-RQLOC     PIC X(4) .

```

WORKING-STORAGE SECTION.

01 PDM-PARAMETERS.

```

05 FUNCTION        PIC X(5)          VALUE SPACES .
05 PRIMARY-FILE   PIC X(4)          VALUE 'CUST' .
05 RELATED-FILE   PIC X(4)          VALUE 'CORD' .
05 REFERENCE      PIC X(4)          VALUE SPACES .
05 LINKPATH       PIC X(8)          VALUE 'CUSTLKOR' .
05 QUALIFIER      PIC X(30)         VALUE SPACES .
05 ENDP           PIC X(4)          VALUE 'END.' .

```

01 REALM.

```

05 FILLER         PIC X(6)          VALUE 'REALM=' .
05 ENTRIES       PIC X(30)         VALUE SPACES .

```

01 PDM-STATUS.

```

05 STATUS        PIC X(4)          VALUE SPACES .

```

01 PDM-MISC.

```

05 ABEND-DUMP-SWITCH PIC X          VALUE SPACE .
05 IO-SWITCH       PIC X          VALUE SPACE .
05 X-PARM          PIC X          VALUE SPACE .
05 TAPE-COUNT      PIC 9(4)        VALUE ZERO .

```

01 PRIMARY-DATA-LIST.

```

05 FILLER         PIC X(8)          VALUE 'CUSTCTRL' .
05 FILLER         PIC X(8)          VALUE 'CUSTNAME' .
05 FILLER         PIC X(8)          VALUE 'CUSTADDR' .
05 FILLER         PIC X(8)          VALUE 'CUSTCITY' .
05 FILLER         PIC X(8)          VALUE 'CUSTSTXX' .
05 FILLER         PIC X(8)          VALUE 'CUSTZIPC' .
05 FILLER         PIC X(4)          VALUE 'END.' .

```

```

01 PRIMARY-DATA-AREA.
   05 CONTROL-KEY          PIC X(7)          VALUE SPACES.
   05 CUSTOMER-NAME       PIC X(20)         VALUE SPACES.
   05 CUSTOMER-ADDRESS    PIC X(25)         VALUE SPACES.
   05 CUSTOMER-CITY       PIC X(20)         VALUE SPACES.
   05 CUSTOMER-STATE      PIC X(2)         VALUE SPACES.
   05 CUSTOMER-ZIP-CODE   PIC 9(5)         VALUE ZEROES.

01 RELATED-DATA-LIST.
   05 FILLER              PIC X(8)         VALUE 'CORDCUST'.
   05 FILLER              PIC X(8)         VALUE 'CORDVALU'.
   05 FILLER              PIC X(8)         VALUE 'CORDTERM'.
   05 FILLER              PIC X(8)         VALUE 'CORDDISC'.
   05 FILLER              PIC X(4)         VALUE 'END.'.

01 RELATED-DATA-AREA.
   05 CORD-CUSTOMER       PIC X(7)         VALUE SPACES.
   05 CORD-VALUE          PIC S9(5)V99     VALUE ZERO COMP-3.
   05 CORD-TERMS          PIC X(10)        VALUE SPACES.
   05 CORD-DISC-RATE      PIC SV99         VALUE .02 COMP-3.

01 COMIT-AREA.
   05 COMIT-ID            PIC X(4)         VALUE 'ASGN'.
   05 COMIT-LENGTH        PIC 9(8)         VALUE 107 COMP.
   05 COMIT-DATA.
      10 COMIT-TAPE-IN-REC PIC X(105).
      10 COMIT-TAPE-COUNT  PIC 9(4)         VALUE ZERO.
      10 COMIT-IO-SWITCH  PIC X           VALUE SPACE.

```

Example of read-only environment

Here is an analysis of the example program that follows:

1. PROGRAM INITIALIZATION—The first step of a task must be a **SINON**. If the **SINON** fails, the task terminates with an error message by skipping to step 7.
2. PROGRAM-OPEN—This task opens the two PDM files processed in subsequent steps. You can omit this command in certain environments. If the **OPENX** fails, the task terminates with an error message by skipping to step 8. The task then opens the sequential input and output files.
3. PROGRAM-MAINLINE—The task reads an input record from sequential file TAPE-IN. One of the input fields provides the control key value for the primary file read in step 4. See the record layout labeled TAPE-IN-REC in “[Example of IDENTIFICATION, ENVIRONMENT, and DATA divisions](#)” on page 274. When TAPE-IN reaches its end, the task skips to its normal termination processing at step 12.
4. READ-PRIMARY—The task reads the primary record whose key was read in step 3. The elements read are named under data name PRIMARY-DATA-LIST in the WORKING-STORAGE SECTION (see “[Example of IDENTIFICATION, ENVIRONMENT, and DATA divisions](#)” on page 274). If a record with a requested key does not exist, the task terminates with an error message by skipping to step 9. If the record exists, the task prepares to read the subordinate chain in the related file.
5. READ-RELATED—The task reads a related record on the linkpath named CUSTLKOR. If an error occurs, the task terminates with an error message by skipping to step 10. If another record on the chain does not exist, the task loops back to step 3 to process the chain for another key value.
6. PROGRAM-PROCESSING—If a record was read in step 5, the task moves selected fields from that record to the output record and writes the output record to sequential file TAPE-OUT. The task then loops back to step 5 to read the next record in the same chain.

7. STATUS-DISPLAY-SINON—The task prints an error message reporting the failure of the **SINON** in step 1. It then skips to abnormal termination processing at step 11.
8. STATUS-DISPLAY-OPEN—The task prints an error message reporting the failure of the **OPENX** in step 2. It then skips to abnormal termination processing at step 11.
9. STATUS-DISPLAY-PRIM—The task prints an error message reporting the failure of the **READM** in step 4. It then skips to abnormal termination processing at step 11.
10. STATUS-DISPLAY-RELA—The task prints an error message reporting the failure of the **READV** in step 5. It then skips to abnormal termination processing at step 11.
11. ABEND-PARA—This is the abnormal termination processing. First, the task displays a console message. Then the task checks the state of the sequential files. If they are open, the task skips to step 12 to close them. If they are not open, the task skips to step 13, avoiding the close step. In both cases, the task proceeds with its normal termination processing.
12. NORMAL-EOJ—This is the beginning of the normal termination processing. The task closes the sequential input and output files.
13. FILE-CLOSE—The task closes the two PDM files. If you omit the **OPENX** in step 2, you should omit this step. If the **CLOSX** fails, the task generates an error message before continuing with termination processing.
14. SINOF—The task issues **SINOF**, then terminates. If the **SINOF** fails, the task generates an error message before terminating.

Sample coding sequence for read-only environment

The following example shows an implementation of the preceding analysis. The coding sections are numbered by item to match the preceding analysis.

```
PROCEDURE DIVISION.  
  
PROGRAM-INITIALIZATION.          (Item 1)  
MOVE 'SINON' TO FUNCTION.  
CALL 'DATBAS' USING FUNCTION, STATUS, ENDP.  
IF STATUS IS NOT EQUAL TO '****' GO TO STATUS-DISPLAY-SINON.  
  
PROGRAM-OPEN.                    (Item 2)  
MOVE 'OPENX' TO FUNCTION.  
MOVE 'CUSTREAD****,CORDREAD****,END.' TO ENTRIES.  
CALL 'DATBAS' USING FUNCTION, STATUS, REALM, ENDP.  
IF STATUS IS NOT EQUAL TO '****' GO TO STATUS-DISPLAY-OPEN.
```

Comment

All files in the REALM have been logically opened, but lock bytes have not been set because READ was used as the mode in the REALM parameter.

```
OPEN INPUT TAPE-IN.  
OPEN OUTPUT TAPE-OUT.  
MOVE 'X' TO IO-SWITCH.  
  
PROGRAM-MAINLINE.                (Item 3)  
    READ TAPE-IN AT END GO TO NORMAL-EOJ.  
    MOVE TAPE-CONTROL TO CONTROL-KEY.  
    MOVE 'READM' TO FUNCTION.  
  
READ-PRIMARY.                    (Item 4)  
    CALL 'DATBAS' USING FUNCTION, STATUS, PRIMARY-FILE,  
    CONTROL-KEY,  
    PRIMARY-DATA-LIST, PRIMARY-DATA-AREA, ENDP.  
    IF STATUS IS NOT EQUAL TO '****' GO TO STATUS-DISPLAY-PRIM.  
    MOVE 'READV' TO FUNCTION.  
    MOVE 'LKOR' TO REFERENCE.
```

Comment The primary record has been read. If found, the related read is set up. If not, control is passed to a special error handling routine.

```

READ-RELATED.                (Item 5)
    CALL 'DATBAS' USING FUNCTION, STATUS, RELATED-FILE,
    REFERENCE,
        LINKPATH, CONTROL-KEY, RELATED-DATA-LIST,
    RELATED-DATA-AREA, ENDP.

```

Comment After reading the related record, inspect to determine if the read was successful. If so, test the *reference* field to determine if the related end-of-chain has been reached.

```

IF STATUS IS NOT EQUAL TO '****' GO TO STATUS-DISPLAY-RELA.
IF REFERENCE IS EQUAL TO 'END.' GO TO PROGRAM-MAINLINE.

```

```

PROGRAM-PROCESSING.          (Item 6)
    MOVE CORD-CUSTOMER TO TO-NAME.
    MOVE CORD-VALUE TO TO-VALUE.
    WRITE TAPE-OUT-REC. MOVE SPACES TO TAPE-OUT-REC.
    GO TO READ-RELATED.

```

```

STATUS-DISPLAY-SINON.        (Item 7)
    DISPLAY 'SINON UNSUCCESSFUL - STATUS - ' STATUS.
    GO TO ABEND-PARA.

```

```

STATUS-DISPLAY-OPEN.         (Item 8)
    DISPLAY 'OPENX UNSUCCESSFUL - REALM - ' REALM.
    GO TO ABEND-PARA.

```

```

STATUS-DISPLAY-PRIM.         (Item 9)
    DISPLAY 'FUNCTION -' FUNCTION ' STATUS ' STATUS ' FILE '
    PRIMARY-FILE
        ' CONTROL-KEY ' CONTROL-KEY.
    GO TO ABEND-PARA.

```

```

STATUS-DISPLAY-RELA.         (Item 10)
    DISPLAY 'FUNCTION ' FUNCTION ' STATUS ' STATUS ' FILE '
    RELATED-FILE
        ' LINKPATH ' LINKPATH ' CONTROL-KEY ' CONTROL-KEY.
    GO TO ABEND-PARA.

```

```
ABEND-PARA.                                (Item 11)
  DISPLAY 'ABNORMAL EOJ'.
  IF IO-SWITCH IS EQUAL TO 'X',
  GO TO NORMAL-EOJ.
  GO TO FILE-CLOSE.

NORMAL-EOJ.                                (Item 12)
  CLOSE TAPE-IN TAPE-OUT.

FILE-CLOSE.                                (Item 13)
  MOVE 'CLOSX' TO FUNCTION.
  MOVE 'CUSTCOMP****,CORDCOMP****,END.' TO ENTRIES.
  CALL 'DATBAS' USING FUNCTION, STATUS, REALM, ENDP.
  IF STATUS IS EQUAL TO '****' GO TO SINOF.
  DISPLAY ' CLOSX UNSUCCESSFUL - REALM ' REALM.

SINOF.                                      (Item 14)
  MOVE 'SINOF' TO FUNCTION.
  CALL 'DATBAS' USING FUNCTION, STATUS, ENDP.
  IF STATUS IS EQUAL TO '****' GO TO END-OF-JOB.
  DISPLAY 'SINOF UNSUCCESSFUL - STATUS - ' STATUS.

END-OF-JOB.
  STOP RUN.
```

Example of update mode

Here is an analysis of the example program that follows:

1. **INITIALIZATION**—The first step of a task must be a **SINON**. If the **SINON** fails, the task displays an error message, then terminates by skipping to step 12.
2. **OPENX-PARA**—The task opens all PDM files available to its schema and environment description. The **OPENX** sets a mode of **SUPD** in anticipation of the updates in later steps. If the **OPENX** fails, the task displays an error message, then terminates by skipping to step 12. Otherwise, the task opens its sequential input file.
3. **PROGRAM MAINLINE**—The mainline loop reads a transaction record from sequential file **TAPE-IN**, performs the appropriate function routine (steps 4 through 11), then loops back to read another record. The record layout for **TAPE-IN** is after data name **TAPE-IN-REC** in the **FILE SECTION** in “**Example of IDENTIFICATION, ENVIRONMENT, and DATA divisions**” on page 274. The transaction code is in the field named **TAPE-CODE**. When **TAPE-IN** reaches its end, the loop terminates and the task skips to step 13 to complete its processing.
4. **PRIMARY-ADD**—This is the routine that adds a record to a primary file. The task builds the primary record, calls the general purpose routine (step 7) to issue an **ADD-M** command, then returns to the mainline loop.
5. **PRIMARY-DELETE**—This is the routine that deletes a record from a primary file. The task calls the general purpose routine (step 7) to issue a **READM** command, calls the general purpose routine again to issue a **DEL-M** command, then returns to the mainline loop. The **READM** verifies that the record exists, reads the record, and holds it for the **DEL-M**. If the **READM** fails, the task does not try the **DEL-M**.
6. **PRIMARY-CHANGE**—This is the routine that changes a record on a primary file. The task calls the general purpose routine (step 7) to issue a **READM** command, changes fields in the record, calls the general purpose routine again to issue a **WRITM** command, then returns to the mainline loop. The **READM** makes the record available for update, and the **WRITM** replaces the updated record on the file. If the **READM** fails, the task does not try the **WRITM**.

7. PRIMARY-CALL—This is the general purpose routine that issues DML commands for reading and writing primary records. The task issues the DML command set by the caller of the routine, then returns. If the command fails, the task prints an error message before returning.
8. RELATED-ADD—This is the routine that adds a record to the end of a chain in a related file. The task calls the general purpose routine (step 7) to issue a **READM** command, calls the other general purpose routine (step 11) to issue an **ADDVA**, then returns to the mainline loop. The **READM** verifies that a primary record with the appropriate key value exists. If such a record does not exist, its subordinate chain cannot exist; therefore, if the **READM** fails, the task does not try the **ADDVA**. The **READM** is optional because the **PDM** would read the appropriate primary record as part of the servicing of the **ADDVA**. The **PDM** would reject the **ADDVA** if the primary record did not exist.
9. RELATED-DELETE—This is the routine that deletes an entire chain from a related file. The task calls the general purpose routine (step 7) to issue a **READM** command, repeatedly calls the other general purpose routine (step 11) to issue **READV** commands followed by **DELVD** commands, then returns to the mainline loop. The **READM** verifies that a primary record with the appropriate key value exists. If such a record does not exist, its subordinate chain cannot exist; therefore, if the **READM** fails, the task does not try the **READVs** and **DELVDs**. The **READM** is optional because the **PDM** would read the appropriate primary record as part of the servicing of the first **READV**. The **PDM** would reject the **READV** if the primary record did not exist. When a **READV** sets **END.** in **REFERENCE**, the task has processed the entire chain; therefore, it immediately returns. Note how this handles the case of an empty chain.

Each successful **DELVD** sets **REFERENCE** to the **RRN** of the related record preceding the deleted record on the chain. Since each **DELVD** promotes the second record on the chain to the head of the chain, each **DELVD** sets **REFERENCE** to **LKOR**. The **READV** in the next iteration of the loop uses that value of **REFERENCE** to read the new head of the chain, and the following **DELVD** deletes it.

10. RELATED-CHANGE—This is the routine that changes a record on a related file. The task calls the general purpose routine (step 7) to issue a **READM** command, repeatedly calls the other general purpose routine (step 11) to issue **READV** commands, moves new field values from the input transactions to the appropriate related record, calls the second general purpose routine to issue a **WRITV** command, then returns to the mainline loop. The **READM** verifies that a primary record with the appropriate key value exists. If such a record does not exist, its subordinate chain cannot exist; therefore, if the **READM** fails, the task does not try the **READVs** and **WRITV**.

Each **READV** reads another record on the identified linkpath subordinate to the appropriate primary record. You must add logic to inspect each related record and update only the correct one in the chain. If a **READV** sets **END.** in **REFERENCE** before the logic selects a record, the routine prints an error message and returns to the mainline loop without updating a record. Note how this handles the cases of both an empty chain and a chain that does not contain a matching record.

Each successful **READV** sets **REFERENCE** to the **RRN** of the record just read. If the task updates a record, the **WRITV** uses that value of reference to replace the related record.

11. RELATED-CALL—This is the general purpose routine that issues **DML** commands for reading and writing related records. The task issues the **DML** command set by the caller of the routine, then returns. If the command fails, the task prints an error message before returning.
12. ABNORMAL-EOJ—This is the abnormal termination processing. First, the task displays a console message. Then the task checks the state of **TAPE-IN**. If it is open, the task skips to step 13 to close it. If it is not open, the task skips to step 14, avoiding the close step. In both cases, the task proceeds with its normal termination processing.
13. TAPE-EOJ—This is the beginning of the normal termination processing. The task closes the **TAPE-IN** input file.
14. END-CONTROL—The task closes the **PDM** files it opened in step 2. If the **CLOSX** fails, the task generates an error message before continuing with termination processing.
15. END-SINOF—The task issues **SINOF**, then terminates. If the **SINOF** fails, the task generates an error message before terminating.

Sample coding sequence for update environment

The following example shows an implementation of the preceding analysis. The coding sections are numbered by item to match the preceding analysis.

```
PROCEDURE DIVISION.

INITIALIZATION.                (Item 1)
MOVE 'SINON' TO FUNCTION.
CALL 'DATBAS' USING FUNCTION, STATUS, ENDP.
IF STATUS IS EQUAL TO '*****' GO TO OPENX-PARA.
DISPLAY 'SINON' STATUS.
GO TO ABNORMAL-EOJ.

OPENX-PARA.                    (Item 2)
MOVE 'ALL.SUPD****,END.' TO ENTRIES.
MOVE 'OPENX' TO FUNCTION.
CALL 'DATBAS' USING FUNCTION, STATUS, REALM, ENDP.
IF STATUS IS EQUAL TO '*****' GO TO FILE-INITIALIZE.
DISPLAY 'OPENX UNSUCCESSFUL' STATUS.
GO TO ABNORMAL-EOJ.
FILE-INITIALIZE.
OPEN INPUT TAPE-IN.
MOVE 'X' TO IO-SWITCH.

PROGRAM-MAINLINE.            (Item 3)
READ TAPE-IN AT END
GO TO TAPE-EOJ.
MOVE TAPE-CONTROL TO CONTROL-KEY.
IF TAPE-CODE = '01'
GO TO PRIMARY-ADD.
IF TAPE-CODE = '02'
GO TO PRIMARY-DELETE.
IF TAPE-CODE = '03'
GO TO PRIMARY-CHANGE.
IF TAPE-CODE = '04'
```

```
GO TO RELATED-ADD.  
IF TAPE-CODE = '05'  
GO TO RELATED-DELETE.  
IF TAPE-CODE = '06'  
GO TO RELATED-CHANGE.  
GO TO PROGRAM-MAINLINE.
```

```
PRIMARY-ADD. (Item 4)
```

Comment At this point you would move data from the input record to the PRIMARY-DATA-AREA.

```
MOVE 'ADD-M' TO FUNCTION.  
PERFORM PRIMARY-CALL THRU PRIMARY-CALL-X.  
GO TO PROGRAM-MAINLINE.
```

```
PRIMARY-DELETE. (Item 5)  
MOVE 'READM' TO FUNCTION.  
PERFORM PRIMARY-CALL THRU PRIMARY-CALL-X.  
IF STATUS NOT = '****' GO TO PROGRAM-MAINLINE.  
MOVE 'DEL-M' TO FUNCTION.  
PERFORM PRIMARY-CALL THRU PRIMARY-CALL-X.  
GO TO PROGRAM-MAINLINE.
```

```
PRIMARY-CHANGE. (Item 6)  
MOVE 'READM' TO FUNCTION.  
PERFORM PRIMARY-CALL THRU PRIMARY-CALL-X.  
IF STATUS NOT = '****' GO TO PROGRAM-MAINLINE.
```

Comment At this point the program updates the primary file data elements by moving the input fields to the fields in PRIMARY-DATA-AREA.

```
MOVE 'WRITM' TO FUNCTION.  
PERFORM PRIMARY-CALL THRU PRIMARY-CALL-X.  
GO TO PROGRAM-MAINLINE.  
  
PRIMARY-CALL. (Item 7)  
CALL 'DATBAS' USING FUNCTION, STATUS,  
PRIMARY-FILE, CONTROL-KEY,  
  
PRIMARY-DATA-LIST, PRIMARY-DATA-AREA, ENDP.  
IF STATUS IS EQUAL TO '****' GO TO PRIMARY-CALL-X.  
DISPLAY 'FUNCTION ' FUNCTION ' STATUS ' STATUS ' FILE '  
PRIMARY-FILE ' CONTROL-KEY ' CONTROL-KEY.  
PRIMARY-CALL-X.  
EXIT.
```

```
RELATED-ADD. (Item 8)  
MOVE 'READM' TO FUNCTION.  
PERFORM PRIMARY-CALL THRU PRIMARY-CALL-X.  
IF STATUS IS NOT EQUAL TO '****' GO TO PROGRAM-MAINLINE.  
MOVE 'ADDVA' TO FUNCTION.  
MOVE 'LKOR' TO REFERENCE.  
PERFORM RELATED-CALL THRU RELATED-CALL-X.  
GO TO PROGRAM-MAINLINE.
```

```
RELATED-DELETE. (Item 9)  
MOVE 'READM' TO FUNCTION.  
PERFORM PRIMARY-CALL THRU PRIMARY-CALL-X.  
IF STATUS IS NOT EQUAL TO '****' GO TO PROGRAM-MAINLINE.
```

Comment This value of REFERENCE directs **READV** to read the first related record on the chain of master record CONTROL-KEY.

```
MOVE 'LKOR' TO REFERENCE.
```

Comment The following loop deletes an entire chain in the related file.

```
RELATED-DEL-READ.
MOVE 'READV' TO FUNCTION.
PERFORM RELATED-CALL THRU RELATED-CALL-X.
```

Comment Check status then reference.

```
IF STATUS IS NOT EQUAL TO '****' GO TO PROGRAM-MAINLINE.
IF REFERENCE IS EQUAL TO 'END.' GO TO PROGRAM-MAINLINE.
MOVE 'DELVD' TO FUNCTION.
PERFORM RELATED-CALL THRU RELATED-CALL-X.
IF STATUS NOT EQUAL TO '****' GO TO PROGRAM-MAINLINE.

GO TO RELATED-DEL-READ.

RELATED-CHANGE.                    (Item 10)
MOVE 'READM' TO FUNCTION.
PERFORM PRIMARY-CALL THRU PRIMARY-CALL-X.
IF STATUS NOT EQUAL TO '****' DISPLAY 'UNABLE TO CHANGE RELATED
RECORD'
GO TO PROGRAM MAINLINE.

MOVE 'LKOR' TO REFERENCE.

RELATED-CHG-READ.
MOVE 'READV' TO FUNCTION.
PERFORM RELATED-CALL THRU RELATED-CALL-X.
```

Comment Check status then reference.

```
IF STATUS NOT EQUAL TO '****' GO TO PROGRAM-MAINLINE.
IF REFERENCE IS EQUAL TO 'END.' DISPLAY 'END OF CHAIN REACHED'
GO TO PROGRAM-MAINLINE.
```

Comment Perform a test to identify which related record should change and loop to RELATED-CHG-READ on no match. Otherwise, update the found record.

```

MOVE TAPE-TERMS TO CORD-TERMS.
MOVE 'WRITV' TO FUNCTION.
PERFORM RELATED-CALL THRU RELATED-CALL-X.
IF STATUS NOT EQUAL TO '*****' DISPLAY 'RELATED CHANGE
  UNSUCCESSFUL'.
GO TO PROGRAM-MAINLINE.

RELATED-CALL.                                (Item 11)
CALL 'DATBAS' USING FUNCTION, STATUS, RELATED-FILE, REFERENCE,
  LINKPATH,
CONTROL-KEY, RELATED-DATA-LIST, RELATED-DATA-AREA, ENDP.
IF STATUS = '*****' GO TO RELATED-CALL-X.
DISPLAY 'FUNCTION ' FUNCTION ' STATUS ' STATUS ' FILE '
  RELATED-FILE.
RELATED-CALL-X.
EXIT.

ABNORMAL-EOJ.                                (Item 12)
DISPLAY 'ABNORMAL JOB TERMINATION'
IF IO-SWITCH IS EQUAL TO 'X' GO TO TAPE-EOJ.
GO TO END-CONTROL.

TAPE-EOJ.                                    (Item 13)
CLOSE TAPE-IN.

END-CONTROL.                                 (Item 14)
MOVE 'CLOSX' TO FUNCTION.
MOVE 'ALL.COMP****,END.' TO ENTRIES.
CALL 'DATBAS' USING FUNCTION, STATUS, REALM, ENDP.
IF STATUS IS EQUAL TO '*****' GO TO END-SINOF.
DISPLAY 'CLOSX UNSUCCESSFUL - STATUS' STATUS
GO TO END-SINOF.

END-SINOF.                                   (Item 15)
MOVE 'SINOF' TO FUNCTION.
CALL 'DATBAS' USING FUNCTION, STATUS, ENDP.
IF STATUS IS NOT EQUAL TO '*****' DISPLAY 'FUNCTION ' FUNCTION
  ' STATUS ' STATUS.
END-ALL.
STOP RUN.

```

Example of recoverable update mode

This example adds recoverability and restartability to the previous example. Because most of the processing is identical, only the differences are discussed here. The changes fall into two categories: preparing the program for failure and handling a failure.

To prepare for a failure, the program issues a **COMIT** command between consecutive logical units of work (item 18). **COMIT** directs the PDM to complete I/O pending on behalf of this task, to release resources held by this task, and to identify a recovery point for this task. In addition, the **COMIT** preserves three items of application-dependent data: the number of the input record the task has processed in the preceding logical unit of work, the image of that record, and a switch indicating whether the task has opened its PDM files (item 17). This data is important when the task handles a failure.

In a task logging environment, when a task issues **OPENX**, the PDM prevents other tasks from accessing the opened files until the first task issues **COMIT**. In this example, the task allows other tasks to access the PDM files it has opened when it issues **COMIT** before reading its first input record. Subsequent **COMIT**s merely separate consecutive logical units of work.

When a failure occurs, the operating system terminates the task. When you restart the task, the PDM requires that for recovery purposes, the task execute in the same environment. In other words, you must use the same task identifier, the same interface identifier, the same job name, the same CSIPARM file, and so on. In addition, this program requires that you supply the same input file. Do not remove the input records successfully processed before the failure.

To handle the failure, the task checks for **RSTR** status returned after **SINON** (item 16). The **RSTR** status indicates that the PDM recognizes that your task failed earlier and is now attempting to restart.

The program directs that after your task receives RSTR in response to a **SINON**, the next DML command your task issues is a **RESET** (item 19). **RESET** with a commit identifier of **LAST** directs the PDM to recover the PDM files to their state at the last **COMIT** completed before the failure. The PDM reopens files as appropriate. (Any uncommitted updates were recovered at the time of task failure, or if the PDM failed, at PDM warm start.) In this example, **RESET** also retrieves the application-dependent data saved by that **COMIT**.

The data item **COMIT-IO-SWITCH** indicates whether the PDM files were open at the time of the failure (item 21). If the task failed before its first **COMIT**, **RESET** cannot retrieve any application-dependent data. In this case, **COMIT-IO-SWITCH** contains its initial value indicating that the PDM has not reopened the files and the task must do so. If the task failed after it issued its first **COMIT**, **RESET** retrieves application-dependent data indicating that the PDM has reopened the files. The task branches back to its initialization logic and continues as if it were executing for the first time.

If the task failed between its first and second **COMITs**, **RESET** retrieves zero in **COMIT-TAPE-COUNT** (item 20). This indicates that the task was processing its first input record at the time of the failure and it must reread and reprocess that first record. Also, the task branches back to its initialization logic.

If the task failed after its second or any subsequent **COMIT**, **RESET** retrieves a positive value in **COMIT-TAPE-COUNT** (item 20). The task must read and not process that number of records because those records were successfully processed before the failure. After reading the last such record, the task must verify that the record matches the record image retrieved by **RESET** (item 22). The match succeeds when the record is the last one successfully processed before the failure. In this case, the task resumes its mainline processing to read and process the next record. If the match fails, you have probably not supplied the same input file. If the program signs off, you cannot resubmit the job, so it stops the run immediately without a commit. Resubmit the job with the correct input file.

Notice that the program also terminates immediately during a restart if the **RESET** fails. This is an unusual method of terminating a task that processes PDM files. The usual method of terminating invokes closing files (**CLOSX**) and signing off (**SINOF**). If the task successfully signs off and you resubmit the job, the subsequent **SINON** receives a ******** status. Following an unsuccessful termination, the **SINON** executed on a second or subsequent restart again receives a **RSTR** status and the subsequent **RESET** restarts the task from the same commit point with the same application-dependent data.

Sample coding sequence for recoverable update environment

The following example is a repeat of the previous example with one exception. This example shows the use of **COMIT** and **RESET** in an update environment. This technique makes the program restartable.

```

PROCEDURE DIVISION.

INITIALIZATION.                                     (Item 1)
    MOVE 'SINON' TO FUNCTION.
    CALL 'DATBAS' USING FUNCTION, STATUS, ENDP.
    IF STATUS IS EQUAL TO '****' GO TO OPENX-PARA.
    IF STATUS IS EQUAL TO 'RSTR' GO TO PROGRAM-RESTART.
                                                    (Item 16)
    DISPLAY 'SINON' STATUS.
    GO TO ABNORMAL-EOJ.

OPENX-PARA.                                         (Item 2)
    MOVE 'ALL.SUPD****,END.' TO ENTRIES.
    MOVE 'OPENX' TO FUNCTION.
    CALL 'DATBAS' USING FUNCTION, STATUS, REALM, ENDP.
    IF STATUS IS EQUAL TO '****' GO TO FILE-INITIALIZE.
    DISPLAY 'OPENX UNSUCCESSFUL' STATUS.
    GO TO ABNORMAL-EOJ.

FILE-INITIALIZE.
    OPEN INPUT TAPE-IN.
    MOVE 'X' TO IO-SWITCH.
    MOVE IO-SWITCH TO COMIT-IO-SWITCH.             (Item 17)

PROGRAM-MAINLINE.                                   (Item 3)
    MOVE 'COMIT' TO FUNCTION.                       (Item 18)

```

```
MOVE TAPE-COUNT TO COMIT-TAPE-COUNT.
MOVE TAPE-IN-REC TO COMIT-TAPE-IN-REC.
CALL 'DATBAS' USING FUNCTION, STATUS, COMIT-ID,
COMIT-LENGTH,
    COMMIT-DATA, ENDP.
PROGRAM-MAIN-RESTART
READ TAPE-IN AT END
GO TO TAPE-EOJ.
ADD 1 TO TAPE-COUNT.
MOVE TAPE-CONTROL TO CONTROL-KEY.
IF TAPE-CODE = '01'
    GO TO PRIMARY-ADD.
IF TAPE-CODE = '02'
    GO TO PRIMARY-DELETE.
IF TAPE-CODE = '03'
    GO TO PRIMARY-CHANGE.
IF TAPE-CODE = '04'
    GO TO RELATED-ADD.
IF TAPE-CODE = '05'
    GO TO RELATED-DELETE.
IF TAPE-CODE = '06'
    GO TO RELATED-CHANGE.
GO TO PROGRAM-MAINLINE.
```

PRIMARY-ADD.

(Item 4)

Comment At this point the primary file data elements are updated by moving the input fields to the fields in PRIMARY-DATA-AREA.

```

MOVE 'WRITM' TO FUNCTION.
PERFORM PRIMARY-CALL THRU PRIMARY-CALL-X.
GO TO PROGRAM-MAINLINE.

PRIMARY-CALL.                                     (Item 7)
  CALL 'DATABS' USING FUNCTION, STATUS, PRIMARY-FILE,
  CONTROL-KEY,
  PRIMARY-DATA-LIST, PRIMARY-DATA-AREA, ENDP.
  IF STATUS IS EQUAL TO '****' GO TO PRIMARY-CALL-X.
  DISPLAY 'FUNCTION ' FUNCTION ' STATUS ' STATUS ' FILE '
  PRIMARY-FILE ' CONTROL-KEY ' CONTROL-KEY.
PRIMARY-CALL-X.
EXIT.

RELATED-ADD.                                       (Item 8)
  MOVE 'READM' TO FUNCTION.
  PERFORM PRIMARY-CALL THRU PRIMARY-CALL-X.
  IF STATUS IS NOT EQUAL TO '****' GO TO PROGRAM-MAINLINE.
  MOVE 'ADDVA' TO FUNCTION.
  MOVE 'LKOR' TO REFERENCE.
  PERFORM RELATED-CALL THRU RELATED-CALL-X.
  GO TO PROGRAM-MAINLINE.

RELATED-DELETE.                                   (Item 9)
  MOVE 'READM' TO FUNCTION.
  PERFORM PRIMARY-CALL THRU PRIMARY-CALL-X.
  IF STATUS IS NOT EQUAL TO '****' DISPLAY 'UNABLE TO DELETE
  RELATED'
  GO TO PROGRAM-MAINLINE.

```

Comment This value of REFERENCE directs READV to read the first record on the chain.

```

MOVE 'LKOR' TO REFERENCE.

RELATED-DEL-READ.
  MOVE 'READV' TO FUNCTION.
  PERFORM RELATED-CALL THRU RELATED-CALL-X.

```


Comment Check status then reference.

```

IF STATUS NOT EQUAL TO '****' GO TO PROGRAM-MAINLINE.
IF REFERENCE IS EQUAL TO 'END.' DISPLAY 'END OF CHAIN REACHED'
GO TO PROGRAM-MAINLINE.

MOVE TAPE-TERMS TO CORD-TERMS.
MOVE 'WRITV' TO FUNCTION.
PERFORM RELATED-CALL THRU RELATED-CALL-X.
IF STAT NOT EQUAL TO '****' DISPLAY 'RELATED CHANGE
UNSUCCESSFUL'.
GO TO PROGRAM-MAINLINE.

RELATED-CALL.                                     (Item 11)
  CALL 'DATBAS' USING FUNCTION, STATUS, RELATED-FILE, REFERENCE,
  LINKPATH,
  CONTROL-KEY, RELATED-DATA-LIST, RELATED-DATA-AREA, ENDP.
  IF STATUS = '****' GO TO RELATED-CALL-X.
  DISPLAY 'FUNCTION ' FUNCTION ' STATUS ' STATUS ' FILE '
  RELATED-FILE.
RELATED-CALL-X.
  EXIT.
PROGRAM-RESTART.                                  (Item 19)
  DISPLAY 'PROGRAM RESTART IN PROGRESS'
  MOVE 'RESET' TO FUNCTION.
  MOVE 'LAST' TO COMIT-ID.
  CALL 'DATBAS' USING FUNCTION, STATUS, COMIT-ID, COMIT-LENGTH,
  COMIT-DATA, ENDP.
  IF STATUS IS NOT EQUAL TO '****'
    DISPLAY 'FUNCTION ' FUNCTION ' STATUS ' STATUS
    STOP RUN.
  IF COMIT-TAPE-COUNT=ZERO                         (Item 20)
    IF COMIT-IO-SWITCH IS NOT EQUAL TO 'X'        (Item 21)
      GO TO FILE-INITIALIZE
    ELSE GO TO OPENX-PARA.
  MOVE COMIT-IO-SWITCH TO IO-SWITCH.

SKIP-FOR-RECORD.
  READ TAPE-IN AT END GO TO TAPE-EOJ.
  ADD 1 TO TAPE-COUNT.
  IF TAPE-COUNT IS NOT EQUAL TO COMIT-TAPE-COUNT
    DISPLAY 'SKIPPING FOR RESTART: ' TAPE-IN-REC.
  GO TO SKIP-FOR-RECORD.

```

```
CHECK-MATCH.                                     (Item 22)
  IF TAPE-IN-REC IS NOT EQUAL TO COMIT-TAPE-IN-REC
    DISPLAY 'MISMATCH IN RESTART INPUT TAPE'
    STOP RUN.
  DISPLAY 'RESTARTING WITH: ' TAPE-IN-REC.
  GO TO PROGRAM-MAIN-RESTART.

ABNORMAL-EOJ.                                    (Item 12)
  DISPLAY 'ABNORMAL JOB TERMINATION'
  IF IO-SWITCH IS EQUAL TO 'X' GO TO TAPE-EOJ.
  GO TO END-CONTROL.

TAPE-EOJ.                                        (Item 13)
  CLOSE TAPE-IN.

END-CONTROL.                                     (Item 14)
  MOVE 'CLOSX' TO FUNCTION.
  MOVE 'ALL.COMP****,END.' TO ENTRIES.
  CALL 'DATBAS' USING FUNCTION, STATUS, REALM, ENDP.
  IF STATUS IS EQUAL TO '****' GO TO END-SINOF.
  DISPLAY 'FUNCTION ' FUNCTION ' STATUS ' STATUS.
  GO TO END-SINOF.

END-SINOF.                                       (Item 15)
  MOVE 'SINOF' TO FUNCTION.
  CALL 'DATBAS' USING FUNCTION, STATUS, ENDP.
  IF STATUS IS NOT EQUAL TO '****' DISPLAY 'FUNCTION ' FUNCTION
' STATUS ' STATUS.
END-ALL.
  STOP RUN.
```

Example of primary serial processing

This example shows how to read every record in a primary file by sweeping the file serially with **RDNXT**. **RDNXT** reads the records in the order they are stored on disk.

1. **INITIALIZATION**—The first step of a task must be a **SINON**. If the **SINON** fails, the task terminates with an error message.
2. **OPEN**—The task opens the PDM file to prepare for the following serial sweep of the file. The **OPENX** sets a mode of EUPD to reserve the entire file for this task. This is recommended because if another task updates the file, this task might process some record twice or skip a record. If the **OPENX** fails, the task generates an error message and terminates by skipping to step 6.
3. **SERIAL-INITIAL**—The task sets **QUALIFIER** to **BEGN**. This value directs the first **RDNXT** command to read the first allocated record in the file.
4. **SERIAL-READ**—The task repeatedly issues **RDNXT** commands to read every record in the file in ascending order of disk addresses. Following a successful **RDNXT**, your task can inspect any fields in the record, but it is recommended that your task not update any record before completing the serial sweeps. Following this processing, the task loops back to issue another **RDNXT**.

RDNXT can be unsuccessful either because it has reached the end of the file or because it encounters an error. At the end of the file, the task terminates normally by skipping to step 5. On an error, the task terminates abnormally by generating an error message before skipping to step 5.

5. **CLOSE**—This is the beginning of normal termination processing or the beginning of abnormal termination processing following an error in step 4. The task closes the PDM file it opened in step 2. If the **CLOSX** fails, the task generates an error message before continuing with termination processing.
6. **SINOF**—This is the continuation of normal termination processing or the beginning of abnormal termination processing following an error in step 2. (A failure in step 2 means the file is not open so step 2 skips to this step to avoid the **CLOSX** in step 5.) The task issues **SINOF**, then terminates. If the **SINOF** fails, the task generates an error message before terminating.

Sample coding sequence for serial processing of primary files

PROCEDURE DIVISION.

INITIALIZATION. (Item 1)

MOVE 'SINON' TO FUNCTION.
CALL 'DATBAS' USING FUNCTION, STATUS, ENDP.
IF STATUS IS EQUAL TO '****' GO TO OPEN.
DISPLAY 'SINON FAILURE - STATUS ' STATUS.
DISPLAY 'ABNORMAL JOB TERMINATION'.
STOP RUN.

OPEN. (Item 2)

MOVE 'OPENX' TO FUNCTION.
MOVE 'CUSTEUPD****,END.' TO ENTRIES.
CALL 'DATBAS' USING FUNCTION, STATUS, REALM, ENDP.
IF STATUS = '****' GO TO SERIAL-INITIAL.
DISPLAY 'OPENX FAILURE - REALM ' REALM.
GO TO SINOF.

SERIAL-INITIAL. (Item 3)

MOVE 'RDNXT' TO FUNCTION.
MOVE 'BEGN SERIAL' TO QUALIFIER.

SERIAL-READ. (Item 4)

CALL 'DATBAS' USING FUNCTION, STATUS, PRIMARY-FILE, QUALIFIER,
PRIMARY-DATA-LIST, PRIMARY-DATA-AREA, ENDP.
IF STATUS IS EQUAL TO 'END.' GO TO CLOSE.
IF STATUS IS NOT EQUAL TO '****' DISPLAY 'RDNXT FAILURE
-STATUS ' STATUS
GO TO CLOSE.

Example of related serial processing (physical)

This program shows that the coding for serially reading a related file with **RDNXT** is nearly identical to the coding in the previous example for a primary file. The only difference is in the parameter fields of the CALL 'DATBAS' statement that executes the RDNXT command.

Sample coding sequence for serial processing of related files

```

PROCEDURE DIVISION.

INITIALIZATION.

MOVE 'SINON' TO FUNCTION.
CALL 'DATBAS' USING FUNCTION, STATUS, ENDP.
IF STATUS IS EQUAL TO '*****' GO TO OPEN.
DISPLAY 'SINON FAILURE - STATUS ' STATUS.
INITIAL-ABEND.
DISPLAY TASK 'TASK ABNORMALLY TERMINATED'.
STOP RUN.

OPEN.
MOVE 'OPENX' TO FUNCTION.
MOVE 'CORDREAD****,END.' TO ENTRIES.
CALL 'DATBAS' USING FUNCTION, STATUS, REALM, ENDP.
IF STATUS = '*****' GO TO SERIAL-INITIAL.
DISPLAY 'OPENX FAILURE STATUS ' STATUS 'REALM' REALM.
GO TO SINOF.

SERIAL-INITIAL.
MOVE 'RDNXT' TO FUNCTION.
MOVE 'BEGN' TO QUALIFIER.

SERIAL-READ.
CALL 'DATBAS' USING FUNCTION, STATUS, RELATED-FILE, QUALIFIER,
RELATED-DATA-LIST, RELATED-DATA-AREA, ENDP.
IF STATUS IS EQUAL TO 'END.' GO TO CLOSE.
IF STATUS IS NOT EQUAL TO '*****' DISPLAY 'RDNXT FAILURE - STATUS
' STATUS
GO TO CLOSE.

```

Comment At this point, you can do whatever data manipulation you desire.

```
CLOSE.  
  MOVE 'CLOSX' TO FUNCTION.  
  MOVE 'CORDCOMP****,END.' TO ENTRIES.  
  CALL 'DATBAS' USING FUNCTION, STATUS, REALM, ENDP.  
  IF STATUS IS EQUAL TO '****' GO TO SINOF.  
  DISPLAY FUNCTION REALM.  
  
SINOF.  
  MOVE 'SINOF' TO FUNCTION.  
  CALL 'DATBAS' USING FUNCTION, STATUS, ENDP.  
  IF STATUS IS EQUAL TO '****' GO TO JOB-END.  
  DISPLAY 'SINOF FAILURE - STATUS ' STATUS.  
  
JOB-END.  
  STOP RUN.
```

Example of related serial processing (logical)

The coding in this example of **RDNXT** is nearly the same as the previous example, but the processing is much different. It does not read the entire file. This example reads only those related records chained on a particular linkpath. Also, the program reads each chain in linkpath sets; that is, consecutive **RDNXT** commands read successive records in a chain subordinate to one primary record. The following analysis highlights only the differences between this example and the previous example.

1. **RDNXT-PREP**—The task sets **QUALIFIER** to 'BEGN CUSTLKOR'. This directs the first **RDNXT** command to read the first record on a chain linked on linkpath **CUSTLKOR**.
2. **RDNXT-READ**—This step performs all the same processing as the similar step in “[Example of related serial processing \(physical\)](#)” on page 303, but also checks for **ENDC** in **STATUS**. **ENDC** indicates that the preceding **RDNXT** read the last record on the chain and this **RDNXT** did not read a record. **ENDC** also indicates that **QUALIFIER** now contains the value that directs the next **RDNXT** to read the first record on the next chain of the same linkpath. The task immediately loops back to issue another **RDNXT**.

When the **END** is returned, it signals the end of all chains on this linkpath. This may not have been all records on the file if there are other linkpaths.

Sample coding sequence for serial processing of related files by chain

```

PROCEDURE DIVISION.

INITIALIZATION.
    MOVE 'SINON' TO FUNCTION.
    CALL 'DATBAS' USING FUNCTION, STATUS, ENDP.
    IF STATUS IS NOT EQUAL TO '****' GO TO ABEND-PARA.
    MOVE 'OPENX' TO FUNCTION.
    MOVE 'CORDEUPD****,END.' TO ENTRIES.
    CALL 'DATBAS' USING FUNCTION, STATUS, REALM, ENDP.
    IF STATUS IS EQUAL TO '****' GO TO RDNXT-PREP.
    GO TO ABEND-PARA.

RDNXT-PREP.                                (Item 1)
    MOVE 'RDNXT' TO FUNCTION.
    MOVE 'BEGN CUSTLKOR' TO QUALIFIER.

RDNXT-READ.                                (Item 2)
    CALL 'DATBAS' USING FUNCTION, STATUS, RELATED-FILE, QUALIFIER,
        RELATED-DATA-LIST, RELATED-DATA-AREA, ENDP.
    IF STATUS IS EQUAL TO '****'
        GO TO DATA-PROCESS.
    IF STATUS = 'ENDC'
        GO TO RDNXT-READ.
    IF STATUS IS EQUAL TO 'END.'
        GO TO CLOSE.
    GO TO ABEND-PARA.

DATA-PROCESS.

Comment At this point, you can do whatever data manipulation you desire.

    GO TO RDNXT-READ.

CLOSE.
    MOVE 'CLOSX' TO FUNCTION.
    MOVE 'CORDCOMP****,END.' TO ENTRIES.
    CALL 'DATBAS' USING FUNCTION, STATUS, REALM, ENDP.
    IF STATUS IS EQUAL TO '****' GO TO SINOF.
ABEND-PARA.
    DISPLAY 'ABNORMAL JOB TERMINATION'.
    DISPLAY 'FUNCTION - ' FUNCTION ' STATUS ' STATUS.

SINOF.
    MOVE 'SINOF' TO FUNCTION.
    CALL 'DATBAS' USING FUNCTION, STATUS, ENDP.
    IF STATUS NOT EQUAL '****'
        DISPLAY 'ABNORMAL JOB TERMINATION'
        DISPLAY 'FUNCTION - ' FUNCTION ' STATUS ' STATUS.
    STOP RUN.

```

Index

*

****, status code, in error
 handling 36
BIND, data list keyword 63
CODE, data list keyword 64
DATA, data list keyword 62
NONE, data list keyword 62
REST, data list keyword 63
*, in SHOWX 54
*CODE=xx
 compatibility 66
*CODE=xx, data list keyword 64
COMMON, data list keyword 66
*FILL=nn, data list keyword 64
*IGN status code and ENDLG
 command 114, 115
*IPO status code, READX
 command 188
*NXT status code, READX
 command 188
*PON status code, READX
 command 188

A

access mode, PDM 98, 146, 254,
 263
Add Primary command. See
 ADD-M command
Add Related After command. See
 ADDVA command
Add Related Before command.
 See ADDVB command
Add Related Continue command.
 See ADDVC command
Add Related Replace command.
 See ADDVR command
adding
 primary records 22
 related records 26
ADD-M command
 record holding 22, 42
 serial processing 25, 71, 128,
 156, 196
 syntax description 68

ADDVA command
 introduction 26, 29
 navigation 29
 record code processing 65
 record holding 42
 syntax description 72
ADDVB command
 introduction 26, 28
 navigation 28
 record code processing 65
 record holding 42
 syntax description 77
ADDVC command
 introduction 26
 navigation 26
 record code processing 65
 record holding 42
 syntax description 82
ADDVR command
 record code processing 65
 record holding 32, 42, 43
 syntax description 86
 to change record code or
 control key 32, 86
algorithm, hashing
 for primary RRN 22
 for primary run 201
application programs, linking 60
argument parameter, FINDX
 command 124
automatic record holding 42

B

BDAM files
 and WRITD 262
 primary serial records 23
 qualifier
 FINDX command 129, 133
 RDNXT command 157, 161
BEGN, in qualifier
 FINDX
 BDAM or ESDS related 129,
 133
 KSDS 130
 KSDS 159
 RDNXT
 BDAM or ESDS related 157,
 162
 READX 192
 SHOWX 37, 54, 222, 227

- BIND, in ****BIND**** 63
- binding
 - the data list, any command 63
 - the qualifier, READX 192
- blocking factors, establishing 51
- bound qualifier, READX 192
- buffers, tuning 51

- C**
- CALL statement, for DML
 - commands 60, 273
- cccc, qualifier context area
 - FINDX, KSDS 130
 - RDNXT, KSDS 158
 - READX 192
 - SHOWX 37, 54, 222, 227
- CERR status code 94, 98
- CICS applications, linking 59
- closing
 - primary and related files 20
 - secondary key (index) files 21
- closing files
 - at SINOF 97, 249
 - with PDM DML 48
- CLOSX command
 - file condition before and after 98
 - introduction 20
 - success factors 97
 - syntax description 93
 - with PDM DML 48
- COBOL, coding examples 273
- CODE, in ****CODE**** 64
- CODE, in ***CODE=xx** 64
- coded records
 - changing control key 32
 - changing record code 32
 - record code processing 64
- COMIT command
 - coding example 291
 - introduction 40
 - logical unit of work 40
 - passing data to TLF 102
 - syntax description 100
 - when opening and closing files 20, 97, 144
- command activity statistics 51, 207
- commit point
 - establish, COMIT command 40, 97, 100, 144, 291, 293
 - restore files, RESET command 197
 - restoring files using RESET 41, 46
- common data items, data list 66
- common prefix, statistics file records 210
- common statistics record prefix 207
- COMMON, in ****COMMON**** 66
- comparison operator, in FINDX 125
- complete, file close mode. See COMP
- context area, cccc qualifier SHOWX 53
- context area, qualifier, cccc
 - FINDX, KSDS 130
 - RDNXT, KSDS 158
 - READX 192
 - SHOWX 37, 222, 227
- control key, changing
 - in a related record 32
 - in related record 86
- counting records
 - KEYCOUNT, READX 194
 - using ****NONE**** 62
- cross-address space, central PDM, data length limitation
 - ADD-M 70
 - ADDVA 75
 - ADDVB 80
 - ADDVC 84
 - ADDVR 90
 - COMIT 102
 - FINDX 126
 - MARKL 138
 - QMARK 149
 - RDNXT 155
 - READD 167
 - READM 170
 - READR 176
 - READV 184
 - READX 195
 - RESET 199
 - WRITM 266
 - WRITV 271

- CSIPARM
 in task level recovery 46
 introduction 18
 MAXIO value. *See* cross-
 address space
 MAXPACKET value. *See* cross-
 address space
 with SINON parameters 252
- D**
- data area
 introduction 18, 59, 62
 length limitation. *See* cross-
 address space
 passing with COMMIT command
 102, 293
 retrieving with RESET
 command 200, 297
- data division, COBOL, coding
 example 273
- data list
 binding 63
 coded records 64
 compatibility 66
 inserting blanks 64
 introduction 18, 59
 keywords
 BIND 63
 CODE 64
 DATA 62
 NONE 62
 REST 63
 *CODE=xx 64
 COMMON 66
 *FILL=nn 64
 moving all fields 63
 moving no fields 62
- Data Manipulation Language
 (DML) 273
 status code returned from
 commands 36, 61, 67
- Data Manipulation Language
 (DML), PDM commands
 ADD-M 68
 ADDVA 72
 ADDVB 77
 ADDVC 82
 ADDVR 86
 CLOSX 93
 COMMIT 100
 DEL-M 104
 DELVD 107
 ENDLG 112
 ENDTO 116
 FINDX 121
 FREEX 137
 MARKL 138
 OPENX 140
 QMARK 149
 QUIET 151
 RDNXT 152
 READD 165
 READM 169
 READR 172
 READV 179
 READX 187
 RESET 197
 RQLOC 201
 RSTAT 203
 SHOWX 221
 SINOF 248
 SINON 250
 SINON, CICS 256
 SINON, TIX 258
 SINON, TOTAL 260
 WRITD 262
 WRITM 264
 WRITV 268
- DATA, in **DATA** 62
 data-area parameter 61
 data-area parameter, purpose 18
 database integrity
 logical unit of work 40, 291
 program recovery 46
 record holding 42
 system logging 49
- database navigation, illustrations
 adding related records 26
 deleting related records 33
 reading serially 25
- data-list parameter 61
 purpose 18
- DATBAS 59
 DATBASC 59
 deadlock situation 45
 deadly embrace 45
- Delete Primary command. *See*
 DEL-M command
- Delete Related Direct command.
See DELVD command

- DEL-M command
 - introduction 25
 - record holding 25, 42
 - serial processing 25, 106, 128, 156, 196
 - syntax description 104
 - DELVD command
 - and READX processing 196
 - effect with read commands 111, 168
 - in reverse delete operation 111, 168
 - introduction 33
 - navigation 33
 - record holding 33, 42
 - syntax description 107
 - diagnosing errors 36, 67, 221
 - direct read, of related record 30, 165
 - DIRECTION option, READX 190, 192
 - directory environment description 18, 46
 - directory environment
 - description. *See also* environment description
 - directory schema 18, 46, 251
 - DML. *See* Data Manipulation Language (DML)
- E**
- EMBR status code
 - deadly embrace 45
 - READX command 188
 - END
 - as delimiter, DML commands 43, 44
 - in qualifier parameter SHOWX 38, 53
 - END.
 - as delimiter, DML commands 61, 62
 - in qualifier parameter READX 192, 193 SHOWX 222, 227
 - in reference parameter READR 174 READV 182
 - in status parameter FINDX 122, 129 RDNXT 153, 157
 - ENDC, in status parameter FINDX 122, 133 RDNXT 153, 163
 - ENDLG command
 - syntax description 112
 - system logging 50
 - ENDS, in qualifier parameter 55 FINDX, KSDS 132 RDNXT, KSDS 160 SHOWX 38, 222, 227
 - ENDTO
 - in qualifier parameter, SHOWX 58
 - syntax description 116
 - environment description
 - CLOSOX command 93, 98
 - during warm recovery 46
 - ENDLG command 113, 115
 - introduction 18
 - OPENX command 140, 144
 - RSTAT command 206
 - SINON command 251
 - WRITD 263
 - errors, diagnosing 36, 67, 221
 - ESDS files
 - primary serial reads 23
 - qualifier FINDX command 123, 129, 133 RDNXT command 154, 157, 161
 - EUPD
 - file open mode
 - for OPENX command. *See* exclusive update (EUPD) upon closing. *See* exclusive update (EUPD). *See* exclusive update (EUPD)
 - file open mode. *See* exclusive update (EUPD)
 - introduction 19
 - exclusive update
 - general considerations 144
 - exclusive update (EUPD)
 - index files 21
 - introduction 19
 - updating files 19
 - Execution Statistics utility report 51
 - explicit record holding 43
 - extended memory usage 216, 233

F

failure, task 46
 fatal embrace 45
 file
 primary. *See* primary file
 related. *See* related file
 file statistics record 209, 217
 files
 closing
 CLOSX command 93
 SINOF command 248
 opening
 OPENX command 140
 resetting records, RESET 197
 searching for particular records
 FINDX command 121
 READX command 187
 statistics on 52, 217
 user
 adding records 22, 26
 closing
 introduction 20
 deleting records 25, 26
 locked. *See* locked files
 opening
 introduction 19, 21, 48
 reading records 23, 30
 resetting records (RESET)
 41, 46
 searching for particular
 records
 FINDX command 23
 READX command 30
 serial processing 23, 30
 updating records 25, 32
 using files, coding examples
 273
 FILL, in *FILL=nn 64
 find next record command. *See*
 FINDX command
 FINDX command
 argument parameter 124
 introduction 23, 30
 KSDS primary file 130
 qualifier
 BDAM or ESDS file 129, 133
 record code processing 65
 record holding 42
 syntax description 121

FNTF status code
 for CLOSX command 94, 96
 for OPENX command 141, 143
 free records command. *See*
 FREEEX command
 FREEEX command
 record holding 43
 syntax description 137
 function activity statistics 52, 209,
 212

H

hashing algorithm, for primary rrm
 22
 hashing algorithm, for primary
 RRN 201
 HELD status code, READX
 command 188
 hold records, discussion 42

I

I/O buffer allocation statistics 52
 ICOR status, unreleased context
 FINDX 132
 RDNXT 160
 READX 193
 index files
 introduction. *See also*
 secondary key
 initialization
 CICS compatibility SINON 256
 of task, SINON 250
 TIS compatibility SINON 258
 TOTAL compatibility SINON
 260
 initialization statistics record 207,
 211
 integrity, of database 40
 intended update (IUPD)
 cannot use for KSDS 142, 143
 general considerations 145
 index files 21
 introduction 21, 48
 OPENX 140
 updating files 20
 intended update(IUPD) upon
 closing 98

interface activity statistics 52
 IUPD, file open mode
 OPENX 140
 upon closing 98

K

key, control
 changing value in related
 record 32
 reading a related record 31
 KEY, control
 introduction 61
 KEY=control key, qualifier
 parameter
 reading with FINDX 24, 122,
 129
 reading with RDNXT 24, 152
 KEYCOUNT option 190
 keywords, for data list 62
 kkkk in qualifier, READX 192
 KSDS files, primary
 and SINOF 249
 cannot use IUPD open 142
 cannot use PART close 95
 primary serial reads 23
 qualifier
 FINDX command 123, 130
 RDNXT command 154, 158
 record sequence 23

L

linking CICS programs with
 DATBASC 59
 linkpathKEY=control-key 122
 LKxx, in reference parameter
 ADDVA 73
 ADDVB 78
 DELVD 109
 READR 174
 READV 182
 locked files
 and SINOF 249
 logging I/O statistics 52
 logging. See task logging. See
 system logging
 logical unit of work 40

M

MARKL
 syntax description 138
 MARKL command
 system logging 50
 MASK option, READX 191, 192,
 194
 MAXARG, FINDX 125
 MAXIO, CSIPARM file. See
 cross-address space
 MAXPACKET, CSIPARM file.
 See cross-address space
 MRNF status code 36
 multitask environment
 file closing 20
 file updates 20, 25
 record holding 42

N

navigation, database illustrations
 adding related records 26
 deleting related records 33
 reading serially 24
 NEXT
 RDNXT
 BDAM or ESDS related using
 linkpath 163
 KSDS 158
 READX 192
 SHOWX 222, 227
 NEXT, in qualifier
 FINDX
 BDAM or ESDS related using
 linkpath 135
 ESDS 130
 SHOWX 38, 57
 NONE, in ****NONE**** 62

O

OERR status code 141, 146
 opening
 files with PDM DML 48
 primary and related files 19
 secondary key (index) files 21

OPENX command
 file status condition before and after 146
 introduction 19
 success factors 146
 syntax description 140
 with PDM DML 48
 with secondary key (index) files 21
 OPENX-OPTION, environment
 description
 CLOSX 93, 98
 OPENX 140, 146
 operator, comparison in FINDX 125

P

Packet Size, data area limitation.
 See cross-address space
 PART, file close mode
 at SINOF 97
 in CLOSX command 95
 not for KSDS 95, 96
 PDM DML
 concepts 17
 system control 47
 with RDML 17
 PDM DML. See Data Manipulation Language (DML)
 PDM, retrieving statistical information 51, 53
 Physical Data Manager (PDM) 17
 physical fields, named in data list 61
ppppLKxx, in qualifier
 FINDX, related file 135, 136
 RDNXT, related file 163
 primary files
 adding records 22
 closing
 CLOSX command 93
 introduction 20, 48
 SINOF command 248
 deleting records 25
 opening
 introduction 19, 48
 OPENX command 140
 reading records 23

RESET command 41, 46, 197
 searching for particular records
 FINDX 121
 FINDX command 23
 READX 187
 serial processing 23
 updating records 25
 using files, coding examples 273
 program initialization, coding examples 273
 program recovery
 coding example 291
 discussion 40, 46

Q

QMARK command
 syntax description 149
 system logging 50
 qualifier
 FINDX command
 BDAM or ESDS primary files 129
 KSDS primary file 130
 related file 133
 RDNXT command
 BDAM or ESDS primary files 157
 KSDS primary files 158
 related files 161
 READX command 192
 SHOWX 54
 SHOWX command 37, 222, 227
 qualifier context area
 READX 193
 qualifier context area, *cccc*, releasing
 FINDS, KSDS 132
 RDNXT, KSDS 160
 SHOWX 37, 53, 222, 227
 qualifier values
 BEGN. See BEGN
 END. See END
 ENDC. See ENDC
 ENDS. See ENDS
 KEY=. See KEY=
 NEXT. See NEXT
ppppLKxx. See *ppppLKxx*
 REBD. See REBD

QUIET

- command syntax 151
- QUIET command
- system logging 50

R

RDML. See Relational Data Manipulation Language (RDML)

RDNXT command

- introduction 23, 30
- record code processing 65
- record holding 42
- syntax description 152

RDXT command

- qualifier
 - BDAM or ESDS file 157, 161
 - KSDS primary file 158

read found argument command.

See FINDX command

Read Next command. See

RDNXT

read primary command. See

READM command

read related direct command.

See READD command

read related forward command.

See READV command

read related reverse command.

See READR command

read using index command. See

READX command

read, file open mode

- index files 21
- introduction 19, 21, 48
- updating files 19

READ, file open mode

- for OPENX command 140
- general considerations 144
- OPENX 140
- upon closing 98

READD command

- in reverse delete operation 111, 168

introduction 30

- record code processing 65
- record holding 42
- syntax description 165

reading primary records 23

reading related records 30

READM command

- introduction 23
- record holding 42
- syntax description 169

READR command

- in reverse delete order 111, 168

introduction 30

- record code processing 65
- record holding 42
- syntax description 172

READV command

- introduction 30
- qualifier 192
- record code processing 65
- record holding 42
- syntax description 179

READX command

- index files 21
 - introduction 23, 30
 - record holding 42
 - releasing context 193
 - syntax description 187
- REBD, in qualifier, READX 23, 30, 192, 193
- record codes in related files
 - changing, ADDVR command 32, 86
 - keyword data list processing 64
 - record contention statistics 52
 - record count, READX KEYCOUNT 190
 - record counting
 - **NONE** 62

record holding

- additional
 - during primary record add 22
 - during primary record delete 25
 - during related record add 26
 - during related record delete 33
- automatic for some commands 42
- database integrity 42
- explicit
 - END., 43
- logical unit of work 40
- preventing deadly embrace 45
- required in multi-task and task logging 43
- uncommitted, by PDM 42, 44

- record layouts, RSTAT statistics
 - file 207, 209
- record processing, special data
 - lists 62
- records
 - adding 22, 26
 - coded. *See* coded records
 - committing 40
 - counting with READX
 - KEYCOUNT 190
 - deleting 25, 33
 - finding primary RRN, RQLOC
 - command 201
 - finding specific secondary keys,
 - READX command 193
 - finding specific values, FINDX
 - 23, 31
 - finding specific values, FINDX
 - command 121
 - holding, freeing 42
 - reading 23, 30
 - relinking related, ADDVR 32
 - relinking related, ADDVR
 - command 86
 - restoring from update, RESET
 - 41, 46
 - restoring from update, RESET
 - command 197
 - searching for, FINDX 23
 - searching for, FINDX command
 - 31, 121
 - statistics 51
 - updating (rewriting/changing)
 - 25, 32
- recoverability, of SHOWX
 - command 223, 230
- recovery
 - system level 46, 49, 262
 - task level
 - coding example 291
 - establish commit point,
 - COMIT 40
 - establish commit point,
 - COMIT command 100
 - program recovery discussion
 - 46
 - restore to last commit,
 - RESET 41, 46
 - RSTR status code 41, 46,
 - 102, 197, 200, 250
- related files
 - adding records 26
 - changing record code or control
 - key 32
 - closing
 - CLOSX command 93
 - introduction 20, 48
 - SINOF command 248
 - deleting records 33
 - opening
 - introduction 19, 48
 - OPENX command 140
 - reading records 30
 - record code processing 64
 - resetting records, RESET 41,
 - 46
 - resetting records, RESET
 - command 197
 - searching for particular records
 - FINDX 31
 - FINDX command 121
 - READX command 193
 - sequentially processing 30
 - serial processing 30
 - updating records 32
 - using files, coding examples
 - 273
- Relational Data Manipulation
 - Language (RDML) 17, 41
- relative record number
 - assigning primary 22
 - assigning related 26
 - introduction 18
 - navigation 24, 26, 28, 29, 33
 - record code processing 64
 - related record delete 33
- Relative Record Number
 - and related record delete 111
 - find primary, RQLOC command
 - 201
 - obtaining for ADDVR 92
- releasing context areas. *See*
 - context area
- releasing held records 42
- Request Location command. *See*
 - RQLOC command
- RESET command
 - introduction 41
 - syntax description 197
 - usage after SINON RSTR
 - status 41, 46, 250
- REST, in ****REST**** 63

RESTART

- passing data with COMIT 102
- retrieving data with RESET 200
- SINON command 253
- reverse delete, related file 111, 168
- reverse READX 190
- RLSE, as delimiter, DML
 - commands 43
- RLSE, as delimiter, DML
 - commands 61
- RSTAT command
 - introduction 51
 - statistics written by 207, 209, 210
 - syntax description 203
- RSTR status code
 - coding example 291
 - from SINON 200, 250
 - introduction 41, 46

S

schema

- introduction 18
- RSTAT 207, 211
- SINON 251
- warm recovery 46
- searching for records
 - FINDX argument 121
 - READX *kkkk* 193
- SECKEY option, READX 189
- secondary key
 - index file 21
 - read records using 23, 30, 187
- sequential read
 - of related file 303, 305
 - of related files 30
- serial read
 - primary file 23
- serial read, FINDX, RDNXT, READX
 - illustration of primary RDNXT 24
 - RDNXT coding examples 300, 303, 305
 - related file 30
- serial-sequential read, of related file 31

- shared update (SUPD)
 - general considerations 144
 - index files 21
 - introduction 19
 - OPENX 140
 - upon closing 98
- SHOWX
 - data item list 231
- SHOWX command
 - for diagnosing status codes 36
 - for monitoring resources 53
 - syntax descriptions 221, 224
- shutting down central PDM 58
- shutting password, ENDTO 117
- signing on/off, coding examples 273
- sign-off command. See SINOFF command
- single-task environment
 - file closing 249
 - record holding 42
- SINOF command
 - syntax description 248
- SINOFF command
 - introduction 18
- SINON command
 - compatibility SINONs
 - CICS 256
 - TIS 258
 - TOTAL 260
 - in program recovery 46
 - introduction 18
 - syntax description 250
- ssss in qualifier, READX 192
- statistics
 - how to generate 206
 - retrieving with RSTAT command 51, 203, 221
 - retrieving with SHOWX command 51, 53
 - viewing online 51
 - written by RSTAT command 207, 210
- statistics file
 - and RSTAT command 51
 - how to incorporate 51

Statistics File
 common statistics record prefix
 207, 210
 file statistics record 209, 217
 how to incorporate 206
 initialization and termination
 records 207, 211
 record layouts 207, 209
 RSTAT command 203
 system statistics record 209,
 212
 Statistics Indicator parameter
 206
 statistics record prefix 207
 status code
 EMBR 45
 FNTR 96
 ICOR 132, 160, 193
 returned by every DML
 command 36, 61, 67
 status, file
 condition with CLOSX
 command 98
 condition with OPENX
 command 146
 SUPD
 introduction 19
 SUPD, file open mode
 OPENX 140
 upon closing. *See also* shared
 update
 suppressed logging for a file,
 SHOWX 241
 switching option, and ENDLG
 command 114, 115
 System Log File
 ENDLG command 112
 system logging 50
 system logging
 suppressed for a file 240
 using 49
 system statistics record 209, 212

T

task extended status, SHOWX
 37, 221
 task level recovery point,
 establish 40
 Task Log File
 establish commit point, COMIT
 40
 establish commit point, COMIT
 command 100
 logical unit of work 40
 program recovery 46
 system logging 49
 task logging
 logical unit of work 40
 program recovery 46
 read with explicit hold before
 file updates 25, 32
 record holding 42
 suppressed for a file 240
 system logging 49
 task sign-on statistics 52
 tasks, initializing with SINON 18,
 250
 CICS compatibility 256
 TIS compatibility 258
 TOTAL compatibility 260
 termination
 of PDM 58, 116
 of task
 coding examples 273
 SINOF command 18
 unsuccessful SINOF 293
 termination statistics record 207,
 209

U

- uncoded files, and related record
 - update 32
- uncommitted record holding 44
- UPDATE access mode, SINON 254
- update mode, program
 - coding example 283
- updating a primary file
 - adding records 22
 - deleting records 25
 - updating records 25
- updating a related file
 - adding records 26
 - deleting records 33
 - updating records 32

W

- WRITD command
 - syntax description 262
 - system logging 50
- Write Primary command. *See* WRITM command
- Write Related command. *See* WRITV command
- WRITM command
 - introduction 25
 - record holding 25, 43
 - serial processing 25
 - syntax description 264, 268
- WRITV command
 - introduction 32
 - record code processing 65
 - record holding 32, 43
 - updating a related record 32

X

- XA memory used 209, 216, 234